

HWI HAMBURG

HOCHSCHULÜBERGREIFENDER STUDIENGANG
WIRTSCHAFTSINGENIEURWESEN HAMBURG

Masterarbeit

Causal Machine Learning with Deep Learning approaches using the DoubleML Framework

mit den Schwerpunkten

- Implementation and Simulation Study

vorgelegt von

Moritz Sundermann

Matrikelnummer: 6896662

- Implementation and Case Study

vorgelegt von

Jan Teichert-Kluge

Matrikelnummer: 6906277

Bereich:	**Causal Machine Learning**
1. Gutachter:	**Prof. Dr. Martin Spindler**
2. Gutachter:	**Prof. Dr. Michael Merz**
vorgelegt am:	**02.05.2023**

AM LEHRANGEBOT BETEILIGTE HOCHSCHULEN:
Universität Hamburg
Hochschule für Angewandte Wissenschaften Hamburg
Helmut Schmidt Universität - Universität der Bundeswehr Hamburg

Contents

List of Figures IV

List of Tables VI

Abbreviations VIII

List of Symbols X

Allocation of Chapters XII

1 Introduction 1
- 1.1 Motivation and Relevance . . . 1
- 1.2 Current State of Research . . . 3
- 1.3 Objective . . . 5
- 1.4 Structure . . . 5

2 Theoretical Foundations 7
- 2.1 Causal Inference . . . 7
- 2.2 Causal Machine Learning . . . 12
- 2.3 DoubleML Framework . . . 14
  - 2.3.1 Problem Statement . . . 15
  - 2.3.2 Regularization Bias through a naive Approach . . . 16
  - 2.3.3 Overcoming the Regularization Bias by using Orthogonalization . 18
  - 2.3.4 Overcoming the Bias induced by Overfitting by using Sample Splitting 20
  - 2.3.5 Generalization . . . 22
  - 2.3.6 Inference in different Model Settings . . . 24
  - 2.3.7 Software Implementation . . . 25
  - 2.3.8 Other Causal Inference Frameworks and Implementations . . . 27
- 2.4 Deep Learning . . . 28
- 2.5 Text Data and their Use in Causal Inference . . . 31
- 2.6 Image Data and their Use in Causal Inference . . . 35

3 Implementation of Deep Learning Approaches in DoubleML Framework 37
- 3.1 Implementation using PyTorch/ Skorch . . . 38
- 3.2 Validation of Skorch Implementation . . . 39
- 3.3 Implementation and Validation using Keras/ Scikeras . . . 43

4 Simulation Study 45
- 4.1 Structure for Planning and Design of the Simulation Study . . . 46
- 4.2 Planning of the Simulation Study . . . 47
  - 4.2.1 Specify the Objectives of the Simulation Study . . . 48
  - 4.2.2 Specify a Data Generating Process . . . 48
  - 4.2.3 Define Estimands and Targets of the Simulation Study . . . 58
  - 4.2.4 Specify which Methods to be evaluated/ compared . . . 59
  - 4.2.5 Specify Performance Measures to be estimated . . . 62

4.2.6 Specify Number of Simulation Repetitions . . . 65
4.2.7 Overview Planning Steps . . . 65
4.3 Design of the Simulation Study . . . 66
4.4 Execution of the Simulation Study . . . 71
4.4.1 Exploratory Phase . . . 72
4.4.2 Analysis of the Exploratory Phase and Selection of a Data Generation Process for the Detail Phase . . . 73
4.4.3 Detail Phase . . . 80
4.4.4 Computational Demand and Performance . . . 81
4.5 Analysis of the Detail Phase of the Simulation Study . . . 85
4.5.1 Partially Linear Regression Scenarios . . . 86
4.5.2 Interactive Regression Model Scenarios . . . 91
4.6 Summary Simulation Study . . . 97

5 Use Cases for Causal Machine Learning with Deep Learning Approaches 100

6 Case Study 102
6.1 Problem Definition . . . 102
6.2 Dataset . . . 106
6.3 Naive Approach . . . 111
6.4 First Modelling Approaches . . . 111
6.4.1 Fine-tuning of BERT . . . 113
6.4.2 Fine-tuning of BEiT . . . 115
6.5 Multimodal Models . . . 117
6.5.1 Identify Opportunities . . . 118
6.5.2 Transformer Ensemble Fine-tuning . . . 120
6.6 Causal Inference . . . 124
6.6.1 Methods and Approaches . . . 124
6.6.2 First Analysis . . . 127
6.6.3 Full Analysis . . . 136
6.7 Summary Case Study . . . 145

7 Conclusion 147
7.1 Summary and Discussion of Results . . . 147
7.2 Limitations and Future Research . . . 149

References 151

Appendix XIII

List of Figures

1 Causal Graph . . . 7
2 Causal Inference Framework . . . 8
3 Causal Diagram of Partial Linear Regression Models . . . 16
4 Naive Machine Learning Estimation . . . 17
5 Double Machine Learning Estimation (no Sample Splitting) . . . 19
6 Debiased Estimation (no Sample Splitting) based on Partialling-out Approach 20
7 Double Machine Learning Estimation (Cross-fitting) . . . 21
8 Debiased Estimation (Cross-fitting) based on Partialling-out Approach . . 21
9 Class Structure of DoubleML Package . . . 26
10 Feed Forward Neural Network . . . 29
11 Multilayer Perceptron . . . 30
12 Causal Diagram with Text . . . 34
13 Class NeuralNetRegressorDoubleML . . . 39
14 Outline Phases of Simulation Study . . . 47
15 Packages and Modules in DMLSim . . . 66
16 Class network_builder . . . 67
17 Class simulation_study . . . 67
18 Process Diagram Simulation Study . . . 71
19 Coverage of 95 % Confidence Interval for Partial Linear Regression Scenarios in Exploratory Phase in % . . . 75
20 Relative Bias for Scenario A03 in % . . . 76
21 Coverage of 95 % Confidence Interval for Interactive Regression Model Scenarios in Exploratory Phase in % . . . 78
22 Relative Bias for Scenario A09 in % . . . 79
23 Workflow on the Hummel HPC Cluster . . . 85
24 Root Mean Squared Error for Partially Linear Regression Scenarios with $dim_x = 20$ . . . 87
25 Root Mean Squared Error for Partially Linear Regression Scenarios with $dim_x = 100$ . . . 88
26 Relative Bias for Partially Linear Regression Scenarios in % . . . 90
27 Coverage of 95 % Confidence Interval for Partially Linear Regression Scenarios in % . . . 91
28 Root Mean Squared Error for Interactive Regression Model Scenarios with $dim_x = 20$ . . . 93
29 Root Mean Squared Error for Interactive Regression Model Scenarios with $dim_x = 100$ . . . 93
30 Relative Bias for Interactive Regression Model Scenarios in % . . . 95
31 Coverage of 95 % Confidence Interval for Interactive Regression Model Scenarios in % . . . 96
32 Causal Graph Price Elasticity of Demand . . . 104
33 Negative Sales Rank over Price . . . 107
34 Logarithmic Negative Sales Rank over logarithmic Price . . . 108
35 Logarithmic Negative Sales Rank over logarithmic Price as binned Plot . . 108
36 Distribution of Numerical Variables . . . 109

37 Product Images . . . 110
38 Performance Metrics and Duration bert-base-uncased . . . 115
39 Performance Metrics and Duration beit-base-patch16 . . . 117
40 High-Level Multimodal Model Architecture . . . 119
41 Performance Metrics and Duration Multimodal Model . . . 123
42 Performance of Multimodal Model Ensemble true vs. predicted Values . . 124
43 Feature / Representation Learning Process . . . 125
44 High-Level Overview of the Usage of Embeddings in the DoubleML Model 126
45 Estimated Coefficient and Confidence Intervals for first Analysis . . . 130
46 Root Mean Squared Error values of the used Learners in the Causal Model for first Analysis . . . 131
47 Relationship between Output and Treatment Variable for first Analysis . . 132
48 Relationship between Output and Treatment Variable after mean-centring for first Analysis . . . 133
49 Relationship between Output and Treatment Variable after Residualizing for first Analysis . . . 135
50 Comparison between centred and residualized Relationships for first Analysis . . . 135
51 Estimated Coefficient and Confidence Intervals for full Analysis . . . 141
52 Root Mean Squared Error values of the used Learners in the Causal Model for full Analysis . . . 142
53 Relationship between Output and Treatment Variable for full Analysis . . 143
54 Relationship between Output and Treatment Variable after mean-centring for full Analysis . . . 143
55 Relationship between Output and Treatment Variable after Residualizing for full Analysis . . . 144
56 Combined Histogram with different Estimation Approaches . . . XIV
57 Histograms for Exploratory Phase Scenario A3 - Part 1 . . . XV
58 Histograms for Exploratory Phase Scenario A3 - Part 2 . . . XVI
59 Histograms for Exploratory Phase Scenario A9 - Part 1 . . . XVII
60 Histograms for Exploratory Phase Scenario A9 - Part 2 . . . XVIII
61 Performance Metrics and Duration bert-tiny . . . XIX
62 Performance Metrics and Duration distilbert-base-uncased . . . XX
63 Performance Metrics and Duration beit-large-patch16 . . . XXI
64 Performance Metrics and Duration tiny-vit-random . . . XXII

List of Tables

1 Fundamental Problem of Causal Inference . . . 9
2 Specification of Opossum Data Generating Processes . . . 54
3 Network Architectures . . . 61
4 Parameters for Training the Neural Networks . . . 62
5 Performance Measures . . . 63
6 Scenarios in Exploratory Phase of Simulation Study . . . 73
7 Relative Bias for Partial Linear Regression Scenarios in Exploratory Phase in % . . . 74
8 Relative Bias for Interactive Regression Model Scenarios in Exploratory Phase in % . . . 77
9 Scenarios in Detail Phase of Simulation Study . . . 81
10 Average Time when calling the fit() Method . . . 82
11 Root Mean Squared Error for Partially Linear Regression Scenarios in Detail Phase . . . 87
12 Relative Bias for Partially Linear Regression Scenarios in Detail Phase in % 89
13 Coverage of 95 % Confidence Interval for Partially Linear Regression Scenarios in Detail Phase in % . . . 91
14 Root Mean Squared Error for Interactive Regression Model Scenarios in Detail Phase . . . 92
15 Relative Bias for Interactive Regression Model Scenarios in Detail Phase in % 94
16 Coverage of 95 % Confidence Interval for Interactive Regression Model Scenarios in Detail Phase in % . . . 96
17 Used Features from Keepa Data . . . 106
18 Text Variables for Confounding Covariates . . . 109
19 First Rows of the transformed Dataset . . . 110
20 Descriptive Statistics for $\ln q$ and $\ln p$ . . . 111
21 BERT Models . . . 114
22 R^2 Score on Holdout Sample for BERT Models . . . 114
23 BEiT Models . . . 116
24 R^2 Score on Holdout Sample for BEiT Models . . . 116
25 R^2 Score on Holdout Sample for Multimodal Model . . . 124
26 Summary of DoubleMLPLR models with different learners . . . 129
27 Root Mean Squared Error values for considered learners in first Analysis . 130
28 Summary of Ordinary Least Squares Regression for first Analysis . . . 134
29 Price Elasticity of Demand for Naive and Causal Estimation for first Analysis 136
30 Summary of DoubleMLPLR models with different Learners for full Analysis 141
31 Root Mean Squared Error values for considered learners in full Analysis . 142
32 Summary of Ordinary Least Squares Regression for full Analysis . . . 144
33 Price Elasticity of Demand for Naive and Causal Estimation . . . 145
34 Overview of Key Steps in Planning of the Simulation Study . . . XXIII
35 Root Mean Squared Error for Partial Linear Regression Model Scenarios in Exploratory Phase . . . XXV
36 Average Standard Error for Partial Linear Regression Model Scenarios in Exploratory Phase . . . XXV

37 Root Mean Squared Error for Interactive Regression Model Scenarios in Exploratory Phase XXVI
38 Empirical Deviation for Interactive Regression Model Scenarios in Exploratory Phase XXVI
39 Average Standard Error for Interactive Regression Model Scenarios in Exploratory Phase XXVII

Abbreviations

ADEMP Aims, Data-generating mechanism, Methods, Estimands, and Performance measures

API Application Programming Interface

ATC Average Treatment Effect for the Controls

ATE Average Treatment Effect

ATT Average Treatment Effect for the Treated

BEiT Bidirectional Encoder Representations from Image Transformers

BERT Bidirectional Encoder Representations from Transformers

CI Confidence Interval

CML Causal Machine Learning

CNN Convolutional Neural Network

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

DAG Direct Acyclic Graphs

DGP Data Generating Process

DL Deep Learning

DML Double Machine Learning

ED Empirical Deviation

ELMo Embeddings from Language Models

GloVe Global Vectors

GPT Generative Pre-Training

GPU Graphics Processing Unit

HPC High Performance Computing

i.i.d. independent and identically distributed

IIVM Interactive IV Model

IRM Interactive Regression Model

IV Instrumental Variable

LATE Local Average Treatment Effect

LGBM Light Gradient-Boosting Machine

ML Machine Learning

MLP Multilayer Perceptron

MCTS Monte Carlo Tree Search

NLP Natural Language Processing

OLS Ordinary Least Squares

PLIV Partially Linear IV Regression

PLR Partially Linear Regression

ReLU Rectified Linear Unit

RMSE Root Mean Squared Error

RNN Recurrent Neural Network

SCP Secure Copy

SE Standard Error

SGD Stochastic Gradient Descent

SLURM Simple Linux Utility for Resource Management

SSH Secure Shell

SUTVA Stable Unit Treatment Value Assumption

ViT Vision Transformer

List of Symbols

Y	Outcome variable of interest (population-based)
y	Outcome variable of interest (sample-based)
$\hat{y}$	Estimated outcome variable of interest (sample-based)
D	Policy/ treatment variable of interest (population-based)
d	Policy/ treatment variable of interest (sample-based)
i	Individual units/ observations
$i \in I$	Individual observations in the main sample part
$i \in I^c$	Individual observations in the auxiliary sample part
n_1	Number of units that have received the treatment
n_2	Number of units that have not received the treatment
$X = (X_1, ..., X_{dim_x})$	Confounding covariates (population-based)
$x = (x_1, ..., x_{dim_x})$	Confounding covariates (sample-based)
dim_x	Number of confounding covariates
U	Stochastic error
V	Stochastic error
$\hat{V}$	Estimated stochastic error
N	Sample
n	Sample size
$\hat{\theta}_0$	Double Machine Learning estimator for θ_0
$\check{\theta}_0$	Naive estimator for θ_0
θ_0	True treatment effect parameter
θ	Treatment effect parameter, causal parameter in certain Data Generating Processes
η_0	True nuisance parameters
η	Nuisance parameters
$\hat{\eta}_0$	Estimated true nuisance parameters
m_0	True nuisance function
m	Nuisance function
g_0	True nuisance function
g	Nuisance function
$\hat{m}_0$	Estimated true nuisance function
$\hat{g}_0$	Estimated true nuisance function
$\mathcal{N}(0, 1)$	Standard normal distribution
ℓ_0	True nuisance function for partialling-out approach
ℓ	Nuisance function for partialling-out approach
φ	Score function
W	Random element
ψ	Neyman orthogonal score function
K	Number of folds
n_k	Sample size of each fold
$\hat{\theta}_{0,k}$	Double Machine Learning estimator for θ_0 for fold k
$\hat{\eta}_{0,k}$	Estimated true nuisance parameters for fold k
$\tilde{\theta}_0$	DML1 estimator for θ_0

Z	Instrumental variable
σ_{sig}	Sigmoid activation function
σ	Rectified Linear Unit activation function
n_{obs}	Sample size of a simulated dataset
n_{rep}	Number of repetitions/ simulation sample size
$j = 1, ..., n_{rep}$	Index of simulation repetitions
$\hat{\theta}_{0,j}$	Estimate of θ_0 in the jth repetition of the simulation
$\overline{\hat{\theta}_0}$	Average estimate of θ_0 over all simulation repetitions
p	Price of a good
q	Quantity demanded of a good
r_p	Residualized price of a good
r_q	Residualized quantity demanded of a good
α	Causal parameter in certain Data Generating Processes
a_0	Design parameter in certain Data Generating Processes
a_1	Design parameter in certain Data Generating Processes
b_0	Design parameter in certain Data Generating Processes
b_1	Design parameter in certain Data Generating Processes
s_1	Design parameter in certain Data Generating Processes
s_2	Design parameter in certain Data Generating Processes
c_y	Design parameter in certain Data Generating Processes
c_d	Design parameter in certain Data Generating Processes
β_j	Design vector in certain Data Generating Processes
b_j	Design vector in certain Data Generating Processes
R_y^2	Design parameter in certain Data Generating Processes
R_d^2	Design parameter in certain Data Generating Processes
ν	Design parameter in certain Data Generating Processes
γ	Design parameter in certain Data Generating Processes
$\mathcal{B}()$	Bernoulli Distribution
$\mathcal{U}(0, 1)$	Uniform Distribution
π	Numeric value for π
ρ	Design parameter in certain Data Generating Processes
α_p	Design parameter in certain Data Generating Processes
α_θ	Design parameter in certain Data Generating Processes
α_g	Design parameter in certain Data Generating Processes
β_g	Design parameter in certain Data Generating Processes
β_θ	Design parameter in certain Data Generating Processes
cov	Covariance
var	Variance
$\epsilon_{q,p}$	Price elasticity of demand
$\boldsymbol{\beta} = (\beta_0, \beta_1)$	Regression coefficients
ϵ	Error term
$\hat{\boldsymbol{\beta}}$	Esitmated regression coefficients
$\boldsymbol{X} = [\mathbf{1}_n, \boldsymbol{x}]$	Design matrix
$c()$	Context

Allocation of Chapters

Moritz Sundermann:
{1.2, 1.3, 2, 2.1, 2.2., 2.3.5, 2.3.6, 2.3.7, 2.3.8, 3, 3.3, 4, 4.1, 4.2, 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5, 4.2.6, 4.2.7, 4.3, 4.4, 4.4.1, 4.4.3, 4.5, 4.5.1, 4.5.2, 4.6, 6, 6.1, 7.1}

Jan Teichert-Kluge:
{1.1, 1.4, 2.3, 2.3.1, 2.3.2, 2.3.3, 2.3.4, 2.4, 2.5, 2.6, 3.1, 3.2, 4.4.2, 4.4.4, 5, 6.2, 6.3, 6.4, 6.4.1, 6.4.2, 6.5, 6.5.1, 6.5.2, 6.6, 6.6.1, 6.6.2, 6.6.3, 6.7, 7.2}

1 Introduction

1.1 Motivation and Relevance

Machine Learning (ML) is not a new phenomenon anymore. In fact, the first developments already date back several decades but recent success with impressive prediction capabilities has accelerated the adoption of ML approaches in the last years [109]. In empirical research, however, good predictive performance may not be enough. For example, if one would want to know what effect the administration of a particular drug has on a patient's health, it is necessary to know what would have happened if the drug had not been administered to the patient. A prediction task can estimate what health status a patient has given a specific set of circumstances [109]. Causal inference, on the other hand, investigates the effect of the intervention, i.e. the administration of a drug [109]. In general, predictive ML does not provide such insights about causal effects [60]. Furthermore, causal inference is fundamentally different from a prediction task [38], for which ML techniques are showing superior results compared to classical statistical methods.

The prominence of ML, fueled by the increasing availability of data and increasingly sophisticated learning methods, motivated some authors to argue that causal inference based on traditional theories has become less important for decision-making in business [60]. However, this position has become increasingly controversial in recent years by pointing out the risks that arise when business decisions are based solely on correlations without establishing any causal reasoning [60]. The debate spurred efforts to leverage the strengths and innovations of ML methods and apply them to causal inference [109]. The hereby emerging field of Causal Machine Learning (CML) aims at providing more precise, less biased and in the end more reliable estimators of causal effects [109].

As stated, one fundamental problem in statistical ML and in computer sciences in general, is about learning causal relations instead of correlations [67]. To answer questions about causal relationships systematically, it is necessary to decode these relationships among the different components of a system [67]. Analysing causal relations can be found in various different and diverse scientific disciplines. For example, some studies analyze the economic impact of potential interventions [35] and other studies infer causal relationships from time series in earth system science [94].

It is common in research to use observational studies for estimating the causal effect of a treatment on an outcome. Ideally, one would like to use randomized trials to infer the causal effect of a treatment. Such trials allow for the randomized allocation of a treatment and ensure that a treatment group and a control group are comparable [92]. Comparable groups facilitate the comparison of the average outcome of both groups to derive estimates of the treatment effect with minimized bias [92]. In practice, a randomized trial may not always be appropriate or possible to answer a research question. In the absence of randomized trials, researchers utilize observational data to estimate the effects of a treatment [19]. Yet, causal inference from observational data is seen as a core

problem in many scientific fields [124] and there are some fundamental challenges when it comes to drawing causal inference. On the one hand, the counterfactual outcomes can not be fully observed for each unit [81]. On the other hand, in most observational studies it is likely that the assignment of the treatment is somehow confounded [81]. Due to some imbalance in the confounders, it is challenging to evaluate causal inference in observational studies because these imbalances affect the treatment as well as the outcome [118].

Therefore, a central task of causal inference is to remove any confounding bias that would be present if the outcome would be naively compared in different treatment groups without any statistical adjustments [19]. The adjustment needed to remove the bias can be described in two steps. First, a set of variables to be adjusted for is selected, and hereafter a suitable causal inference algorithm is implemented to adjust for the selected variables [19]. While various statistical methods exist for causal inference under the condition of unconfoundedness given pretreatment covariates, there is no single optimal causal method that can be applied universally across different scenarios [81]. Furthermore, the number of covariates needs to be considered. On the one hand, the assumption of unfoundedness becomes more plausible when a larger number of pretreatment covariates are included [4]. On the other hand, standard methods for estimating the treatment effect may be adversely affected by a higher number of variables [4]. In such cases, ML methods might be a suitable approach to overcome the issues previously mentioned.

In fact, causal inference has a close relationship with the research field ML. Since ML boomed in recent years, also the development of causal inference methods has enhanced [122]. The field of CML, where ML and causal inference are combined, is an area of methodological research that is currently being actively explored [57]. Yet, most standard ML approaches fail to understand causal relationships between the components of a system [34]. Since, several authors already proposed to use ML methods to assess causal inference [4], [16], [61], [117]. In particular, powerful ML methods like decision trees or deep neural networks can be applied to estimate a potential outcome in a more accurate way and provide a new aspect on how to handle the confounders [122]. In particular, Deep Learning (DL) models have proven to be exceptionally powerful when it comes to prediction tasks [34], from which interesting and promising opportunities for CML arise.

New opportunities for statistical methods related to causal inference have emerged as the advent of Big Data [118]. Additionally, the increase in computing capabilities has favoured the use of especially DL methods [118]. It should be noted, however, that abundant data is not the only important aspect for causal inference. Although more data can be used to obtain more accurate, low-variance estimates, it is not necessarily associated with unbiased estimates [122]. Thus, while ML methods and especially DL methods enhance the development of causal inference, ML also benefits from causal inference in the sense that predictive accuracy alone is insufficient and that correctness and interpretability should also be sought by modern ML methods [122]. Therefore, leveraging the effectiveness of DL as well as the interpretability of causal inference is a promising direction [67].

With recently developed methods, new solutions for causal inference problems are available that can be scaled to larger sizes of problems [67]. Exemplary, new methods can be exploited to infer causal transcriptome networks with more than 10000 genes [62]. This can not be done by classical causal models in a reasonable time frame [67]. Moreover, such ML methods can be used in high-dimensional settings where the application of standard methods is not feasible [57]. Thus, the use of new approaches and methods from ML and especially DL will therefore be of great importance for future research in various scientific disciplines.

1.2 Current State of Research

This section provides an overview of the current state of research regarding the use of DL methods for CML applications. The gap in existing research should be highlighted to emphasize the value of this thesis.

This work contributes to several rapidly growing literature. In the following section and throughout the thesis, only the particularly relevant citations are given, however, further references and details can be found within these works.

As a primary reference, this thesis is referring to the double/ debiased machine learning framework and therefore the theoretical research provided by Chernozhukov et al. [17] as well as the software implementations of Bach et al. [7], [8]. The software implementation of the double/ debiased machine learning framework is available either as an R package or as a Python package and is called `DoubleML`. In the following thesis, the software implementation shall be described by the term Double Machine Learning (DML) package whilst the theoretical framework is referred to as DML framework. At present, the DML package can be used with several ML methods but DL approaches are not implemented yet. For other basic references, theoretical foundations, and software for DML it is referred to the literature set which is presented on the DML web page.

For an initial literature screening about this thesis' topic, combining CML with DL approaches, a keyword research on various distribution services was conducted. The following distribution services were researched for different combinations of the keywords "Deep Learning"/ "Neural Net" and "Causal Inference"/ "Causal Machine Learning" from November 1st to November 11th, 2023:

- *arXiv*
- *ResearchGate*
- *Springer*
- *Google Scholar*
- Digital Library of *Association for Computing Machinery*

The following paragraph should only highlight a few papers that seem to be of great relevance to this thesis. One paper that deals with the use of deep neural networks in

semiparametric inference is by Farell et al. [26]. In addition to theoretical explanations, the authors demonstrate the use of DL methods in an empirical application of mail marketing and also conduct a simulation study. In another paper, Yuan et al. [124] present and discuss a DL framework to infer causal interactions by transforming the input in an image-like representation before using a convolutional neural network for causality predictions. Besides, Whata et al. [118] evaluate the use of DL methods for causal inference and examine the use with simulation studies. In some other article, Luo et al. [67] describe causality in ML and outline how causal inference with DL could look like. Already a few years ago, Ramachandra [87] proposed the use of DL techniques for causal inference and for estimating treatment effects. Schwab et al. [98] provide a simplified method to learn representations for counterfactual inference with neural networks based on observational data. Besides, Goudet et al. [34] propose a new approach using neural networks for functional causal modelling from observational data. Last, but not least, Yao et al. [122] address the use of DL methods as part of their survey on causal inference.

In addition to the literature screening, the software development platform GitHub was used to search for further practical applications of DL methods in causal problem instances. In the above-mentioned time period, there were only a few applications to be found. The search revealed one collection of papers for the intersection of causal inference and Natural Language Processing (NLP) [112]. In this collection, there are several papers listed in which the topics influence, causality, and language are considered within case studies. But only a few papers use DL methods in their study to answer causal questions. Exemplary, Manzoor et al. [69] use neural models of language in the DML framework to control for unstructured texts. As another example, Weise et al. [116] use deep neural networks to isolate some individual speaking styles when modelling entrainment.

In addition to its use in language-related applications, Moraffah et al. [78] analyse the application of neural networks in the context of causal inference for time series.

Furthermore, Ramachandra [88] presents another example in which DL approaches are used for a causal inference problem. In this study, DL methods are used in a first step to preprocess the data before a causal model is applied.

Chernozhukov et al. [17] state in their paper introducing the DML framework that they have also experienced with DL methods in an empirical example but did not report on the results because of stability and computational issues.

Despite the above references, to the best of the author's knowledge, DL approaches have only been used once before for CML in the specific way that they are referring to the theoretical foundations of the DML framework [69]. However, other applications using DL methods in the DML framework are missing so far and practical issues still need to be investigated. No available software implementation for using DL methods in the DML framework or for the application in causal inference problem instances could be found.

Although there might be already some literature about the use of DL approaches in the context of CML or causal inference in general, there is no holistic review of the

application of these approaches. No comparison of the performance of DL methods in particular with other ML methods in causal applications was found in the literature review. As of now, there are only a handful of applications and case studies that directly use DL methods to answer causal questions or that consider DL approaches for causal problem instances at all.

1.3 Objective

As described in section 1.2 there is a clear gap in current research. There is no general software implementation for CML with DL approaches available. Furthermore, the use of DL methods in the context of the DML framework has not been studied holistically and only one paper sketches the use of these methods in their case study.

Therefore, this thesis aims at providing an object-oriented software implementation for using DL methods in the published DML package. The aim is to allow the use of DL approaches in the context of CML. By this, the DML package should be extended to a broader scope of applications so that future studies about causal problem instances can easily use DL methods in their investigation.

The objective of this work is not only to implement DL approaches as learners in the DML framework, but also to verify the implementation and to compare the DL methods with previously implemented ML methods using a simulation study. It is also intended to provide guidance on when DL methods should be used for causal problem instances. Finally, the practical applicability shall be demonstrated using a case study.

In summary, this thesis will contribute to the steadily growing literature of CML and focus on the promising opportunities of applying DL approaches in causal inference.

The contribution of this thesis to the current research can be summarized as follows:

- Develop an object-oriented software implementation for using DL methods in the DML package
- Conduct an extensive simulation study to verify the implementation, analyze the behaviour of DL methods in various data settings, and compare the DL methods with standard ML methods
- Present and discuss a case study where the application of DL methods in a real-world causal inference problem is described

1.4 Structure

At first, some theoretical foundations shall be provided in order to allow a holistic and comprehensive overview of the thesis topic. In this section, a short overview of causal inference in general and of CML is given before the DML framework is outlined. It is intended to only present the most important parts of this framework, for more details

it is referred to Chernozhukov et al. [17]. Further, the software implementation of the DML framework that sets the basis for this thesis is briefly presented. Additionally, the topic DL is presented in short with some typical approaches, strengths, and challenges. Already anticipating the case study, text and images as a dataform, and their use in causal inference are introduced.

After presenting those general theoretical principles, the implementation of DL approaches as learners in the DML framework and the DML package, respectively, is described and validated.

In the following sections of this thesis, a simulation study is conducted in order to validate the prior implementation and to examine the behaviour of the implemented DL methods. The simulation study is planned and designed in a structured approach that details the various steps of the study design. Thereafter, the execution of the simulation study is presented in a two-stage process and the results are analysed and discussed.

Afterwards, potential use cases for CML with DL approaches are provided based on the results of the simulation study and on general theoretical foundations. According to these explanations, a case study with observational data is conducted in order to show the potential benefit of using DL approaches for CML in a practical example. Finally, all results and findings of this thesis are summarized and an outlook on directions for future research is given.

2 Theoretical Foundations

The following sections provide a brief introduction to causal inference, CML, the DML framework as well as DL. Additionally, text and images as a special datatype and their use in causal inference are introduced. It is not intended to provide a detailed background for all of the topics that are mentioned, for this, it is kindly referred to the relevant works that are cited. Nevertheless, these theoretical basics shall provide a common understanding of the topics covered in this thesis.

2.1 Causal Inference

Unlike typical prediction or description tasks, causal inference is about the understanding of how one variable causally affects another variable [40], [51], [93]. In other words, causal inference refers to the process of determining the causal relationships between variables by analyzing empirical data through statistical methods [68]. In practice, causality and the understanding of cause and effect are important for decision-making in various disciplines like medicine, social sciences, and economics as simple predictions are often insufficient for choosing impactful interventions [101].

Figure 1 shows a typical causal graph highlighting the three important elements of a causal relationship. Classical predictions uncover the correlation pattern between a treatment and an outcome. Actually, this pattern combines multiple causal pathways, as the treatment can have a direct effect on the outcome. Additionally, some confounding factors can influence the probability of the treatment and the outcome (see Fig. 1) [71]. The causal treatment effect is the sole effect of the treatment on the outcome.

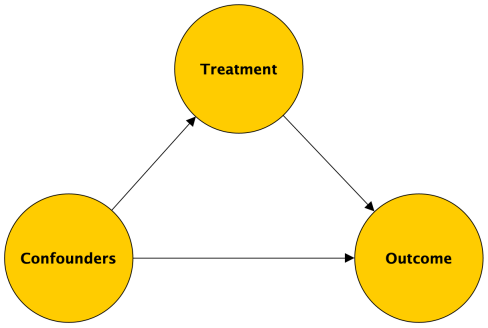


Figure 1: Causal Graph

Source: own figure.

As a basis for further explanation, it is worthwhile to first take a closer look at a framework for causal inference. Figure 2 shows a typical high-level causal inference framework.

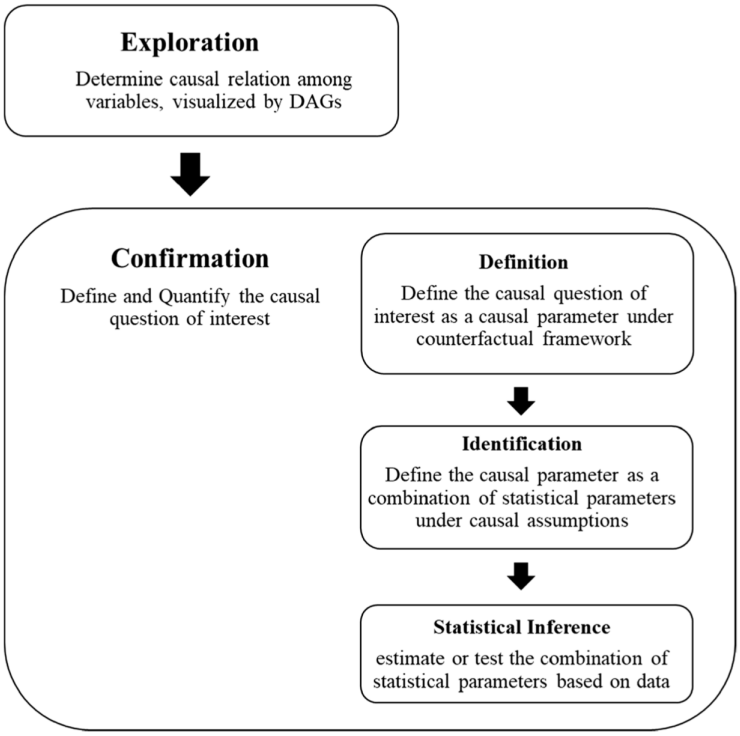


Figure 2: Causal Inference Framework

Source: [64, p. 184]

At first, there is the stage of exploration where the structure of the causal relationship is built up and, for example, visualized by a Direct Acyclic Graphs (DAG) [64]. A DAG thereby is a graphical representation which visualizes the key concepts of treatment, outcome, causation, confounding, and bias just like in the simple example in Figure 1 [119]. The nodes represent the individual variables and the edges represent the relationship [12]. A DAG has no cycles and is directed in the sense that each edge is oriented from one specific node to another [12].

Secondly, the stage of confirmation is about the definition and quantification of the considered causal question. This stage consists of three steps (see Fig. 2). In the first step, the causal question is defined in a formal way as a causal parameter under the counterfactual outcome model [64]. Those parameters underlying the counterfactual model cannot be measured directly by observational data. Therefore, it is required to identify those parameters as statistical parameters in combination with some assumptions in the second step [64]. In the last third step, the causal question is estimated or tested through statistical inference based on available data [64].

As mentioned in the first step of the confirmation phase in Figure 2, there is a counterfactual outcome model that is used to formally define causal questions. In the case of a binary cause, the counterfactual framework assumes the existence of two well-defined causal states to which all units of the population can be assigned [79].

Usually, these two states are labelled as treatment and control [79]. Under the condition of well-defined causal states, causal inference in the sense of the counterfactual model assumes the existence of potential outcome random variables defined for all units in the considered population [79]. For a binary cause, the two potential outcome variables associated with the causal states are denoted as $Y(1)$ and $Y(0)$. As per convention, a policy or causal exposure variable D is defined which can take two values. It is defined that D equals 1 for all units of the population that have been exposed to the treatment. D equals 0 for all those units of the population that have been exposed to the control state. By convention, the units exposed to the treatment condition are assigned to the treatment group, while the units exposed to the control condition are assigned to the control group [79]. As the variable D is defined on population level, both the treatment and the control group exist in the population and in some observed data [79]. It is worth mentioning that exposure to the treatment or control state can be either by an individual's decision or by a planned or random assignment [79]. To summarize, the observable outcome variables can be defined as:

$$Y = Y(1) \qquad \text{if } D = 1 \tag{1}$$

$$Y = Y(0) \qquad \text{if } D = 0 \tag{2}$$

As already stated in section 1.1, one fundamental challenge of causal inference is that counterfactual outcomes can not be fully observed for each unit [81]. This problem is outlined in Table 1. In other words, the potential outcome for the treatment state can never be observed for those units who are assigned to the control group due to their value in the policy variable D and vice versa. This makes it typically impossible to directly calculate causal effects on unit-level [79].

Table 1: Fundamental Problem of Causal Inference
Source: own table, in the style of Morgan et al. [79]

	$Y(0)$	$Y(1)$
Treatment group ($D = 1$)	Counterfactual	Observable as Y
Control group ($D = 0$)	Observable as Y	Counterfactual

Because of the impossibility to calculate unit-level causal effects, it is usual to focus on the calculation of aggregated effects [79]. Typically it is referred to the Average Treatment Effect (ATE) in the population [79]:

$$ATE = \mathbb{E}[Y(1)] - \mathbb{E}[Y(0)] \tag{3}$$

In some cases, more specific ATEs are of greater interest. Two conditional ATEs which are often referred to are the Average Treatment Effect for the Treated (ATT) and the Average Treatment Effect for the Controls (ATC) [79]:

$$ATT = \mathbb{E}[Y(1)|D = 1] - \mathbb{E}[Y(0)|D = 1] \tag{4}$$

$$ATC = \mathbb{E}[Y(1)|D = 0] - \mathbb{E}[Y(0)|D = 0] \tag{5}$$

The counterfactual model can retain its transparency if a basic assumption is maintained which is called Stable Unit Treatment Value Assumption (SUTVA). This assumption requires that the potential outcome of units does not change when the treatment exposure of other units is potentially changed [79]. In other words, the effect of the treatment on a unit's outcome is independent of whether another unit has received the treatment or not.

To analyse the effect of a treatment one would preferably use a randomized experiment where the assignment of the treatment can be controlled [79]. The randomization eliminates any relationship between potential confounding factors and the assignment of the treatment (see Fig. 1) [71]. In such research design, the policy variable D and therefore the treatment status is independent of the potential outcomes $Y(0)$ and $Y(1)$ [79]. Therefore, any difference between the treated individuals and the un-treated individuals can be directly interpreted as the causal treatment effect [71]. No confounding variables need to be measured in this case. This treatment assignment mechanism is then also called ignorable under the independence assumption [79]:

$$(Y(0), Y(1)) \perp\!\!\!\perp D \tag{6}$$

In some weaker situations, ignorability also holds when a set of observed confounding variables X is introduced which completely characterizes the assignment of the treatment. It is assumed that after conditioning on those confounders, any other measured covariates are irrelevant [101]. The treatment assignment is ignorable when the potential outcomes are independent of the treatment status conditional on those variables [79]:

$$(Y(0), Y(1)) \perp\!\!\!\perp D|X \tag{7}$$

The ignorability assumption is also known as exogeneity, selection-on-observables or unconfoundedness assumption [58]. If D is conditionally exogenous it is as good as randomly assigned conditional on the confounding covariates [17]. Conditioning on relevant confounders X breaks the dependency between those confounders and the treatment (see Fig. 1).

More common than using randomized experiments are analyses based on observational data. Here, the researcher cannot control the treatment assignment and therefore the independence assumption as presented in equation 6 cannot be maintained [79]. It is up to the researcher to investigate how the treatment has been assigned or more generally how the different units ended up in their causal state [79]. In practice, fully examining the process of selecting alternative treatments in an empirical way is often infeasible and challenging [79]. This is because there might be underlying expectations of individuals that cannot be measured empirically [79]. In such cases, the analysis must be statistically adjusted using methods like matching or regression adjustments to remove the confounding bias that results from violating the assumption shown in equation 6 [51]. For this, the confounding influence needs to be properly measured via variables that need to be specified, observed, and conditioned on in order to comply with the unconfoundedness assumption as stated above.

For further explanations, it is also referred to sample-based quantities besides the already introduced population-based quantities. Therefore, the sample size is defined by N and each unit in the observed treatment and control group is referred to with lowercase letters. Therefore, the policy variable D exemplary take on values $d_i = 1$ for all i units observed to be in the treatment group.

The ATE can be naively estimated by subtracting the sample mean of the observed outcome for the observed treatment group (n_1 as the number of units that have received the treatment) from the observed outcome of the control group (n_2 as the number of units that have not received the treatment):

$$ATE_{naive} = \frac{1}{n_1}\sum_{i}^{n_1}(y_i|d_i = 1) - \frac{1}{n_2}\sum_{i}^{n_2}(y_i|d_i = 0) \tag{8}$$

This naive estimator rarely leads to a consistent estimate of the ATE because it converges to $\mathbb{E}[Y(1)|D = 1] - \mathbb{E}[Y(0)|D = 0]$ [79]. This can be shown by decomposing the ATE in a function of five unknowns. For more details on the decomposition, it is referred to Morgan et al. [79]. Essentially, there are two counterfactual conditional expectations that cannot be estimated without stating any assumptions because of the fundamental problem outline in Table 1 [79]: $\mathbb{E}[Y(0)|D = 0]$ and $\mathbb{E}[Y(0)|D = 0]$

One cannot provide an estimate based on an observed outcome that is converging to the true values for either of the above's conditional expectations [79]. By decomposing, the expected bias in the naive estimator of the ATE can be analyzed. Actually, there are two potential sources of bias in the naive estimator. One of those sources is a baseline bias which equals the difference in the average outcome between the treatment and control group in absence of the treatment [79]. A practical example of the appearance of this baseline bias can be that people who attend universities might have already been smarter in the first place so the causal effect of attending a university on smartness is overestimated. The second potential source of bias is the differential treatment effect bias [79]. This bias equals the difference in the ATE between those in the treatment and in the control group multiplied with a defined proportion of the population [79]. The differential treatment effect bias means that the treatment would have a different effect among the treatment and control groups. For example, the mental performance of those who attend a university may increase more than those who do not attend a university if they had attended a university instead. For more details, it is once again referred to Morgan et al. [79]. To enable unbiased and consistent estimates of the ATE with the naive estimation approach, additional assumptions are needed. Referring back to Figure 2, these assumptions are required in the second step of the confirmation stage. One assumption might be that the potential outcomes are independent because of a random assignment of the treatment, as stated above [79]. But as also already mentioned, for observational studies a random assignment of the treatment is infeasible and thus the assumption might not be justified.

As stated, the naive estimator of the ATE might be biased and inconsistent unless some assumptions can be maintained. Thus, there are strategies and methods

necessary to overcome this issue. One of those strategies can be to condition on one or more confounding variables X in order to partition the sample into subgroups in which certain assumptions can be maintained [79]. Hereby, two major conditioning strategies are matching and regression analysis [79]. In some more detail, the regression analysis attempts to account for all other relevant direct causes of the outcome [79]. The conditioning strategy is to eliminate all the associations between the causal variable and the outcome variable that are produced by the confounding variables X [79]. For this, a supervised model is fitted from the observed data for the expected conditional outcome [51]:

$$y(d, x) = \mathbb{E}[Y|D = d, X = x] \tag{9}$$

The learned conditional outcome $\hat{y}$ is then used to predict the counterfactual outcome for each observation given the treatment and the confounding [51]:

$$ATE_{regression} = \frac{1}{n}\sum_{i}^{n}(\hat{y}_i(1, x_i) - \hat{y}_i(0, x_i)) \tag{10}$$

In summary, it can be shown that observational data often contains confounding variables which can influence both, the outcome and the treatment (see Fig. 1) [51]. Neglecting such confounders violates the unconfoundedness assumption in equations 6 and 7 and can bias the estimation of the causal effect.

Other conditioning strategies will not be discussed further at this point and it is referenced to pertinent literature, like Morgan et al. [79]. One concrete approach to de-bias causal estimates using covariates will be presented in section 2.3 by introducing the DML framework.

2.2 Causal Machine Learning

As stated at the beginning of this thesis, CML describes the application of ML methods to causal inference problems. Historically, ML methods were mostly used for predictive tasks, for example, to identify patterns of variables in some data which then could be used to predict a particular outcome [60]. Therefore, ML methods have historically been tailored for prediction. In this field of research ML has outperformed conventional predictive models [64]. This performance led to the expectation that ML can similarly impact explanatory research which focuses on identifying and understanding causes [64]. However, the application of ML in the context of causal inference is not free of challenges [64]. Actually, conventional ML methods that are built on top of pattern recognition and correlation analysis might fail in causal analyses [101]. In recent years, efforts are ongoing to develop suitable theoretical frameworks to use ML methods in causal analyses [64]. These efforts led to the steadily growing literature field of CML.

Referring back to the causal inference framework in Figure 2, the question is where and at which stage to apply ML. It can be argued that already the structure of causal relationships in the exploration stage can be determined in a data-driven way, but a general consensus

is that this structure must be defined on the basis of essential a priori available information [64]. Also, the definition and identification of the causal question in the second stage in Figure 2 require substantive knowledge about the topic of research [64]. Based on this explanation, applying ML is considered to be most suitable for the last step of statistical inference in Figure 2. It can then be argued that ML per se is insufficient to infer causality but it can still be useful to serve causal inference by being a part of statistical inference [64].

One example of applying ML in the above-mentioned way are the estimation of propensity scores. Propensity scores are an individual's tendency or likelihood to be treated that can only be estimated and not observed [101]. Essentially, propensity scores are used to control for confounding in case of high-dimensional covariates [64] as individuals with similar covariates get similar scores [101]. Propensity score models are regressed on all potential covariates and included in the final causal model [64]. Referring back to the causal inference framework in Figure 2, constructing the propensity score model and the resulting causal model can be assigned to the third step of statistical inference. In the case of high-dimensional covariates, regular statistical methods are likely to fail due to model misspecification and estimation problems [64]. Therefore, ML can be a suitable and beneficial approach to construct a propensity score model [64].

To sum up, ML might not be suitable for explanatory research as this part of research requires prior knowledge of the scientific field of research or insights on the methodological issues in the study design [64]. However, ML might bring a benefit to other parts of causal inference, especially to those related to model specification and statistical inference [64].

As stated in section 2.1 there might be some assumptions or strategies necessary in order to come up with unbiased and consistent estimates of the ATE, especially in the common case where observational data is used. If for example all relevant variables that jointly affect the potential outcome as well as the treatment assignment are observed and therefore can be controlled for (ignorability or unconfoundedness assumption), ML methods enable the evaluation of causal effects in observational studies [60]. ML methods can be used to model non-linear effects of confounders in a better way or to find low-dimensional representations of the confounding factors [101]. One advantageous benefit compared to regular or traditional methods for causal inference is that the ML methods can exploit a lot of information in a data-driven way referring back to the catchphrase of Big Data [60]. CML can be used to address research questions based on high-dimensional data which contains many confounding variables that are observed and therefore can be used as control variables [60]. To summarize once again, CML is the combination of ML methods with causal inference approaches to, for example, select potential control variables in a data-driven way in order to estimate causal effects.

One major advantage of using CML in comparison to conventional causal inference methods is the above-mentioned selection of covariates. In conventional methods, these covariates must be selected manually [60]. This is difficult to infeasible for large datasets in the context of Big Data due to a large set of potential covariates and due to unstructured data like text [60]. Furthermore, there is also a risk of overlooking important

confounding variables [60]. An additional challenge in conventional methods is that the functional relationship between the covariates and the treatment assignment as well as the potential outcome has to be specified manually a priori [60]. CML has the benefit of taking the full amount of data to select the relevant covariates and control for them in a data-driven way as well as to estimate the functional relationship of the associations flexibly [60].

2.3 DoubleML Framework

The DML framework by Chernozhukov et al. [17] provides one CML approach to use ML methods in the context of causal inference and to appropriately estimate a causal effect. In more detail, the DML framework uses in a semi-parametric environment various ML methods to estimate high-dimensional nuisance parameters η_0 to infer on a low-dimensional parameter of interest θ_0 in presence of η_0 [17]. Hereby, the parameter of interest θ_0 is the causal effect or treatment effect parameter [17]. The DML framework provides a suitable approach for scenarios where η_0 is so high-dimensional that traditional assumptions to limit the complexity of the parameter space fail [17]. Actually, the ML methods to estimate the nuisance parameters η_0 are able to handle a large number of covariates and can obtain natural estimators even for highly complex parameters [17]. This deviates from the traditional semi-parametric framework where the complexity of the nuisance parameter space is sufficiently small [17]. To summarize, the DML framework contributes a general framework to estimate and to perform inference on the parameter of interest θ_0 in a formally valid way in the presence of a high-dimensional nuisance parameter η_0 [17].

The approach of using ML methods, which are well suited for high-dimensional settings due to the employment of regularization, to estimate η_0 brings certain challenges that are addressed in the DML framework [17]. When the estimators of η_0 generated by an ML method are used to estimate θ_0, the regularization bias and overfitting in the estimation of η_0 can cause a heavy bias in the estimators of θ_0 [17].

In the context of ML, the term regularization describes a method or technique to improve the generalization ability of a model [107]. Thereby, generalization ability means that the model performs well not only on the finite training dataset but also on a new, unseen test dataset. Regularization can, for example, impose a penalty term on the model's complexity to allow for a good generalization ability [107]. The fundamental issue of overfitting prevents perfectly generalized models [123]. In other words, overfitting describes that a model performs well on the training set while having poor performance on the test set. Thus, the model is not generalizing well from known training data to unseen test data. In supervised ML overfitting happens due to some noise present in the training data, the limited size of the training set, and the complexity of classifiers [123].

As mentioned above, there is a heavy bias in the estimator of θ_0 that is caused if one naively plugs in the estimators for η_0, that are obtained by ML methods, in the estimation equations for θ_0 [17]. The naive estimator for θ_0 fails to be root-N ($N^{-1/2}$) consistent due to this bias [17]. Per definition, an estimator is called consistent when it

has the property to converge in probability to the true value as the sample increases [22]. Root-N consistency describes how quickly the estimator converges to the true value. Obtaining root-N estimates for low-dimensional objects in presence of nuisance parameters is one known problem in classical semi-parametric estimation [17]. For further details, it is referred to Robinson et al. [89].

As the generic theory behind the DML framework is elementary and relies on only weak requirements, this framework allows the use of a broad range of different ML methods [17]. Chernozhukov et al. [17] have already identified deep neural networks as a suitable approach, among others, for estimating the nuisance parameters η_0.

The paper from Chernozhukov et al. [17] in which the DML framework is introduced has over 1,400 citations in articles available on Google Scholar as of this writing (25.11.2022). This relatively high amount of citations shows how well received the DML framework is in scientific research but also in practical applications. There are several examples where the DML framework is used in empirical studies. As already presented in section 1.2, Manzoor et al. [69] use neural models of language in the DML framework to control for unstructured text and therefore, to the best of the author's knowledge, are the only ones who use DL approaches in the context of DML. Other than that, several papers rely on the DML framework while using more traditional ML methods. For example, Gordon et al. [33] compare the performance of DML with the method of propensity score matching to estimate the causal effect of conversion ads on Facebook. The authors find out that DML outperforms propensity score matching but that also both approaches significantly overestimate the effect due to missing controlling factors [33]. These two aforementioned works are only intended as examples to outline the various studies that are built on top of the DML framework.

In the following section, the basic principles and theories behind the DML framework shall be briefly presented and explained in order to provide a basic understanding for the upcoming sections. For more details, especially about the underlying statistical principles when constructing the functions of the framework, it is referred to Chernozhukov et al. [17].

2.3.1 Problem Statement

To show the challenges that arise when naively using ML methods in the above-mentioned way for causal inference, it is referred to the lead example as presented by Chernozhukov et al. [17]. The lead example shall be a Partially Linear Regression (PLR) model as described by Robinson [89]:

$$Y = D\theta_0 + g_0(X) + U, \qquad \mathbb{E}[U|X, D] = 0 \tag{11}$$

$$D = m_0(X) + V, \qquad \mathbb{E}[V|X] = 0 \tag{12}$$

In the above equations, Y is the outcome variable of interest, D is the policy variable of interest, and the vector $X = (X_1, ..., X_{dim_x})$ describes other controls or confounding factors. U and V are disturbances. In the style of Chernozhukov et al. [17], D shall be a

scalar for simplicity reasons. Figure 3 shows the causal diagram of PLR models which is similar to the basic DAG shown in Figure 1.

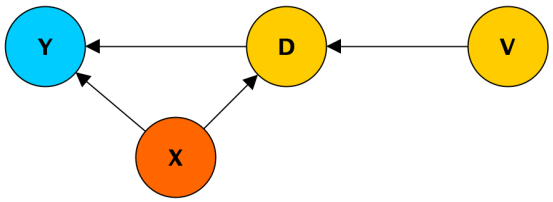


Figure 3: Causal Diagram of Partial Linear Regression Models
Source: own figure.

Equation 11 is the main equation and θ_0 is the parameter of interest that one would like to infer. This parameter can be interpreted as the treatment effect parameter of D on potential outcomes if D is exogenous conditional on the covariates X (see also equation 7) [17]. Equation 12 describes the confounding or how the policy variable D depends on the controls X, respectively. The policy variable D is affected by the confounding factors X through the nuisance function $m_0(X)$ whilst the outcome variable Y is affected by X via the nuisance function $g_0(X)$. When the dimension of X dim_x is large relative to the sample size N, traditional assumptions limiting the complexity of the parameter space for the nuisance parameters $\eta_0 = (m_0, g_0)$ fail [17].

For the following subsection, it is referred to the lead example of a PLR model as presented in equations 11 and 12.

2.3.2 Regularization Bias through a naive Approach

A naive approach for inference would be the direct application of a ML method. This would be the construction of a ML estimator to learn the regression function in equation 11. For simplicity and in the style of Chernozhukov et al. [17] the sample N is assumed to be split into two equal parts, one main and one auxiliary part. In the following, the observations in the main sample part with size n are indexed by $i \in I$ and in the auxiliary part by $i \in I^c$. If the nuisance function g_0 is now estimated as $\hat{g}_0$ with ML methods using the auxiliary part of the sample, the final estimate of θ_0 can be obtained with the main part of the sample [17]:

$$\check{\theta}_0 = \left(\frac{1}{n} \sum_{i \in I} D_i^2 \right)^{-1} \frac{1}{n} \sum_{i \in I} D_i (Y_i - \hat{g}_0(X_i)) \tag{13}$$

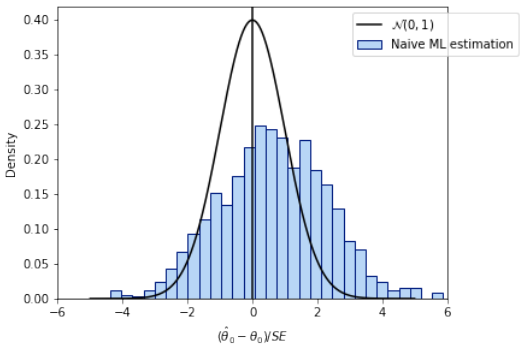


Figure 4: Naive Machine Learning Estimation

Source: own figure.

The results of the naive approach for estimating θ_0 in a PLR model are shown in Figure 4. These results arise from a minimal code example in the style of Bach et al. [7] where data is simulated with $n = 500$ and $dim_x = 20$ and Random Forest is used as a ML method to learn g_0 and m_0. The results of 1000 independent repetitions of data generation are studentized and the distribution of the studentized estimator (blue histogram) is displayed in comparison to the standard normal distribution $\mathcal{N}(0, 1)$ (black curve). The code for this minimal example can be found in Appendix C.1. In Figure 4 it can be observed that the estimator has some significant bias because the distribution of the studentized estimator is shifted to the right side and differs essentially from a normal distribution. This bias is the aforementioned regularization bias, which is induced by the Random Forest learner in estimating g_0 and m_0. Also, the estimator $\check{\theta}_0$ will have a slow convergence rate in general [17]:

$$|\sqrt{n}(\check{\theta}_0 - \theta_0)| \xrightarrow{dim_x} \infty \tag{14}$$

This convergence rate is slower than $1/\sqrt{N}$ [17] and thus $\check{\theta}_0$ fails to be root-N consistent. The main factor for this behaviour is the bias in learning g_0 [17] that is visible in Figure 4. The impact of the bias can be shown by decomposing the scaled estimation error in estimating $\check{\theta}_0$ [17]:

$$\begin{aligned}\sqrt{n}(\check{\theta}_0 - \theta_0) = &\underbrace{\left(\frac{1}{n}\sum_{i\in I} D_i^2\right)^{-1} \frac{1}{\sqrt{n}}\sum_{i\in I} D_i U_i}_{=:a} + \\ &+ \underbrace{\left(\frac{1}{n}\sum_{i\in I} D_i^2\right)^{-1} \frac{1}{\sqrt{n}}\sum_{i\in I} D_i(g_0(X_i) - \hat{g}_0(X_i))}_{=:b}\end{aligned} \tag{15}$$

Part a of equation 15 is approximately normally distributed and centred around zero under mild conditions [17] and part b is the regularization bias. The regularization bias is not centred and generally diverges [17]. Instead, the regularization bias can be described through a sum of terms that represents the nuisance functions m_0 and g_0 as well as the

estimator $\hat{g}_0$ [17]. These terms have non-zero means as they are obtained with regularized estimators in a high-dimensional setting [17]. As mentioned in section 2.3, regularization is used to improve generalization ability, which in this case is described primarily by not exploding the variance in the estimators [17]. However, the regularization entails a heavy bias in the estimator $\hat{g}_0$ of g_0.

2.3.3 Overcoming the Regularization Bias by using Orthogonalization

As seen in subsection 2.3.2, there is a regularization basis if one naively applies a ML method directly. To tackle this bias, Chernozhukov et al. [17] suggest to use a second construction with an orthogonalized formulation. To obtain the orthogonalized regressor $V = D - m_0(X)$ (rearranging equation 12), the effect of X from D is partialled out. In order to get the estimator $\hat{V}$, $\hat{m}_0$ is used as an ML estimator of m_0 based on the auxiliary part of the sample [17]. This key step also defines the name for the DML framework, as an auxiliary prediction problem is solved with a ML method resulting in the application of "double ML" [17]. The debiased estimator for θ_0, which is called DML estimator $\hat{\theta}_0$ further on, using the main part of the sample can be defined as [17]:

$$\hat{\theta}_0 = \left(\frac{1}{n} \sum_{i \in I} \hat{V}_i D_i \right)^{-1} \frac{1}{n} \sum_{i \in I} \hat{V}_i (Y_i - \hat{g}_0(X_i)). \tag{16}$$

Equation 16 shows that, just as in equation 13, the direct effects of confounding are removed by subtracting $\hat{g}_0$. Additionally, as D is approximately orthogonalized with respect to X, the DML estimator $\hat{\theta}_0$ has no such regularization bias as seen in Figure 2 [17]. Linking the DML estimator $\hat{\theta}_0$ to classical econometric literature, $\hat{\theta}_0$ can also be interpreted as a linear Instrumental Variable (IV) estimator [17].

The benefits of the estimation of θ_0 with $\hat{\theta}_0$ and thus of the auxiliary second prediction step are illustrated by Chernozhukov et al. [17]. To summarize, the scaled estimation error in estimating $\hat{\theta}_0$ can be decomposed into three terms:

$$\sqrt{n}(\hat{\theta}_0 - \theta_0) = a^* + b^* + c^* \tag{17}$$

Similar to part a of equation 15, part a^* in equation 17 is approximately normally distributed and centred around zero under mild conditions [17]. Part b^* in equation 17, which is capturing the regularization bias in estimating both, g_0 and m_0, can now vanish for a broad range of Data Generating Process (DGP)s [17]. To secure good properties of $\hat{\theta}_0$, also the third term c^* needs to be sufficiently well behaved [17]. This remainder term will be looked at in subsection 2.3.4.

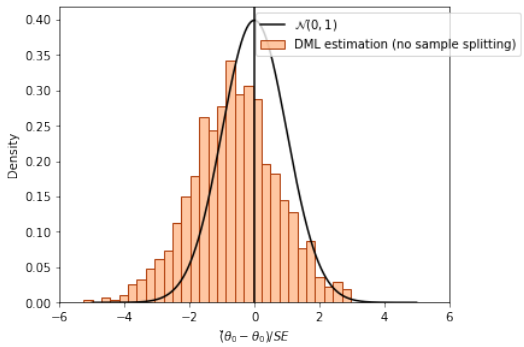


Figure 5: Double Machine Learning Estimation (no Sample Splitting)

Source: own figure.

The results of using an orthogonalized regressor as a second construction for estimating $\hat{\theta}_0$ in a PLR model are shown in Figure 5. These results arise from a minimal code example in the style of Bach et al. [7] and simulated data matching the data for Figure 4. Also, the same Random Forest learner as for Figure 4 is used as a ML method to learn both nuisance functions. Actually, it is important to note that at this point both nuisance functions, $\hat{m}_0$ and $\hat{g}_0$ are estimated on the whole dataset that is also used to obtain the estimates $\hat{\theta}_0$. This deviates from the above explanations where only the auxiliary part of the sample was used to estimate the nuisance functions. This adjustment shall highlight the importance of sample splitting, which is discussed in more detail in subsection 2.3.4. The resulting estimators $\hat{\theta}_0$ of 1000 independent data generations are studentized and the distribution of the studentized estimator (orange histogram) is displayed in comparison to the standard normal distribution $\mathcal{N}(0, 1)$ (black curve). The code for this minimal example can be found in Appendix C.1.

In Figure 5 it can be observed that the DML estimator still has some substantial bias because the distribution of the studentized estimator is now shifted to the left side. This bias is resulting from overfitting as the whole dataset was used to, on the one hand, estimate the nuisance functions and, on the other hand, to obtain the estimator of θ_0.

In addition to the DML estimator $\hat{\theta}_0$ as defined in equation 16, Chernozhukov et al. [17] introduce another debiased estimator which is based on the partialling-out approach of Robinson [89]:

$$\hat{\theta}_0 = \left(\frac{1}{n} \sum_{i \in I} \hat{V}_i \hat{V}_i \right)^{-1} \frac{1}{n} \sum_{i \in I} \hat{V}_i (Y_i - \hat{\ell}_0(X_i)), \qquad \ell_0(X_i) = E(Y|X) \tag{18}$$

This approach has the minor advantage over the DML estimator in equation 16 that the nuisance parameters can be directly estimated by a ML method since all these nuisance parameters are conditional mean functions [17]. For the DML estimator in equation 16, the estimator for g_0 has to be constructed first either based on the Robinson-style approach above or directly by the use of assumption-specific methods [17].

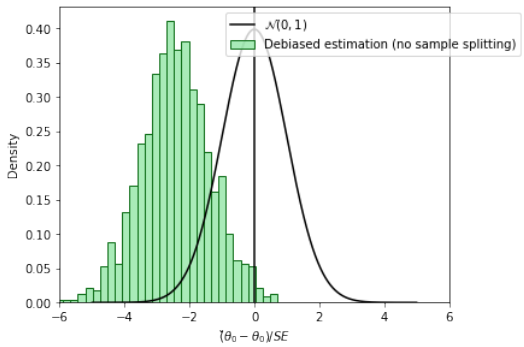


Figure 6: Debiased Estimation (no Sample Splitting) based on Partialling-out Approach

Source: own figure.

The results of using the debiased estimator based on the partialling-out approach of Robinson [89] as in a PLR model are shown in Figure 6. These results arise from a minimal code example in the style of Bach et al. [7] and simulated data matching the data for Figures 4 and 5. Also, the same Random Forest learner as for Figures 4 and 5 is used as a ML method to learn both nuisance functions. As for Figure 5, at this point, both nuisance functions, $\hat{m}_0$ and $\hat{g}_0$ are estimated on the whole dataset that is also used to obtain the estimates $\hat{\theta}_0$. The resulting estimators $\hat{\theta}_0$ of 1000 independent data generations are studentized and the distribution of the studentized estimator (green histogram) is displayed in comparison to the standard normal distribution $\mathcal{N}(0, 1)$ (black curve). The code for this minimal example can be found in Appendix C.1. One can see a substantial bias in Figure 6 which is caused by overfitting, just as in Figure 5.

2.3.4 Overcoming the Bias induced by Overfitting by using Sample Splitting

As seen in Figures 5 and 6, there is still a bias present when estimating $\hat{\theta}_0$ despite using orthogonalization as described in subsection 2.3.3. This bias is occurring when the same parts of the data are used to estimate the nuisance functions and to estimate $\hat{\theta}_0$ due to overfitting. This is why sample splitting has already been introduced in the aforementioned explanations. The auxiliary part of the sample was used to estimate the nuisance functions, while the main sample was used to obtain the final estimates of θ_0. Through sample splitting, one can argue that the third term in the decomposition (part c^* in equation 17) vanishes in probability [17].

But, however, there is a certain disadvantage in directly applying sample splitting. The estimator for θ_0 only exploits the data in the main part of the sample and thus only makes use of a subset of the available data. This can result in a significant loss of efficiency [17]. One solution to this issue is cross-fitting which describes that both, the main and the auxiliary parts of the sample, are used to obtain an estimator of interest. In this simple case, where the sample is divided into only two parts, two approximately independent estimators can be obtained [17]. These two resulting estimators can then be averaged to retrieve full efficiency. In other words, in cross-fitting, the role of both parts of the sample is swapped to obtain several independent estimators that then can be averaged [17].

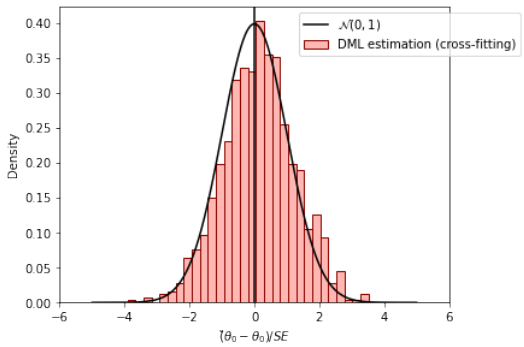


Figure 7: Double Machine Learning Estimation (Cross-fitting)

Source: own figure.

The results of using an orthogonalized regressor as described in equation 16 to estimate $\hat{\theta}_0$ in a PLR model in combination with a twofold cross-fitting procedure are shown in Figure 7. These results arise from a minimal code example in the style of Bach et al. [7] and simulated data matching the data for Figures 4 - 6. Also, the same Random Forest learner as for Figures 4 - 6 is used as a ML method to learn both nuisance functions. The difference to Figure 5 is that sample splitting is applied. One part of the sample is used to estimate the nuisance functions $\hat{g}_0$ and $\hat{m}_0$ and the other part of the sample is used to obtain the estimates for $\hat{\theta}_0$. Additionally, cross-fitting is used to exploit the entire sample for estimation. The resulting estimators $\hat{\theta}_0$ of 1000 independent data generations are studentized and the distribution of the studentized estimator (red histogram) is displayed in comparison to the standard normal distribution $\mathcal{N}(0, 1)$ (black curve). The code for this minimal example can be found in Appendix C.1.

In Figure 7 it can be observed that the DML estimator has no significant bias anymore. To summarize, the regularization bias could be removed by using orthogonalization in terms of partialling out the effect of X from D to obtain an orthogonalized regressor that is used to get $\hat{\theta}_0$ as the final DML estimate of θ_0. Furthermore, the bias induced by overfitting could be removed through sample splitting. Additionally, the full efficiency in the estimation could be regained through cross-fitting.

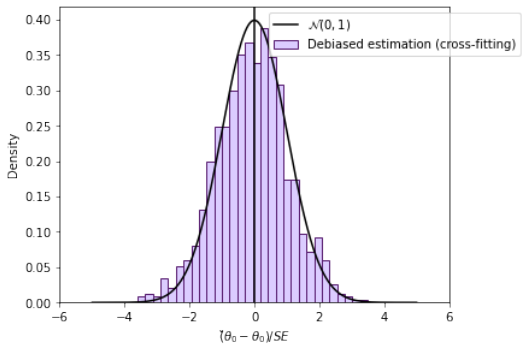


Figure 8: Debiased Estimation (Cross-fitting) based on Partialling-out Approach

Source: own Figure.

For the sake of completeness, Figure 8 shows the results of using the debiased estimator based on the partialling out approach by Robinson [89] as defined in equation 18 in a PLR model. These results arise from a minimal code example in the style of Bach et al. [7] and simulated data matching the data for Figures 4 - 7. Also, the same Random Forest learner as for Figures 4 - 7 is used as a ML method to learn both nuisance functions. In distinction to Figure 6 sample splitting and cross-fitting are applied. The resulting estimators $\hat{\theta}_0$ of 1000 independent data generations are studentized and the distribution of the studentized estimator (purple histogram) is displayed in comparison to the standard normal distribution $\mathcal{N}(0, 1)$ (black curve). Figure 8 shows that there is no bias. As for the DML estimator, sample splitting can be used to overcome the bias induced by overfitting also for the Robinson-style estimator.

To visualize the above progress in overcoming the bias, Figures 4 - 8 are combined in one graph in Appendix A.1.

2.3.5 Generalization

In order to provide more detail to follow the software implementation of the DML framework, also generalizations of the prior theories shall be presented.

The estimators of θ_0 as previously defined can be seen as the solution to estimate an equation involving a score function. Thereby, the naive estimator $\check{\theta}_0$ as described in equation 13 solves the following equation [17]:

$$\frac{1}{n} \sum_{i \in I} \varphi(W_i; \check{\theta}_0, \hat{g}_0) = 0 \tag{19}$$

In equation 19, φ describes a known score function, $\hat{g}_0$ an estimator for the nuisance function g_0, and W a random element that takes values in a measurable space and whose regularity is determined by a probability measure. For the considered lead example of a PLR model, the score function is defined as [17]:

$$\varphi(W; \theta, g) = (Y - \theta D - g(X))D \tag{20}$$

In equation 20 $W = (Y, D, X)$ is representing functions that map the support of X [17]. The score function in equation 20 is sensitive to any bias in estimating the nuisance function g [17]. As presented, there is a bias induced in the estimation of g when using ML methods due to the regularization.

In contrast, the DML estimator $\hat{\theta}_0$, as described in equation 16, solves the following equation [17]:

$$\frac{1}{n} \sum_{i \in I} \psi(W_i; \hat{\theta}_0, \hat{\eta}_0) = 0 \tag{21}$$

In equation 21, ψ describes an orthogonalized or debiased score function, $\hat{\eta}_0$ an estimator for the true nuisance parameters η_0, and W a random element that takes values in a

measurable space and whose regularity is determined by a probability measure.

Chernozhukov et al. [17] refer to the Gateaux derivative operator as being one term that needs to vanish in order to establish good behaviour of an estimator of θ_0. For the score function of the naive approach (see equation 20), the Gateaux derivative operator with respect to g does not vanish when evaluated with true parameter values [17]:

$$\partial_g \mathbb{E}[\varphi(W; \theta_0, g_0)][g - g_0] \neq 0 \tag{22}$$

In contrast, the orthogonalized score function ψ has the property that the Gateaux derivative operator vanishes with respect to η when evaluated with true parameter values [17]:

$$\partial_\eta \mathbb{E}[\psi(W; \theta_0, \eta_0)][\eta - \eta_0] = 0 \tag{23}$$

Chernozhukov et al. [17] define the property presented in equation 23 as Neyman orthogonality and ψ as a Neyman orthogonal score. For the considered lead example of a PLR model, the DML estimator $\hat{\theta}_0$ as defined in equation 16 uses following score function [17]:

$$\psi(W; \theta, \eta) = (Y - D\theta - g(x))(D - m(X)), \qquad \eta = (m, g) \tag{24}$$

The score function ψ in equation 24 is in comparison to φ (see equation 20) not sensitive to any bias in the estimation of η_0 in that sense that the Neyman orthogonality (see equation 23) holds [17]. For the sake of completeness, the partialling-out score function based on the approach of Robinson [89] for the debiased estimation of θ_0 is defined as follows [17]:

$$\psi(W; \theta, \eta) = (Y - l(X) - \theta(D - m(X))(D - m(X)), \qquad \eta = (l, m) \tag{25}$$

The use of an orthogonal Neyman score removes the regularization bias that results from replacing the true nuisance parameters η_0 with an ML estimator $\hat{\eta}_0$ [7]. This is especially in high-dimensional settings of great importance, as $\hat{\eta}_0$ must be heavily regularized in order to be a good estimator for η_0 and therefore a significant bias is induced [7]. To summarize, the property of Neyman orthogonality and sample splitting establish well-behaved estimators for θ_0 [17]. For general proofs and detailed explanations on the construction of Neyman orthogonal scores, it is referred to Chernozhukov et al. [17].

Generally speaking, the DML procedure as presented in subsections 2.3.3 and 2.3.4 can be defined by moment conditions. The following explanations shall only outline the definition of the cross-fitted DML and are therefore simplified wherever possible. For further details, it is referred to Chernozhukov et al. [17]. The data with sample size n is randomly partitioned in K folds with each fold being of the size n_k. The estimator $\hat{\theta}_{0,k}$ for each fold k can be constructed as the solution of the following equation of the empirical expectation [17]:

$$\mathbb{E}_{n_k,k}[\psi(W; \hat{\theta}_{0,k}, \hat{\eta}_{0,k})] = 0 \tag{26}$$

As described, $\mathbb{E}_{n_k,k}$ is the empirical expectation over the kth fold of the data, ψ is the Neyman orthogonal score and $\hat{\eta}_{0,k}$ is a ML estimator of η_0 for the kth fold of the data. The individual estimators $\hat{\theta}_{0,k}$ that solve equation 26, can finally be aggregated to obtain the final estimate of the causal parameter $\tilde{\theta}_0$:

$$\tilde{\theta}_0 = \frac{1}{K} \sum_{k=1}^{K} \hat{\theta}_{0,k} \tag{27}$$

Chernozhukov et al. [17] introduce the approach defined by equations 26 and 27 as DML1. A variation of this basic cross-fitting approach is presented as follows [17]:

$$\frac{1}{K} \sum_{k=1}^{K} \mathbb{E}_{n_k,k}[\psi(W; \hat{\theta}_{0,k}, \hat{\eta}_{0,k})] = 0 \tag{28}$$

This approach is introduced as DML2 and may behave better in small samples which is why DML2 is generally recommended [17]. Nevertheless, both DML1 and DML2 can be used, and in some cases, there is no difference between both approaches [17].

To highlight just one key property of the DML estimator, it can be shown that the orthogonal score-based estimator achieves a convergence rate of root-N and is also approximately normally distributed [17]. This desirable property leads back to the classical semiparametric estimation of low-dimensional objects in the presence of perturbation parameters. For further theorems and a detailed description of statistical properties it is referred to Chernozhukov et al. [17].

Furthermore, Chernozhukov et al. [17] make several structured assumptions on η_0 which allow the use of different ML methods. For example, it is stated that deep neural networks are attractive if η_0 can be well approximated by some sparse or deep neural networks [17].

2.3.6 Inference in different Model Settings

Until this point, the explanations followed and referred to the lead example of a PLR model as described in subsection 2.3.1. Moreover, the DML framework provides the theoretical background to estimate θ_0 and to perform inference on it for other model settings. The context of these other models in the DML framework shall only be briefly presented, for more details, it is referred to Chernozhukov et al. [17].

First, the PLR model as described in equations 11 and 12 can be extended to allow for IV identification. In reference to the DML package these models are called Partially Linear IV Regression (PLIV) models [7] and include an IV Z. One can similar to the explanations for the PLR models define orthogonal score functions, either for the DML approach (as in equation 24) or for the partialling-out approach by Robinson (as in equation 25). As this extension of the PLR model is quite simple and straight-forward, it is renounced to present the respective equations.

Besides the partially linear models, PLR and PLIV, the DML framework provides also one solution to estimate treatment effects under the unconfoundedness assumption (see equation 7). As outlined in section 2.1 there are various methods for this setting and as described in section 2.2, the application of ML methods might be beneficial and advantageous. Chernozhukov et al. [17] focus on the setting with fully heterogeneous treatment effects and binary treatment variable $D \in \{0, 1\}$. The model is built up with the following equations where the notation of the probability distribution is neglected for simplicity reasons [17]:

$$Y = g_0(D, X) + U, \qquad \mathbb{E}[U|X, D] = 0 \tag{29}$$

$$D = m_0(X) + V, \qquad \mathbb{E}[V|X] = 0 \tag{30}$$

In reference to the DML package the model defined by equations 29 and 30 is called Interactive Regression Model (IRM) model [7] which is more general than the PLR model in settings where D is binary [17]. The treatment variable D is affected by the confounding variables X through the propensity score $m_0(X)$ [17]. The outcome variable Y is affected by X via the function $g_0(D, X)$ [17]. Those functions, g_0 and m_0, are unknown and can be learned by ML methods, which might be advantageous as presented in section 2.2. A target parameter of interest in this model is the ATE which is defined by the following equation [17]:

$$ATE = \theta_0 = \mathbb{E}[g_0(1, X) - g_0(0, X)] \tag{31}$$

Equation 31 is a concrete definition of the ATE introduced in equation 3 for the IRM model. As stated in section 2.2, it is also often referred to the conditional ATEs. Thus, the ATT as another target parameter can be defined as follows [17]:

$$ATT = \theta_0 = \mathbb{E}[g_0(1, X) - g_0(0, X)|D = 1] \tag{32}$$

Similar to the explanations in subsection 2.3.5 orthogonal scores ψ for the estimation of the ATE and the ATT can be constructed.

The IRM model can just like the PLR model be extended with a binary instrument Z to estimate a Local Average Treatment Effect (LATE). The resulting model is called, in reference to the DML package, Interactive IV Model (IIVM) model [7]. The LATE describes the average effect of the treatment for some compilers, which are observations that take on the value $D = 1$ if $Z = 1$ and $D = 0$ if $Z = 0$ [17]. Just as before, one can use an orthogonal score ψ to estimate the LATE.

2.3.7 Software Implementation

The software implementation of the DML framework within the DML package shall only briefly be presented here. For more details, it is referred to the user guide and the corresponding paper by Bach et al. [7]. As stated in section 1.2, the DML package by Bach et al. [7], [8] provides an open-source implementation of the DML framework for both, R and Python. The two packages for R and Python have a similar software architecture and Application Programming Interface (API) and mainly deviate in the package they are

built on [7]. Specifically, the DML package is built on top of scikit-learn [83] for Python and on top of mlr3 respective the mlr3 ecosystem [59] for R [7]. Here, Mlr3 provides an object-oriented ML framework in R and scikit-learn is an open-source ML library. The DML package is increasingly adopted in research and practice with being starred over 215 times on GitHub as of this writing (07.02.2023).

The explanations in this thesis shall follow the implemented Python package and its documentation in the corresponding paper by Bach et al. [7] and the corresponding website with user guide.

The theories outlined in subsections 2.3.3 and 2.3.4 have been implemented in the DML package for a range of causal models described in subsection 2.3.5. The DML package enables the use of ML methods for estimating nuisance functions and performing statistically valid causal inference on a parameter of interest. The implementation of the DML package is object-oriented and therefore the model specifications are flexible and extensions are easy to implement [7]. The class structure of the DML package is illustrated in Figure 9.

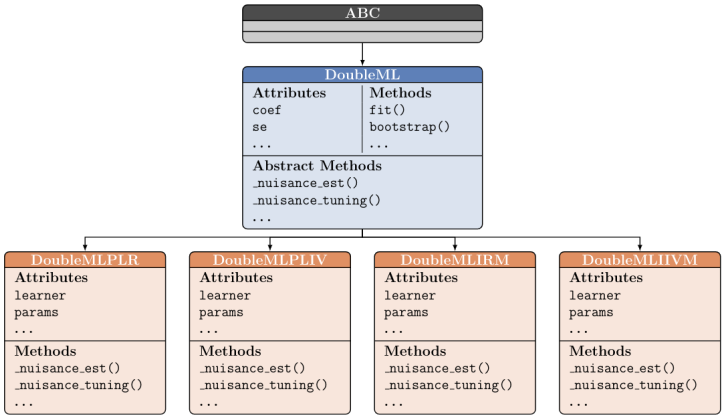


Figure 9: Class Structure of DoubleML Package

Source: [7, p.3].

The abstract base `class DoubleML` implements all key functionalities which are for example the estimation of the causal parameters, t-statistics, and more sophisticated methods for simultaneous inference [7]. The model classes for the different causal models PLR, PLIV, IRM, and IIVM are inherited from the central base class (see Fig. 9). In those model classes model-specific parts are added, for example, the estimation of the nuisance function through some ML method or methods to compute the Neyman orthogonal score function for the specific model [7].

A key property of the DML package is that the Neyman orthogonal score functions ψ are linear in the parameter θ [7]. The case of linear scores is defined by the

following equation in a simplified way [17]:

$$\psi(W;\theta,\eta) = \psi^a(W;\eta)\theta + \psi^b(W;\eta) \tag{33}$$

The assumptions for linear scores are presented by Chernozhukov et al. [17]. Exemplary, the score function for the estimation of the DML estimator $\hat{\theta}_0$ in a PLR model (see equation 24) can be decomposed if linear in the sense of equation 33:

$$\begin{aligned}\psi(W;\theta,\eta) &= (Y - D\theta - g(X))(D - m(X) \\ &= \underbrace{-D(D - m(X))\theta}_{\psi^a(W;\eta)} + \underbrace{(Y - g(X))(D - m(X))}_{\psi^b(W;\eta)}\end{aligned} \tag{34}$$

Through the linearity of the score functions it is possible to implement the key concepts of the DML framework in a general way [7].

In the DML package, the user can specify various options. These include the ML method for estimating the nuisance functions, the use of DML1 or DML2 as an algorithm, and different Neyman orthogonal score functions. For example, for a PLR model, the package offers the choice of using the partialling-out approach by Robinson [89] (see equation 25) or the so-called "IV-Type," which refers to the score function as defined in equation 24. Also, other functionalities are customizable and alternative, user-written score functions can be implemented via `callables` [7].

2.3.8 Other Causal Inference Frameworks and Implementations

Besides the DML framework and its software implementation, there are also other frameworks and software implementations that provide a solution for using ML methods in the context of causal inference. Since this work is primarily focused on extending the DML package through the application of DL approaches, this subsection offers only a brief overview of two alternative frameworks and software packages. These frameworks and packages could potentially be used to explore the use of DL approaches in a similar context.

One other solution to apply ML methods to estimate causal effects is provided by the Python package EconML, which is developed by Microsoft Research [74]. In more detail, EconML offers a solution to estimate heterogeneous treatment effects from different kinds of data sources [74]. EconML is built on top of standard Python packages and is highly adapted in research and practice with being starred over 2800 times on GitHub as of this writing (07.02.2023) [11]. The package EconML also implements parametric DML estimators as presented in subsection 2.3.3 in a designated class [70], [73]. Therefore, EconML partially refers to the DML framework using a two-stage process, where in the first stage nuisance functions are estimated using cross-fitting and in the second stage a certain causal model is estimated [70]. It is stated that neural networks are already implemented as a technique to model the effect heterogeneity [72]. After analyzing the code of the currently published version 0.14.0 it can be found that neural networks are only implemented for specific IV methods. This implementation is based

on the high-level DL API Keras which will be introduced in section 3. The comments in the respective code of EconML [11, *econml/iv/nnet/_deepiv.py*] indicate that the use of neural networks is under development as there are various To Do's stated referring to the reproducibility of the results among others.

Besides DML and EconML, CausalNex provides another Python library to combine ML with causal reasoning. Compared to the former two packages, CausalNex has a different approach as it focuses on Bayesian networks [12]. Bayesian networks describe probabilistic graphical models that include both the dependencies of variables and their distributions [12]. For this, Bayesian networks consist of a DAG and conditional probability distributions for each variable [12]. Actually, Bayesian networks are not a causal model by nature as the structure learning, which can be done by some ML method or by domain knowledge, only learns that there are certain dependencies between variables [12]. Nevertheless, Bayesian networks are said to be one of the most widely used structural causal models that combine graphical models and structural equations in a comprehensive framework including counterfactual logic [12]. The use of Bayesian networks has the disadvantage that causal inference cannot be fully automated [12]. CausalNex is developed by QuantumBlack and is highly adapted in research and practice with being starred over 1800 times on GitHub as of this writing (07.02.2023) [12]. To the best of the author's knowledge, there are no DL methods implemented up to the currently published version 0.11.1.

2.4 Deep Learning

After already referring to the terms "DL" and "DL approaches" several times, the following section shall outline important key points of this interesting and highly relevant field of ML. As DL is a large field of theoretical research and practical application, the following explanations shall only be brief and on a high level to serve the understanding of this thesis. For further details it is referred to pertinent literature for example Goodfellow et al. [32], Zhang et al. [125] or Schmidt-Hieber [97].

Generally speaking, ML is a subset of the broad field of artificial intelligence, where a system is learning in a data-driven way and improving from its own experience without any explicit programming [118]. In particular, DL is a subgroup of methods of ML where the algorithms are inspired by the functionality of the human brain and many layers of consecutive algorithms are used to create so-called deep neural networks [118]. This solves one of the central problems in representation learning, where the representation of an original problem needs a sophisticated and human-like understanding of the data due to the variety of factors of variation [32]. DL introduces representations that themselves can be expressed by simpler representation and therefore the algorithm can build up complex concepts out of simpler ones [32]. Deep neural networks are a modern implementation of DL [26] and the DL approach that is referred to in this thesis.

In general, neural networks describe one estimation method where the relationship between the inputs and the outputs is modelled with such above-mentioned layers

of connected computational units which are also called neurons [26]. In other words, neural networks have the goal of approximating an unknown function that maps an input to an output and learning the value of the parameters which results in the best function approximation [32]. As said, these layers are patterned in the style of biological neural networks of human brains [26]. As DL is a subgroup of methods of ML, an appropriate model and the parameters of the model are also learned in a data-driven way [26]. Classical shallow neural networks did not get much attention as they provided worse empirical performance together with challenging optimization in comparison to other ML methods like forest-based methods [26]. But due to more recent progress in the development of deep neural networks DL returned to prominence [26]. Deep neural networks consist of many hidden layers of neurons that sit between the inputs and the outputs [26]. The reason for this renewed attention to deep neural networks is the superior performance of these methods in different contexts and that deep neural networks set the state-of-the-art performance for many prediction tasks [26]. But also for complicated tasks like processing images for object detection or speech recognition, DL is considered to be state-of-the-art [97].

There are many different architectures for deep neural networks possible. Following the theory of deep neural networks, these architectures have different approximating power due to different complexities [26]. One easy and often described example is a feed forward neural network. Figure 10 shows such a feed forward neural network but it should be noted that the network in Figure 10 is not particularly deep.

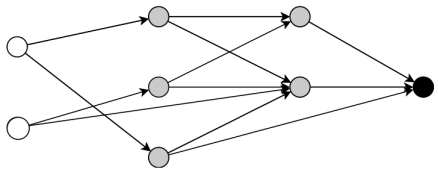


Figure 10: Feed Forward Neural Network
Source: [26, p.186].

In Figure 10 the inputs are shown in white on the left whilst the output is shown in black on the right. Between those, there are five hidden units, the neurons, in two layers. All nodes are fully connected between consecutive layers by a DAG. The information only flows in a feed forward way from the input through the intermediate computations in the hidden layers to the output and there is no feedback. To complete the construction of the network it is needed to specify an activation function that is applied to the output of each node. These activation functions are one fundamental ingredient to DL as they transform input signals to outputs and often add non-linearity [125]. Common activation functions are the sigmoid (equation 35) and the Rectified Linear Unit (ReLU) activation function (equation 36) [125]:

$$\sigma_{sig}(x) = \frac{1}{1+e^{-x}} \tag{35}$$

$$\sigma(x) = max(x, 0) \tag{36}$$

One specific architecture that is fully connected, has no other connections, and eventually has the same number of neurons per hidden layer is called Multilayer Perceptron (MLP) (see Fig. 11). This architecture is commonly used in empirical research [26]. It is known that MLPs are universal approximators for certain activation functions [125]. That means that any function can be modelled even by a neural network with a single hidden layer if one just adds enough nodes to this layer [125]. Until here, Figures 10 and 11 only present shallow networks, but incorporating many layers and thus using deep neural networks brings the benefit of approximating a variety of functions in a more compact way [125]. Other than that, a shallow network with only one layer might cause this layer to be infeasible large and may fail to learn and generalize in a correct way [32]. Describing MLP in other words, a MLP can be seen as a mathematical function to map the set of inputs on the outputs where the function consists out of many simpler functions [32]. Each function applied is providing a new representation of the input [32].

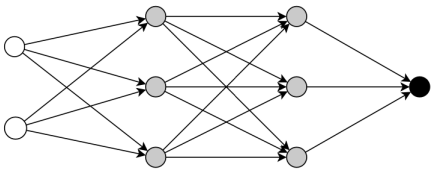


Figure 11: Multilayer Perceptron
Source: [26, p.186].

Farell et al. [26] state that the combination of deep MLP architecture and the ReLU activation function is one crucial point for neural networks to provide empirical state-of-the-art performance and to be feasible to optimize. In contrast, shallow networks with a sigmoid activation function might not perform well empirically and deep networks with a sigmoid activation function are not feasible to optimize [26]. Also, shallow networks with a ReLU activation function may not be able to approximate broad function classes [26]. It is again to be noted that the architecture of a neural network has a crucial impact on the approximation power of the network [26]. The architecture can be defined by the width, depth, activation function, and graph structure [26].

Neural networks are typically optimized layer-by-layer by some variant of Stochastic Gradient Descent (SGD) where the gradients of the network parameters, also called weights, are calculated via back-propagation [26]. This training process relies on the interdependence of forward- and back-propagations [125]. The forward-propagation computes sequentially from the input to the output layer and stores intermediate variables in a computational graph which is defined by the neural network [125]. The back-propagation calculates sequentially from the output to the input layer and stores the gradients of the intermediate variables within the network [125]. Both, forward- and back-propagation are alternated during the training process to update the model parameters after initialization [125]. There are various optimization algorithms available to train DL models but there are also certain challenges like local optima or vanishing gradients that need to be considered [125].

As for all ML methods, generalization issues and problems caused by overfitting arise also for DL approaches like deep neural networks. There are standard techniques available to tackle these challenges. One of such techniques is weight decay which is often integrated into the optimization algorithm for DL methods [125]. Weight decay is one particular choice of adding a penalty term to reduce the complexity of the learned model [125]. The penalty term penalizes the use of a larger weight vector which will be incorporated into the training process when updating the weights. One other technique to avoid overfitting in neural networks is dropout where noise is injected in the training process in order to drop out some of the neurons [125]. In other words, different nodes are randomly removed from the hidden layers at each training step. The intuition behind this regularization strategy is that the model should not only focus on certain specific neurons but instead should use all neurons as one important neuron might be cancelled out.

Often it can be observed that the training error decreases steadily over time when training a large model but the error in a validation set begins to increase [32]. Therefore, the parameter set of the model with the lowest validation error can be returned as this ideally also implies the lowest test error [32]. When the training process terminates, it is worth returning the parameter set that caused the lowest validation error instead of returning the latest parameter values [32]. It can be defined that the training process terminates if the validation error does not improve compared to the lowest recorded validation error for a specified number of iterations [32]. This simple regularization technique is called early stopping and is one of the most common approaches for DL models [32].

2.5 Text Data and their Use in Causal Inference

In addition to the previous theoretical foundations, this subsection provides a brief introduction to text as a dataform and its use in causal inference. Besides, various methods on how to deal with text are presented with a special focus on DL approaches. This section is already anticipating the setting of the case study in section 6.

New technologies have made text data available in vast quantities, representing human interaction, culture, and communication [31]. This text data becomes increasingly important and extends the structured data which is traditionally used in research [31]. In recent years, texts are increasingly used as data in empirical economic research, e.g. in the financial sector, where financial news or social media posts about a company are used to determine the development of its share price [31]. Text data differs fundamentally compared to the structured and numeric data that is traditionally used in economics. One important difference is that text is high-dimensional by nature [31]. Additionally, text as high-dimensional data is sparse as it is distributed over a huge amount of words [101]. Therefore, statistical methods to analyze text need to be capable of dealing with such high-dimensional data, which points in the direction of ML and DL techniques [31]. Further challenges arise from the fact that texts are a type of sequential data that require special treatment in statistical methods [125].

Typically the analysis using text can be outlined in a three-step approach as pre-

sented by Gentzkow et al. [31]:

1. Numerical representation of the raw text data
2. Map numerical text representations to the predictions
3. Use of predictions for descriptive or causal analysis

The first step is about the reduction of the dimensionality of the data in order to make the data manageable for statistical methods [31] and in general about preprocessing the data. In detail, the text first needs to be loaded as strings into the memory and then these strings are split up into tokens [125]. These tokens can either be words, phrases, or even characters [31]. Afterwards, a vocabulary is built to map each token into a numerical index so that the text can be converted to a sequence of numerical indices [125]. The relatively easiest representation of the tokens is called one-hot-encoding where each index is mapped to a different unit vector that can directly be used by a statistical method [125]. These vectors have a length equaling the number of different tokens in the vocabulary with all elements being 0 except for one element being set to 1 in order to represent the original token [125]. A reduction of the dimensionality during this representation can be exemplarily achieved by filtering out punctuation or stop words [31].

In the second step, a high-dimensional statistical method is applied to the pre-processed text data in order to obtain a certain predictor, e.g. to predict an indicator that states whether an email is spam or not [31]. This predictor is then used for further analysis (step three) [31]. In the current research, text analysis is applied in the context of prediction rather than causal inference [31]. But there are also several examples where text is used to infer the causal parameters of economics models [31]. Exemplary, Engelberg et al. [25] determine the coverage of earnings releases in local news which is then used to obtain the causal effect of the news on stock prices.

Embeddings

Having referred to one-hot-encoded vectors for representing words in a text, it is worth noting that this simple representation is not able to encode similarities between words [125]. The improvement of techniques to map words to vectors is one basic principle of NLP and is called word embedding [125]. There are several models to obtain mappings that take similarities and analogies among different words into account, like skip-gram and continuous bag of words [125]. Those models are used in tools like word2vec or Global Vectors (GloVe) [125]. Such word embedding models are typically pre-trained on large text datasets [125]. Pre-training includes training a model on unlabeled data for various different tasks [21]. The output of a word embedding model is a kind of matrix where each row is a vector representing a word of the vocabulary [125]. Both word2vec and GloVe are called context-independent because the same pre-trained vector is assigned to the same word regardless of context [125]. Obviously, due to the context-independent mapping where more complex semantic information is missing, there are certain limitations [125]. In contrast, other models have been developed that focus on context-sensitive word representations, which means that word representation

depends on context [125]. Furthermore, a distinction can be made between task-specific and task-agnostic representation models [125]. One example of a context-sensitive and task-specific model is Embeddings from Language Models (ELMo), which takes the entire sequence as an input [125]. The architecture of the model is task-specific and pre-trained representations are included as additional features [21]. In practice, task-specific models are nontrivial to build since the architecture of the model needs to be specified individually for every NLP task [125]. Task-agnostic models for context-sensitive representation are exemplary Generative Pre-Training (GPT) and Bidirectional Encoder Representations from Transformers (BERT) [125]. For those models, only minimal task-specific parameters are included and all pre-trained parameters are fine-tuned in order to achieve a final representation [21].

One major drawback of GPT compared to BERT is that the context is only encoded from left to right [125]. Therefore, each word responds only to the context on the left, which can result in the same words being represented in the same way, even though they have different meanings. BERT brings the advantage that context is encoded bidirectionally [125]. In addition, BERT does not require significant task-specific changes to the architecture to be suitable for a wide range of tasks [21]. BERT has further improved the state-of-the-art performance and revolutionized the solutions in various NLP tasks in several different categories [125].

The architecture of BERT can be described as a multi-layer bidirectional Transformer encoder [21]. Transformer is a relatively simple neural network architecture that has proved to perform better on machine translation tasks than sophisticated approaches that use complex Recurrent Neural Network (RNN) and attention mechanisms to connect an encoder and decoder [113]. Furthermore, Transformer proved to generalize well also to other tasks [113]. Prior to the introduction of Transformers, RNN and their improved versions have been the state-of-the-art approach in sequence modelling like various NLP tasks [113]. Attention mechanisms, which are often used in combination with RNNs, further improved the sequence modelling in various tasks [113]. In short, attention mechanisms include volitional cues, also referred to as queries and sensory inputs which are also named values in the context of attention mechanisms [125]. For example, these values can be described by intermediate feature representation [125]. Another ingredient of attention mechanisms are keys which can be thought of as nonvolitional cues of the sensory input [125]. In summary, attention mechanisms bias the selection over the values with attention pooling [125]. This attention pooling aggregates values in a selective way to produce the output while incorporating the queries and keys [125]. For further details on attention mechanisms, it is referred to Zhang et al. [125]. The Transformer model is only based on such attention mechanisms to obtain global dependencies between input and output [113]. For further details on the architecture of Transformers, it is referred to Vaswani et al. [113]. Nowadays, Transformer is applied in many modern DL applications like BERT [125].

In summary, BERT is a conceptually simple model for pretraining deep representations for NLP tasks that has demonstrated strong empirical performance and is widely used in research and practice [125]. Originally, BERT comes in two versions of different

model sizes, where the base model has 12 layers with 768 hidden units resulting in 110 million parameters and the large model has 24 layers with 1024 hidden units resulting in 340 million parameters [21]. After pre-training, BERT can be used to represent single text, pairs of text or even every single token in the text [125]. For further details on BERT and the pre-training process, it is referred to Devlin et al. [21]. All in all, it is visible that DL approaches have a huge impact on the representation learning for NLP tasks. The state-of-the-art models like BERT are developed on top of DL and neural network architectures.

Text in Causal Inference Settings

As described in section 2.1, it is common to use observational data in causal inference problems. Furthermore, it is stated that observational data often contains confounding variables that can influence both, the potential outcome and the causal effect. As unmeasured confounding variables can cause a bias when estimating the causal effect, it is of increasing interest to also include confounders from observed texts [51].

Especially for causal research on human behaviour, there are many potential confounding factors that can be obtained from unstructured text data [51]. In addition, there may be cases where the confounding factors cannot be measured directly and therefore text can be used as a surrogate [51]. Figure 12 shows a causal diagram where the text encodes confounders. These can then be measured from the text and therefore be used for adjusting the causal relationship between treatment and outcome.

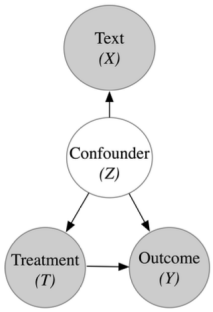


Figure 12: Causal Diagram with Text
Source: [51, p.5332].

Using text for causal inference presents some challenges. First and foremost, text is inherently high-dimensional, as stated above, and therefore suitable methods to adjust for high-dimensional confounders need to be chosen [51]. Second, as also explained above, text cannot be used directly in most cases, but must first be converted into a representation, e.g., by embeddings [51]. Furthermore, Keith et al. [51] state that empirical analysis of causal inference is still an active area of research and that using text in this task brings additional challenges for the evaluation.

Despite the not negligible chance of using text to measure confounders, until a

few years ago there were only a limited number of interactions between the fields of NLP and causal inference [51]. Exemplary, Kiciman et al. [54] use observational text data from Twitter to measure confounders for their evaluation of the causal effect of alcohol on academic success. As the use of alcohol can ethically not be randomly assigned to students, the Twitter data is used to control for alcohol mentions and success indicators like emotions or study behaviour [54].

Keith et al. [51] present two options to measure confounders via an observed text. On the one hand, the confounders that should be measured from the text can be pre-specified, if possible [51]. When pre-specifying confounders one can simply use a classifier to derive whether a particular state of the confounder (e.g. the mental state depression) is present in an observed text [51]. Afterwards, it is only necessary to condition on the confounders measured by the classifiers. On the other hand, confounders can be learned inductively in order to condition for all aspects of the observed text [51]. Using this approach, textual information is summarized with representation just as described above [51]. Therefore, embeddings like word2vec or more sophisticated models like BERT seem suitable for this task. For inductively derived confounders, each text representation is used as a confounder in a method for causal adjustment [51].

Besides using text to remove confounding from causal estimates, there is also recent research on how to directly use text for causal inference problems where text is the outcome or even the treatment. Since this use of text for causal inference is not discussed further in this thesis, the reader is referred to Egami et al. [24] for more details.

2.6 Image Data and their Use in Causal Inference

In addition to the theoretical foundations provided for text data in section 2.5, this section shall provide a brief introduction to images as a dataform and their use in causal inference. This section is already anticipating the setting of the case study in section 6.

In general, image data can be represented as a two-dimensional grid of pixels which can either be in colour or only monochromatic [125]. Thereby, each pixel represents a specific numerical value or multiple values to indicate the colour brightness [125]. Colour images hereby consist usually of three channels to reflect the colours green, red, and blue [125]. There are various DL methods and strategies to deal with images and to exploit the spatial structure rather than just treating each pixel location as a feature. Ideally one would use the prior knowledge that nearby pixels are related to each other in order to come up with efficient models to learn from image data [125]. Typical neural networks that are well designed for the use with image data are Convolutional Neural Network (CNN)s which are efficient with regard to the number of parameters and which show good performance in various computer vision tasks [125].

Similar to NLP there are also more sophisticated models developed to obtain vision representations. In the style of BERT, Bidirectional Encoder Representations from Image Transformers (BEiT) has been developed as Transformers, in general, has shown promising performance in computer vision tasks [10]. Since traditional Transformers

showed to require much training data, BEiT proved to be one promising solution to use image data [10]. As BERT can not be applied directly to image data because of various challenges like the non-existence of suitable vocabularies for vision Transformers input unit, BEiT has been introduced [10]. Bao et al. [10] suggest that the original image is tokenized and that the image patches are masked and fed into Transformers. Therefore, the BEiT model consists of a hybrid architecture that includes a convolutional backbone for processing the image, followed by a Transformer-based architecture for processing the image features and generating the final classification [10]. Before pre-training a Vision Transformer (ViT), an image tokenizer is learned through an autoencoding-style reconstruction process that segments images into discrete visual tokens based on a learned vocabulary [10]. For pre-training, each image is presented in two views: image patches and visual tokens [10]. A certain proportion of the image patches are randomly masked, represented by a specific mask embedding, before the image patches are fed into the Transformer backbone [10]. The pre-training task focuses on predicting the visual tokens of the original image based on the corrupted image patches [10]. BEiT provides a pre-trained model solution where the model parameters then can be fine-tuned for a specific task [10]. The BEiT model achieves state-of-the-art performance on a number of image classification benchmarks [10]. The performance is achieved through the aforementioned use of a Transformer-based architecture, which is able to model the relationships between different parts of the image and generate more accurate predictions based on context [10]. Overall, the BEiT model represents a significant advance in image classification and demonstrates the potential of transformer-based architectures to achieve state-of-the-art performance in a range of tasks.

Image data is becoming increasingly prevalent in various different settings and fields of research [43]. Just as text data, image data is omnipresent through digital services and online advertisement [43]. As already referred to in section 2.1, studying causal effects with observational data requires in general the adjustment of confounding factors. Especially in non-tabular settings, where the confounding factors are not necessarily well-defined and potentially unstructured, confounding brings along challenges [43]. For example, in medical settings, images present an interesting opportunity for causal inference since images can be related to the treatment as well as to the outcome [43]. Therefore, images are inducing a confounding bias if not adjusted for [43]. According to Jerzak et al. [43], there have been limited methodological frameworks for causal inference that presume confounding to be caused by image patterns or objects. One example where images may induce confounding to a causal inference problem is that doctors base their treatment of a patient on visuals like x-ray images [43]. Therefore, the treatment is guided by certain latent patterns in the image but also the outcome, for example, the chance of survival or cure depends on the information in the image. Referring to Figure 1, the latent objects and patterns in the X-ray image represent confounders in the causal graph. As images are often unlabeled in real-world applications it is difficult to adjust for patterns or objects in images directly [114]. There are fairly new endeavours to develop techniques for causal inference under image pattern confounding [43]. Using images to obtain unobserved confounders is motivated by the resolution of each image which is usually more granular than the actual unit of the analysis [43]. Therefore, images can be used to reconstruct the unobserved confounders [43].

3 Implementation of Deep Learning Approaches in DoubleML Framework

The following section describes the implementation of DL approaches as learners in the DML package. This thesis focuses on the DML package for Python and implements DL methods in an object-oriented and generic way. It is also possible to implement DL methods in the DML package for R which shall not be touched on within this thesis due to the limited scope. It is worth mentioning that the implementation of DL methods for the R package is a straightforward next step, as the relevant R packages and libraries refer back to Python for the use of DL methods. Therefore, it can be assumed that an implementation of DL methods in the DML package for R may be easy and without severe obstacles after a successful implementation in the DML package for Python.

When it comes to the implementation of DL methods in Python, there are different potential DL frameworks to be considered. Among others, some popular and widely used frameworks are PyTorch, TensorFlow and Keras. Each one of these frameworks has different benefits and strengths.

PyTorch is an end-to-end ML framework with an optimized tensor library for DL [85]. PyTorch is used for many modern DL projects and shows to become increasingly popular among researchers [106]. Furthermore, some sources describe PyTorch even to be the preferred solution for academic research and to be the most widely used DL framework in research [46], [75]. PyTorch especially stands out compared to other frameworks due to its flexibility and debugging capabilities [106]. In addition, PyTorch is pythonic which facilitates its use for software engineering tasks [53].

TensorFlow is an end-to-end open-source platform for ML [1]. Within TensorFlow, various abstraction levels for building and training neural network models are available [106]. Furthermore, it is said to be very well documented [106]. Raw TensorFlow is verbose and not explicit which may make it hard to understand the underlying principles [75].

Keras is a high-level DL API that is running on the ML platform TensorFlow [52]. Therefore, Keras acts as a wrapper for the TensorFlow framework, as the Keras interface is simpler, more modular, and thus more user-friendly. [106]. Keras is said to be the most popular DL framework due to its comparative simplicity [106].

This thesis focuses on the DL framework PyTorch due to its increasing popularity among researchers. PyTorch has also advantages compared to the other frameworks due to the low API and the good debugging capabilities [106] as well as being pythonic [53]. These benefits weigh heavily because implementing DL approaches into a fairly complex framework such as the DML framework and into existing packages may require understanding and insights at a more detailed level as well as at the code level. This makes PyTorch not only an excellent framework for researchers but also for developers [53]. However, as Keras is more high-level, readable, and concise [75], it is easy to use

for beginners and people without in-depth software knowledge about DL. Therefore, DL approaches are also implemented in the DML Python package using the Keras API in order to make the application of DL approaches for causal inference available for a broader range of users. Nevertheless, it is to bear in mind that from here on the following descriptions and explanations in the further sections of this thesis shall refer to the implementation using the PyTorch framework as described in subsection 3.1.

3.1 Implementation using PyTorch/ Skorch

As described in subsection 2.3.7, the DML package for Python is built on top of scikit-learn. In order to use PyTorch with scikit-learn, skorch can be used. Skorch can be defined as a "scikit-learn compatible neural network library that wraps PyTorch" [108]. By using the high-level library skorch, the flexibility of PyTorch is combined with the clean interface of scikit-learn [108]. Due to the full compatibility with scikit-learn, which is the basis for the DML package, the implementation of DL approaches should be primarily built on top of skorch.

Using skorch-regressors directly from skorch to apply in the DML package, results in an error indicating that the dimension of the target data does not fit the regression target. Reshaping the data prior to creating the DML data object does not solve this error as the data is reshaped again in the DML package (refer to DML package for Python, line 224 in `double_ml_data` module). Therefore, the use of skorch-regressors out of the box is not possible.

To cope with the error that has occurred, the implementation of the skorch-regressors is done via the **class** `NeuralNetRegressorDoubleML` from the newly developed `dml_nn` package (see Appendix C.9). Only the dimensions of the input and output data have been changed as shown below so that the `fit` and the `predict` method is compliant with the requirements of the DML package.

```
from skorch import NeuralNetRegressor

class NeuralNetRegressorDoubleML:
    _estimator_type = "regressor"

    def __init__(self, *args, **kwargs):
        self.torch_nn = NeuralNetRegressor(*args, **kwargs)

    def set_params(self, **params):
        self.torch_nn.set_params(**params)
        return self

    def get_params(self, deep=True):
        dict = self.torch_nn.get_params(deep)
        return dict
```

```
    def fit(self, X, y):
        y = y.reshape(-1, 1)
        self.torch_nn.fit(X, y)
        return self

    def predict(self, X):
        preds = self.torch_nn.predict_proba(X)
        return preds.reshape(-1)
```

The graph in Figure 13 shows the simple class structure of the adjusted API. The respective software implementation of the `dml_nn` package is available in the `DMLSim` library in Appendix C.9.

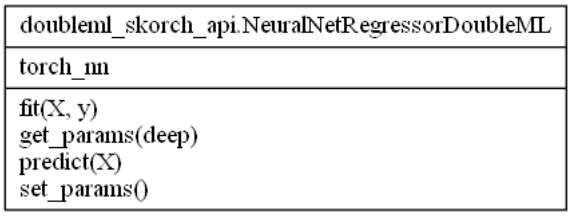


Figure 13: Class NeuralNetRegressorDoubleML

Source: own figure.

The use of skorch-classifiers in the DML package works without any further changes. It is additionally to be mentioned that PyTorch usually works with `float32` due to the computational demand and performance issues that would arise if one has a data type with a higher resolution. The generated data for the DML package has generally a precision of `float64`. This will lead to a `dtype` mismatch which is why the data type needs to be changed in order to be compatible with PyTorch.

3.2 Validation of Skorch Implementation

In the following section, the implementation of DL approaches in the DML package using skorch as described in section 3.1 shall be validated. This section aims to prove the correct implementation of DL methods prior to any further investigations about their use in causal inference settings. In anticipation of further discussion in the simulation study in section 4, the implementation of DL methods is validated using relatively simple examples of a linear regression.

As a first step, the modified version of the `class NeuralNetRegressor` from the skorch library, `class NeuralNetRegressorsDoubleML`, will be compared to the originally implemented version. The data for the unmodified API was adjusted accordingly so that the same behaviour of the APIs can be expected.

```
import torch
import numpy as np
import doubleml as dml
from torch import nn
from torch import optim
from torch.linalg import lstsq
from doubleml.datasets import make_plr_CCDDHNR2018

np.random.seed(1312)
(x, y, d) = make_plr_CCDDHNR2018(alpha=0.5,
                                 n_obs=500,
                                 dim_x=20,
                                 return_type='array')
x = x.astype('float32')
y = y.astype('float32')
d = d.astype('float32')

lr=0.01
n_train = 20
n_batch = 10

class Regressor(nn.Module):
    def __init__(self):
        super(Regressor, self).__init__()
        self.first_layer = nn.Linear(20, 1)

    def forward(self, x):
        X = self.first_layer(x)
        return X

from skorch import NeuralNetRegressor
np.random.seed(1234)
torch.use_deterministic_algorithms(True)
torch.manual_seed(1234)
learner_skorch = NeuralNetRegressor(module=Regressor,
                                    criterion = nn.MSELoss,
                                    optimizer = optim.SGD,
                                    optimizer__lr = lr,
                                    batch_size = 10,
                                    max_epochs=n_train,
                                    verbose=0,
                                    device='cpu')

learner_skorch.fit(x,y.reshape(-1,1))
skorch_score = learner_skorch.score(x,y)
```

```
skorch_score
0.4287868609588015

from dml_nn.doubleml_skorch_api import NeuralNetRegressorDoubleML
np.random.seed(1234)
torch.use_deterministic_algorithms(True)
torch.manual_seed(1234)
learner_skorch_modified = NeuralNetRegressorDoubleML2(module=Regressor,
                                    criterion = nn.MSELoss,
                                    optimizer = optim.SGD,
                                    optimizer__lr = lr,
                                    batch_size = 10,
                                    max_epochs=n_train,
                                    verbose=0,
                                    device='cpu')

learner_skorch_modified.fit(x,y)
skorch_score_mod = learner_skorch_modified.score(x,y)
skorch_score_mod
0.4287868609588015
```

As can be seen in the code example above, the two APIs return the same R^2 score (≈ 0.4288). Therefore, the behaviour of the modified **class** `NeuralNetRegressorsDoubleML` is as intended and expected.

The next step is to examine the interaction of the skorch library with the DML package. For this purpose, a learner from the scitkit-learn library is compared with a neural network. The neural network **class** `Regressor` defined above and the **class** `SGDRegressor` from scikit-learn, which fits a linear model using SGD instead of Ordinary Least Squares (OLS), are suitable for this purpose. By its structure defined as a 1-layer network without any activation function, the behaviour of **class** `Regressor` is expected to be similar to a linear regression. The following code continues from the one above:

```
import doubleml as dml
from sklearn.linear_model import SGDRegressor
from sklearn.base import clone

obj_dml_data = dml.DoubleMLData.from_arrays(x, y, d)

learner_sgd = SGDRegressor(loss='squared_error',
                        penalty=None,
                        fit_intercept=True,
                        max_iter=n_train,
                        tol=None,
                        shuffle=False,
```

```
                        verbose=0,
                        average=10,
                        learning_rate='constant',
                        eta0 = lr)

ml_l = clone(learner_sgd)
ml_m = clone(learner_sgd)

dml_plr_SGDRegressor = dml.DoubleMLPLR(obj_dml_data,
                                       ml_l,
                                       ml_m,
                                       apply_cross_fitting=True,
                                       n_folds=3)
print(dml_plr_SGDRegressor.fit())

================== DoubleMLPLR Object ==================
...
------------------ Fit summary       ------------------
      coef   std err          t         P>|t|    2.5 %   97.5 %
d  0.57035  0.038873  14.672059  9.734617e-49  0.49416  0.64654


ml_l = clone(learner_skorch_modified)
ml_m = clone(learner_skorch_modified)
np.random.seed(1234)

dml_plr_Regressor = dml.DoubleMLPLR(obj_dml_data,
                                    ml_l,
                                    ml_m,
                                    apply_cross_fitting=True,
                                    n_folds=3)
print(dml_plr_Regressor.fit())
================== DoubleMLPLR Object ==================
...
------------------ Fit summary       ------------------
      coef   std err          t         P>|t|     2.5 %    97.5 %
d  0.56835  0.039304  14.460196  2.161800e-47  0.491315  0.645386
```

As can be seen, the two models produce at least similar results for the causal effect and the other statistical attributes, suggesting a correct implementation. The small deviations can be attributed to differences in the SGD algorithms. These may lead to different convergence. If SGD is considered as a stochastic approximation of the gradient descent method, it is, therefore, plausible that there are certain differences.

3.3 Implementation and Validation using Keras/ Scikeras

Just as skorch wraps PyTorch to be compatible with scikit-learn, scikeras makes it possible to use Keras/ TensorFlow with scikit-learn [99]. As described in section 3, Keras acts as a wrapper for the TensorFlow framework. Scikeras wraps Keras and has a scikit-learn interface [99]. The goal of scikeras is to be as flexible as Keras in combination with having the clean interface of scikit-learn [99]. Due to the compatibility with scikit-learn, which is the basis for the DML package, the implementation of DL approaches should be additionally built on top of scikeras.

The use of scikeras works directly out of the box since no error is raised. The correct interaction of Keras and the DML package is checked using the same procedure as for skorch (see section 3.2). The following code builds upon the preceding code:

```
from tensorflow import keras
from scikeras.wrappers import KerasRegressor
from sklearn.base import clone
import tensorflow as tf

def regr_model():
    model = keras.Sequential()
    model.add(keras.layers.Dense(1, input_dim=20, activation=None))
    model.compile(loss='mse',
                  optimizer='sgd')
    return model

skkeras_model = KerasRegressor(model=regr_model,
                        epochs=n_train,
                        batch_size=n_batch,
                        verbose=0)

ml_l = clone(skkeras_model)
ml_m = clone(skkeras_model)
np.random.seed(1234)
dml_plr_keras= dml.DoubleMLPLR(obj_dml_data,
                               ml_l, ml_m, apply_cross_fitting=True,
                               n_folds=3)
print(dml_plr_keras.fit())

================== DoubleMLPLR Object ==================
...
------------------ Fit summary       ------------------
       coef    std err          t         P>|t|     2.5 %     97.5 %
d  0.572301   0.038667  14.800882  1.445741e-49  0.496516   0.648086
```

It can be seen that the scikeras model using Keras regressors produces similar results for

the causal effect and the other statistical attributes compared to the skorch model and the linear SGD model. This indicates a correct implementation of DL approaches in the DML package using the Keras API. As described above, the small deviations can be attributed to differences in the SGD algorithms and are therefore plausible.

4 Simulation Study

In the following section, a simulation study is conducted in order to examine the behaviour of the newly implemented DL approaches in the DML package as described in section 3. By this, it is also intended to additionally validate the implementation in different settings. In the existing literature, it is a common approach to validate and evaluate the performance of causal methods with synthetic data [81]. The design of such approaches using synthetic data typically involves a mechanism to generate data which allows access to the true treatment effect [81]. This effects ground truth can subsequently be used to evaluate the performance and accuracy of causal methods on simulated data [81]. A simulation study generates and uses synthetic data and is one suitable tool to validate the implementation of DL approaches in the DML package as functioning and appropriate for causal inference.

Simulation studies are hereby referred to as computer experiments that include data generation by pseudo-random sampling [80]. For a simulation study, a system or a method is repeatedly simulated [84] so that, for example, the same calculations are performed several times. Through the repetition of the simulation, it is possible to statistically estimate the probability characteristics of the obtained results. These results can ultimately be used for determining the performance of the system or the method [84]. According to Burton et al. [15], simulation studies are a useful tool for assessing the performance of different approaches in relation to a known truth. Therefore, simulation studies can be regarded as empirical experiments [80]. These days, simulation studies are used in various different disciplines and have become one essential tool in scientific research [9]. More specifically, simulation studies are widely used in statistical research to evaluate statistical methods as they can be utilized to obtain empirical results on the performance of methods in particular scenarios [80].

One advantageous point of a simulation study is that a statistical method can be analysed and the underlying methodology can be understood since the parameters of interest are known throughout the study. Furthermore, simulation studies are widely used in statistical research to compare different alternatives of methods and to enable researchers to evaluate novel methods [80].

In literature, the term simulation is used inseparably with the term modelling [39], [104]. Models used in a simulation are in this context defined as "approximations of the real world" [104, p. 3]. In other words, a model is a reasonable representation of a system that is to be analyzed [66]. Since this thesis is about evaluating a statistical method instead of analysing a physical system, the modelling aspect is not to be addressed in the further course. In fact, one could see the various methods implemented in the DML framework as models that are to be evaluated.

Occasionally in the literature, the terms simulation studies and simulation experiments are used synonymously. In this thesis, the term simulation experiment should only refer to the execution of the study in the meaning of conducting different simulation runs. This differentiation is made according to Lorig [66] who states that simulation

experiments do not compulsory have to follow a structured or systematic approach, while a simulation study requires for example the formulation of objectives in the first step. All further steps should then be guided by these objectives. While the simulation experiment represents an essential part of the simulation study, it can also be performed standalone [66]. Simulation experiments that use random sampling and statistical modelling are often also referred to as Monte Carlo simulation experiments [36].

The presented main advantages of a simulation study make it an invaluable tool for this thesis. This stems from the fact that this thesis is not only about proving and evaluating the new implementation of DL approaches in the DML package but also about comparing the results to other implemented ML methods. Therefore, in the following sections, a respective simulation study shall first be planned and designed and then executed in the sense of a simulation experiment before the results can be analysed and evaluated.

4.1 Structure for Planning and Design of the Simulation Study

In order to achieve a satisfactory level of quality in simulation studies, it is beneficial to establish guidelines as theoretical frameworks [66]. The guidelines for preparing and executing simulation studies facilitate generating reliable results which then can be a profound basis for the analysis [66]. Additionally, the guidelines provide recommendations or basic concepts for simulation studies but, in most cases, do not describe a step-by-step process to follow [66].

Few published simulation studies provide sufficient information and details about the study [15]. This lack of insight makes it hard for readers to understand the processes required when designing a study [15]. In order to provide a structured procedure, the main steps for setting up a simulation study are defined. These key steps in planning and designing a simulation study that is fitting the purpose of this thesis shall be following the steps: Aims, Data-generating mechanism, Methods, Estimands, and Performance measures (ADEMP) as a structured approach introduced by Morris et al. [80]. The ADEMP structure is adapted to the requirements of this thesis. In the following, the key steps for the planning and design of a simulation study are highlighted:

1. Planning
    a) Specify the objectives of the simulation study
    b) Specify a DGP
        - Determine which factors to vary
        - Determine the levels in which the factors shall be varied
        - Decide whether factors should be varied factorially or one-at-a-time
    c) Define estimands and targets of the simulation study
    d) Specify which methods to be evaluated/ compared
    e) Specify performance measures to be estimated

f) Specify number of simulation repetitions

2. Design
   a) Separate scripts that are used for analyzing simulated data from scripts for analyzing other datasets
   b) Build up code for simulation study
   c) Include recurring checks while code set-up

This structured approach and the individual design of the key steps for this study allow for meeting the requirement of reproducibility that is very important for scientific methods [66]. Applied to a simulation study, this requirement states that the observed behaviour should be the same when two or more persons perform the same study [66]. In fact, there might be no necessity for equal results in individual simulation runs but more a reproducibility of the final results [66]. In order to ensure the reproducibility of the simulation study, it is necessary to explicitly state all scenarios and instrumentation that were utilized [20]. These specifications will be presented in the upcoming sections according to the key steps above.

To clarify the procedure for the later execution of the simulation study, Figure 14 outlines the different phases that are to be considered. These phases are described in more detail in section 4.4 but are nevertheless important to consider when planning and designing the simulation study.

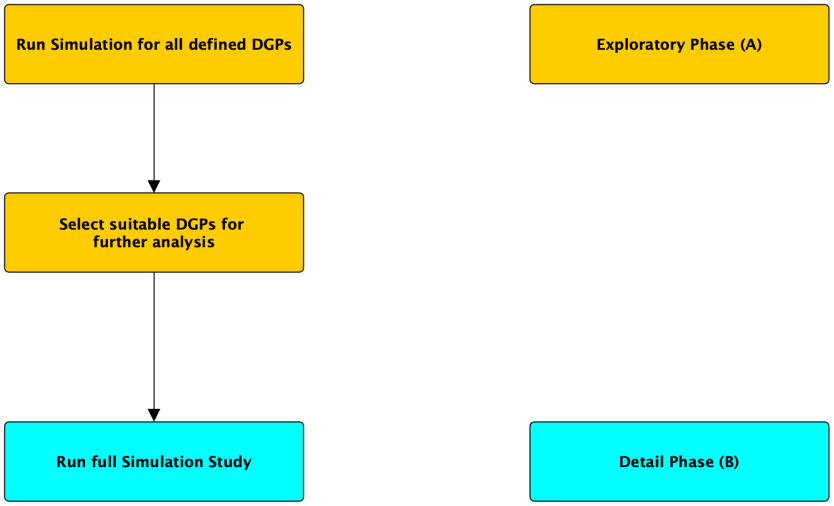


Figure 14: Outline Phases of Simulation Study

Source: own figure.

4.2 Planning of the Simulation Study

The presented simulation study in this thesis shall partly follow the study *"Deep neural networks for estimation and inference"* of Farell et al. [26]. Furthermore, the present simula-

tion study shall refer in some design and parameter choices to the study about *"Evaluating Uses of Deep Learning Methods for Causal Inference"* of Whata et al. [118]. These studies are referenced for some of the present study's design and certain parameter selections. Nevertheless, the simulation study in this thesis goes beyond evaluating the use of DL approaches in the context of CML and in comparison to other ML methods.

4.2.1 Specify the Objectives of the Simulation Study

To begin with, it is crucial to define the objective of the simulation study and to identify the primary focus of the investigation. Recognizing and stating the underlying problems of the study, which are to be investigated ensures a targeted simulation experiment in the study [66].

First of all, the simulation study shall prove the successful implementation of the DL approaches as learners in the DML package as described in section 3 for an extended scope of settings. This objective remains a central purpose throughout the simulation study. Additionally, following Farell et al. [26, supplement p. 1], the simulation study should generally analyse the "finite sample properties of deep neural networks and their use in semiparametric causal inference". Therefore, the aim is to study inference on the causal effect for different ML and DL methods. Besides, this thesis aims at putting the newly implemented DL approaches in comparison to other ML methods that have been formerly implemented in the DML package.

4.2.2 Specify a Data Generating Process

Generating data is an essential aspect of simulation studies. In the absence of real-world or experimentally collected data, one would need a mechanism or procedure to generate random numbers that can be used to create a dataset for the simulation study [80]. As this data sets the basis for the complete study, the specification and decision-making process on a suitable DGP is of great importance. The data can potentially be generated by a parametric model or by repeated re-sampling with replacement from a given dataset [80]. For the latter the true data-generating model is unknown. Morris et al. [80] found that most simulation studies relied on a parametric model to generate the data.

Since the choice of a DGP depends on the aims of the study [80], it is necessary to refer back to the above subsection 4.2.1. Since one of the objectives is to demonstrate a successful implementation of DL approaches in the DML package, a DGP that has already been used in the context of DML might be sufficient. Without introducing excessive complexity to the study through a complex DGP it might be easier to validate the correct implementation of the new concepts in an existing framework. As also referenced, the simulation study should partly be in the style of Farell et al. [26]. Farell et al. [26] use linear as well as nonlinear models and different numbers of covariates to generate the data resulting in eight DGP designs. Using more than one DGP and even more complex ones might also be beneficial when looking at the last stated objective. One would aim for different scenarios in order to properly compare the results arising from the use of DL approaches for causal inference to

the results emerging from other formerly implemented ML concepts. To ensure the coverage of different scenarios simulation studies can generally involve several DGPs [80].

In the following, various DGPs, that are not only suitable but also beneficial to be used in the simulation study for this thesis, shall be specified. As it is far more common in practical research, it is focused on a parametric model to generate the data.

In conducting the simulation study in section 4.4, an exploratory phase first identifies which DGP to focus on for the further course of the study (see Fig. 14). According to the above explanations, several different DGPs shall be used in order to:

1. Prove the successful and correct implementation of DL methods in the DML package for various different settings.
2. Find a setting where the use of DL methods looks promising in the context of CML.

During the initial exploratory phase (see Fig. 14), the simulation study is not carried out in its entirety as outlined in the subsequent subsections. Instead, the aim is to identify a suitable DGP for use in the detail phase of the study. This methodology is designed to concentrate the simulation study on a meticulously selected DGP rather than conducting a simulation study that yields little value.

As described, a more simple DGP or a DGP, that is already proven in the DML package, could suffice for the first part of the study, namely proving the successful and correct implementation of DL methods in the DML package. In addition, more complex DGPs may make it possible to find a setting where the use of DL methods is beneficial.

By screening the literature on CML one can find several DGPs that can be potentially used in this thesis. The following paragraphs shall give an overview of which DGPs to include in the exploratory phase. Anticipating the further explanations in subsection 4.2.4, only DGPs for the causal models PLR and IRM are presented. Further explanations about the implementation of these DGPs will follow in section 4.3.

It is important to state that for all considered DGPs all taken samples are independent and identically distributed (i.i.d.) in each case. This means that the samples assume the same values with equal probability without influencing each other for a respective DGP.

plr_CCDDHNR2018

The DGP plr_CCDDHNR2018 (n_{obs}, dim_x, α) is used to generate data from a PLR model as used in Chernozhukov et al. [17]. The data generation can be described by the following equations which all refer to Chernozhukov et al. [17]:

$$y_i = \alpha d_i + g_0(x_i) + s_2 u_i, \qquad u_i \sim \mathcal{N}(0,1) \tag{37}$$

$$d_i = m_0(x_i) + s_1 v_i, \qquad v_i \sim \mathcal{N}(0,1) \tag{38}$$

The covariates X are normally distributed, $x_i \sim \mathcal{N}(0, \Sigma)$, with $\Sigma_{kj} = 0.7^{|j-k|}$. The nuisance functions are described as:

$$m_0(x_i) = a_0 x_{i,1} + a_1 \frac{\exp(x_{i,3})}{1 + \exp(x_{i,3})} \tag{39}$$

$$g_0(x_i) = b_0 \frac{\exp(x_{i,1})}{1 + \exp(x_{i,1})} + b_1 x_{i,3} \tag{40}$$

The factors to define for this DGP are the value of the causal parameter α, the number of covariates dim_x, and the number of observations to simulate n_{obs}. Per default the parameters s_1, s_2, a_0, and b_0 are set to 1 and a_1 as well as b_1 are set to 0.25. This DGP is already implemented and used in the DML package.

irm_data

The DGP irm_data $(n_{obs}, dim_x, \theta, R2_d, R2_y)$ is used to generate data from an IRM [7], [13]. The data generation can be described by the following equations which all refer to Bach et al. [7] and Belloni et al. [13]:

$$y_i = \theta d_i + c_y x_i' \beta d_i + u_i, \qquad u_i \sim \mathcal{N}(0, 1) \tag{41}$$

$$d_i = 1\left\{ \frac{\exp(c_d x_i' \beta)}{1 + \exp(c_d x_i' \beta)} > v_i \right\}, v_i \sim \mathcal{N}(0, 1) \tag{42}$$

The covariates X are normally distributed, $x_i \sim \mathcal{N}(0, \Sigma)$, with $\Sigma_{kj} = 0.5^{|j-k|}$. The heterogeneity in the treatment effect arises from the vector $\beta_j = \frac{1}{j^2}$ with dim_x entries and for $j = 1, ..., dim_x$. The two scalars c_y and c_d are describing the strength of the relationship between the covariates X, the treatment D, and the outcome Y [13]:

$$c_y = \sqrt{\frac{R_y^2}{(1 - R_y^2)\beta' \Sigma \beta}} \tag{43}$$

$$c_d = \sqrt{\frac{(\pi^2/3) R_d^2}{(1 - R_d^2)\beta' \Sigma \beta}} \tag{44}$$

The factors to define for this DGP are the value of the causal parameter θ, the number of covariates dim_x, the number of observations to simulate n_{obs}, and the value of the parameters R_y^2 and R_d^2. Per default, the parameters R_y^2 and R_d^2 are set to 0.5. This DGP is already implemented and used in the DML package.

plr_turrell2018

The DGP plr_turrell2018 (n_{obs}, dim_x, θ) is used to generate data from a PLR model as used in Turell [111]. The data generation can be described by the following equations which all refer to Bach et al. [7] and Turell [111]:

$$y_i = \theta d_i + g_0(x_i'b) + u_i, \qquad u_i \sim \mathcal{N}(0,1) \tag{45}$$
$$d_i = m_0(x_i'b) + v_i, \qquad v_i \sim \mathcal{N}(0,1) \tag{46}$$

The covariates X are normally distributed, $x_i \sim \mathcal{N}(0, \Sigma)$, with Σ being a random symmetric and positive-definite matrix which is generated by a sklearn function. The nuisance functions are described as:

$$m_0(x_i) = \frac{1}{2\pi} \frac{\sinh(\gamma)}{\cosh(\gamma) - \cos(x_i - \nu)} \tag{47}$$

$$g_0(x_i) = \sin(x_i)^2 \tag{48}$$

The vector $b_j = \frac{1}{j^2}$ has dim_x entries for $j = 1, ..., dim_x$. The factors to define for this DGP are the value of the causal parameter θ, the number of covariates dim_x, and the number of observations to simulate n_{obs}. Per default, the parameter ν is set to 0 and γ is set to 1. This DGP is already implemented and used in the DML package.

irm_friedman

The DGP irm_friedman (n_{obs}, dim_x) is utilized to create data from a five-dimensional test function. This function was initially introduced to assess the predictive ability of various methods and has since been expanded to evaluate the performance of causal inference techniques on highly non-linear outcomes [81]. This test function roots back to early works from Friedman et al. [30] and Friedman [29]. The DGP is used in several papers to simulate data for various ML purposes, for example in Chipman et al. [18], or, more specifically, to evaluate causal inference methods in Parikh et al. [81]. The data generation can be described by the following equations which mainly refer to Parikh et al. [81]:

$$y_i = \theta(x_i) d_i + g(x_i) + u_i, \qquad u_i \sim \mathcal{N}(0,1) \tag{49}$$
$$d_i \sim \mathcal{B}(prob(x_i)) \tag{50}$$

The probability for the individual treatment assignment d is dependent on the first two covariates and can be described by the following equation:

$$prob(x) = \frac{1}{1 + e^{-(x_0 + x_1 - 0{,}5)}} \tag{51}$$

The heterogeneous treatment effect $\theta(x)$ is dependent on three covariates via the following function:

$$\theta(x) = x_3 \cos(\pi x_1 x_2) \tag{52}$$

The nuisance functions can be described as:

$$g(x) = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0,5)^2 + 10x_4 + 5x_5 \tag{53}$$

The covariates X are uniform distributed i.i.d. samples:

$$x = (x_0, x_1, ..., x_n) \quad i.i.d \quad \sim \mathcal{U}(0,1) \tag{54}$$

The factors to define for this DGP are the number of covariates dim_x and the number of observations to simulate n_{obs}. As far as the authors are aware, there is no software implementation of this DGP available as open source.

plr_fingerhut2018

The DGP plr_fingerhut2018 (n_{obs}, dim_x, θ) is used to generate data from a PLR model as used in Fingerhut et al. [28]. The data generation can be described by the following equations which mainly refer to Fingerhut et al. [28]:

$$y_i = \theta d_i + g(x_i) + u_i, \qquad u_i \sim \mathcal{N}(0,1) \tag{55}$$
$$d_i = m(x_i) + v_i, \qquad v_i \sim \mathcal{N}(0,1) \tag{56}$$

The nuisance functions contain a case distinction and can be described as:

$$m(x) = \begin{cases} x_1 + 10x_3 + 5x_6 & \text{if } X \in \text{majority} \\ 10x_1 + x_3 + 5x_6 & \text{if } X \in \text{minority} \end{cases} \tag{57}$$

$$g(x) = \begin{cases} x_0 + 10x_2 + 5x_5 & \text{if } X \in \text{majority} \\ 10x_0 + x_2 + 5x_5 & \text{if } X \in \text{minority} \end{cases} \tag{58}$$

The covariates are sampled from a standard Gaussian auto-regressive process of order one with the correlation parameter $\rho = 0.8$. The simulated data is then divided into two disjoint groups by comparing the value of the first feature X_0 to a fixed threshold. The threshold is calculated in such a way that ≈ 75 % of the observations are assigned to the majority group and ≈ 25 % of the observations are assigned to the minority group.

The factors to define for this DGP are the value of the causal parameter θ, the number of covariates dim_x, and the number of observations to simulate n_{obs}. This DGP has an available open-source software implementation which is linked in Fingerhut et al. [28].

irm_farell2021

The DGP irm_farell2021 (n_{obs}, dim_x) is used to generate data from an IRM as used in Farell et al. [26]. The data generation can be described by the following equations which

mainly refer to Farell et al. [26]:

$$y_i = \theta(x_i)d_i + g(x_i) + u_i, \quad u_i \sim \mathcal{N}(0,1) \tag{59}$$
$$d_i \sim \mathcal{B}(prob(x_i)) \tag{60}$$

The probability for the individual treatment assignment d is dependent on all covariates and can be described by the following equation:

$$prob(x_i) = \frac{1}{1 + e^{-\alpha_p' x_i}} \tag{61}$$

The vector α_p has the length $dim_x + 1$ and 21 non-zero elements with the first entry being fixed, $\alpha_{p,1} = 0.09$, and the remainder $\alpha_{p,2}, ..., \alpha_{p,21}$ being i.i.d. $\sim \mathcal{U}(-0.55, 0.55)$.

The covariates vector has the length $dim_x + 1$ with the first entry being fixed, $x_1 = 1$, and the remainder $(x_2, ..., x_{dim_x+1})$ being i.i.d. $\sim \mathcal{U}(0,1)$.

The heterogeneous treatment effect $\theta(x)$ is dependent on the covariates via the following function:

$$\theta(x) = \alpha_\theta' x_i + \beta_\theta' \varphi(x_i) \tag{62}$$

In equation 62, φ describes a second-degree polynomial including pairwise interactions. The constant vector α_θ has the length $dim_x + 1$ with the first entry being fixed, $\alpha_{\theta,1} = -0.05$, and the remainder of α_θ being i.i.d. $\sim \mathcal{N}(0.1, 0.22)$. The vector β_θ has the length $dim_x + 1$ with all entries being i.i.d. $\sim \mathcal{U}(-0.05, 0.06)$.

The nuisance function can be described as:

$$g(x) = \alpha_g' x_i + \beta_g' \varphi(x_i) \tag{63}$$

In equation 63, the constant vector α_g has the length $dim_x + 1$ with the first entry being fixed, $\alpha_{g,1} = 0,09$, and the remainder of α_g being i.i.d. $\sim \mathcal{N}(0.3, 0.7)$. The vector β_g has the length $dim_x + 1$ with all entries being i.i.d. $\sim \mathcal{N}(0.01, 0.3)$.

Once being chosen, $\alpha_\theta, \alpha_g, \beta_\theta$ and β_g are kept the same for all simulation runs. The factors to define for this DGP are the number of covariates dim_x and the number of observations to simulate n_{obs}. This DGP has an available open-source software implementation which is linked in Farell et al. [26].

Opossum

The Opossum package as presented by Winkel et al. [120] provides a flexible and user-friendly software implementation for simulated DGPs. The motivation behind the development of this software is that model reaction, for example of neural networks, to data changes in a high-dimensional causal inference setting might be insightful for researchers [120]. By proposing an implementation in which all components are explained

and flexibly adaptable, researchers get support in evaluating and comparing different models applied to the generated dataset [120]. The Opossum package intends to provide a detailed backend and an easily-adjustable user interface for common scientific approaches [120]. In general, the Opossum package is built on the PLR model as described in subsection 2.3.1 by both equations 11 and 12. The treatment variable D is binary with $D \in \{0, 1\}$. Furthermore, it allows for heterogeneous treatment effects which are dependent on a subset of the covariates. There are several DGPs that can be created using the Opossum package. Among others, one can flexibly adjust the following principles for the DGP:

- Type of covariates (continuous, binary or categorical)
- Treatment assignment (random or based on covariates)
- Treatment effect (constant, continuous heterogeneous or discrete heterogeneous)
- Type of outcome variable (continuous or binary)
- Relationship between covariates and the outcome variable (linear simple, linear interaction, partial nonlinear simple, partial nonlinear interaction, nonlinear simple, nonlinear interaction)

For all underlying mathematical descriptions, it is referred to the documentation of the Opossum package by Winkel et al. [120]. For the exploratory phase six different DGPs based on the Opossum package shall be considered. The different DGP settings and specification of the above's principles as well as the causal model the DGP is used for are presented in Table 2. The various DGPs are named opossum_1, opossum_2, ... in order to be distinguishable.

Table 2: Specification of Opossum Data Generating Processes

Source: own table.

DGP	**Causal Model**	**Specification**
opossum_1	PLR	Constant treatment effect, continuous outcome and covariates, partial nonlinear interaction
opossum_2	PLR	Constant treatment effect, continuous outcome, continuous and ten binary covariates, partial nonlinear interaction
opossum_3	IRM	Continuous heterogeneous treatment effect, continuous outcome and covariates, nonlinear interaction
opossum_4	IRM	Continuous heterogeneous treatment effect, continuous outcome, continuous and ten binary covariates, nonlinear interaction
opossum_5	IRM	Discrete heterogeneous treatment effect, continuous outcome and covariates, nonlinear interaction
opossum_6	IRM	Discrete heterogeneous treatment effect, continuous outcome, continuous and ten binary covariates, nonlinear interaction

In this thesis, all six considered Opossum DGPs (see Table 2) are referring to a treatment assignment based on the covariates. Furthermore, only positive treatment effects are considered. For the PLR settings, the relationship between the covariates and the outcome variable is given as a partial nonlinear interaction. This relationship is characterized by the following equation, where X consists of individual covariates as well as random interaction terms of some of those covariates $x_i * x_j$ for $i, j \in \{1, ..., dim_x\}$:

$$y_i \sim 2.5 * \cos(X)^3 + 0.5 * X \tag{64}$$

For the IRM settings, the relationship between the covariates and the outcome variable is given as a nonlinear interaction. This relationship is characterized by the following equations, where X consists of individual covariates as well as random interaction terms of some of those covariates $x_i * x_j$ for $i, j \in \{1, ..., dim_x\}$:

$$y_i \sim 3 * \cos(X)^3 + 0.6 * X \tag{65}$$

In addition to the specifications in Table 2 and according to the user interface of the Opossum package, one must also specify a seed and the intensity to adjust the size of the treatment effect. In this thesis, for all six Opossum DGPs the seed is set to `None` so that the seed from the simulation study is used. Further details on the seed and reproducibility will be provided in section 4.3. The intensity is set to 5 in all cases resulting in a treatment effect of, for example, 1 for the constant case. It should be noted that, by design, the Opossum package assigns a treatment effect of zero to all observations without a treatment assignment. This needs to be considered when calculating the true ATE based on the generated data.

As stated, the Opossum package is an open-source software implementation that can be installed in a Python environment. It is worth noting that the Opossum package can also be used to study DGPs and scenarios with binary outcome variables. Due to the limited scope of this work, this is not yet considered but is certainly an interesting point for further investigation.

Generally and non-specific on any of the DGPs, it is worth mentioning that different simulation strategies are possible in terms of data generation when more than one method is involved. First, completely different and independent datasets can be generated for each method and scenario [15]. Besides that, in a moderately independent simulation, the same simulated dataset is used to compare various statistical methods for the same scenario, but for each scenario, a different dataset is generated [15]. The moderately independent designs are sensitive to detect any differences between the considered methods [15]. In the presented simulation study, such a moderately independent simulation design is followed as it supports the objective of comparing DL approaches to other ML methods.

Determine which Factors to vary

In the terminology of experimental designs, a factor is an input that can be varied during the course of an experiment [66]. The factors that one wishes to vary during the simulation study are often also referred to as potential design factors because these factors are considered to have an impact on the performance of a system or method [77]. The determination of factors to vary during the simulation study is directly influencing the specifications of the DGPs.

As already stated, Farell et al. [26] varied the numbers of covariates dim_x for the different DGPs to come up with eight scenarios. Generally speaking, since much can be controlled in a simulation study, there are several factors that can and will vary between different DGPs. One very common factor to deliberately vary is the sample size of the simulated dataset n_{obs}, as the performance often varies across datasets of differing sizes [80]. Varying different factors can be more informative than just taking one DGP because there might be interactions between factors than can be explored during the simulation study [80]. However, it is still important to consider practical implications that might make the variation of different factors difficult. These implications could exemplarily be that presenting the results might be infeasible when there are too many interaction effects between the factors or that the time for computing the simulation study exceeds the available resources [80]. Both implications need to be taken into account when determining which factors to vary. Especially the needed computational time needs to be considered for determining whether the factors should be varied in a fully factorial design or if the simulation study should follow a nonfactorial structure [80]. This aspect is discussed in more detail in the following parts.

In the detail phase of the simulation study the two common factors, the number of covariates dim_x and the sample size of the simulated dataset n_{obs} are varied to generate different scenarios. By this, it is intended to simulate different data settings like "tall" data settings, where n_{obs} is large and dim_x is small, or high-dimensional/ "wide" data settings, where dim_x is relatively large compared to n_{obs}. As stated in the objectives in subsection 4.2.1, the simulation study shall analyse the finite sample properties of DL approaches in CML and therefore it is insightful to consider different data settings. Besides, the results of the comparison with formerly implemented ML methods might depend on the data setting. The variation of the two factors dim_x and n_{obs} is also supported by pertinent literature such as Farell et al. [26] or Whata et al. [118].

The two factors, the number of covariates dim_x and the sample size of the simulated dataset n_{obs} are not varied in the exploratory phase (see Fig. 14) in order to not add too much complexity to this step. The exploratory phase focuses more on demonstrating a successful and correct implementation of DL methods in the DML package for different DGPs and finding a DGP where the use of DL methods looks promising in the context of CML. For this, a base case for the two factors is defined. Referring to the DML package and the associated documentation by Bach et al. [7], the base case is defined with $n_{obs} = 500$ and $dim_x = 20$.

In fact, through the exploratory step in executing the simulation study, where different DGPs are used (see Fig. 14), far more factors are varied subconsciously in the presented simulation study. This includes especially the data structure in the sense of the relationship between covariates and outcome as well as the type and scale of the covariates and outcome. As this variety of DGPs and the associated factors are merely considered in the exploratory phase and therefore are not varied in a structured way during the detail phase of the simulation study, this shall not be discussed further in the next planning steps. Nevertheless, the factors that vary with changing DGPs are obvious by referring to the mathematical descriptions of the different DGPs above. This multitude of additional variable factors through the exploration of different DGPs makes it possible to investigate the suitability of the newly implemented DL approaches for the DML framework for an even larger variety of different data settings.

Determine the Levels in which the Factors shall be varied

After determining which factors to vary it is indispensable to state the ranges and the specific levels in which the factors shall be varied and at which the simulation runs will be made [77]. In other words, it is important to determine the discrete values to use for the factors during the simulation study.

As stated above, the two main factors to vary during the detail phase of the simulation study (see Fig. 14) are the number of covariates dim_x and the sample size of the simulated dataset n_{obs}. The levels in which the factors shall be varied are in the style of Farell et al. [26] for dim_x and in the style of Whata et al. [118] for n_{obs} with one adjustment. As presented by Farell et al. [26], the highest factor level for n_{obs} is increased to 10000 in order to reflect big data settings:

- $dim_x = \{20; 100\}$
- $n_{obs} = \{500; 1000; 10000\}$

By these factor levels, it is possible to reflect six different data settings in the simulation study with a broad range of different dimensions and different amounts of data. This should serve the objective to analyse the finite sample properties of DL approaches in a CML application. Furthermore , this also allows for the comparison with other ML methods for several different data settings.

Decide whether Factors should be varied factorially or one at a time

As mentioned, it is worthwhile to have a deeper look into the design of the simulation study when it comes to the variation of the factors. The two extremes are on the one hand a full factorial design where all possible combinations of factor levels for all factors are used [102]. One advantage of a full factorial design is that through this design it is possible to estimate interactions between different factors. Additionally, this design entails a high external validity as several scenarios and conditions can be studied simultaneously [102]. The extreme, on the other hand, is a nonfactorial structure where only one factor is varied at a time [80]. Just as in the full factorial design several factors

are varied but just one at a time and not simultaneously [102]. A fully factorial design might not be feasible if the number of combinations of factors is simply too high. This could, for example, be due to a high demand for computational resources [102]. On the other hand, varying only one factor at a time assumes that there are no interaction effects as these are neglected and cannot be uncovered with such a factorial design [95]. To solve both issues, a compromise between the two extreme designs can be found by partially factorially varying the factors or by using a fractional factorial experimental design which is an incomplete design [102].

In the presented simulation study, a full factorial design is conducted in the detail phase. By this, all possible combinations of factor levels are considered so that the use of DL approaches in the DML package is investigated for six different data settings. A full factorial design can be chosen since the number of combinations of factors is rather low. This is due to the fact that an exploratory phase is preceding the detail phase and that in this exploratory phase, various other factors are varied in a rather unstructured way. Therefore, in the detail phase only the two above-mentioned factors dim_x and n_{obs} are varied which makes a fully factorial study computationally feasible due to the limited number of factor levels. The associated increased computational demand will be discussed in subsection 4.4.4.

4.2.3 Define Estimands and Targets of the Simulation Study

The term estimand describes the population quantity that is estimated by a method [80]. Since the simulation study is about the evaluation of a method and the comparison of one method to others, it is important to properly define the estimand according to the objectives of the study (see subsection 4.2.1). In fact, there can be more than one population quantity and therefore several estimands that are estimated. Usually, the estimand is a parameter of the DGP but it can for example also be one resulting measure [80]. Generally speaking, the estimand is often also referred to as the parameter of interest [80]. In addition to that, one can also define several other targets of the simulation study if this is needed to represent the objectives of the study in the study design. For example, a simulation study can not only be about the evaluation of a method to estimate an estimand but also about evaluating a method with regards to testing hypothesis, predicting an outcome, or model selection [80]. These are referred to as targets of the simulation study. Depending on the estimand or target of the simulation study different performance measures can be used [80]. This will be reflected in detail in section 4.2.5.

As stated in the objectives in subsection 4.2.1, the simulation study shall, among others, analyse the finite sample properties of DL approaches in a semiparametric causal inference context. Therefore, the estimand for the presented simulation study can be defined as the causal parameter $\hat{\theta}_0$ which can also be described as the ATE for some causal models. As shown in subsection 4.2.2, the true causal parameter is for most DGPs a parameter or factor that needs to be defined. For the irm_farell2021, the irm_friedman and several possible opossum_x DGPs the causal parameter is not manually set to a specific value, as the treatment effect is heterogeneous. Referring back to the objectives, no other targets need to be defined for the simulation study as the study focuses on the

evaluation of DL methods to estimate a causal parameter.

4.2.4 Specify which Methods to be evaluated/ compared

As stated at the beginning of section 4, one advantageous point of a simulation study, among others, is that statistical methods can be analysed or evaluated and also be compared to different other alternatives. Hereby a method can exemplarily describe a model for the analysis of data or a procedure for analyses [80]. If a simulation study aims to compare a method with other methods, it is crucial to incorporate comparators that are appropriate for the specific task being evaluated [80]. These other methods, which should ideally represent serious contenders, can be derived by analysing previous works in the respective area. It may only be necessary to include such methods in the simulation study that have already proved to be suitable for the considered application [80]. One common issue being mentioned in the literature is that, especially for new methods, there might not be an adequate software implementation of these methods to include in the simulation study [80]. Thereby, one important role of simulations is to verify the correctness of the code implementation of a method [80]. Another important point to mention when including several methods in a simulation study is that all methods need to estimate the same estimands [80].

Looking at the stated objectives of the simulation study in this thesis (see subsection 4.2.1) it becomes clear that there surely is more than one method to include in the simulation study. It is the objective to compare the newly implemented DL approaches to some formerly implemented ML methods. Despite the DL methods in the DML package being fairly newly implemented, there are no big issues foreseen. Morris et al. [80] state that the biggest issues with new methods are due to poor implementation and that these methods are not implemented in the same package as the comparators. As described in section 3, the implementation of the new DL approaches has been conducted with great care and initial validation. Furthermore, since the implementation was performed using the same version of the DML package as all other ML methods, there should not be such issues anticipated.

When looking at the methods to specify and choose for the present simulation study there are three main areas to consider:

1. What causal models to include in the simulation study
2. What ML methods to compare the DL approaches with
3. Which network architectures to use to represent the DL approaches

Due to the limited scope of this thesis, the simulation study shall focus on the two causal models PLR and IRM (see enumeration point 1 above). As described in section 2.3.1 the PLR model is used to investigate, as the name says, partially linear treatment effects. In a PLR model there are no IVs included. Additionally as described in 2.3.6., the IRM model focuses on settings with heterogeneous treatment effects and a binary treatment variable, also without any IVs. The limitation of the simulation study to these

two causal models has already been anticipated in subsection 4.2.2 when specifying the DGPs. The prefix of the DGP name states whether the DGP is used to generate data for a PLR model or an IRM. It should be noted that the DGPs that emerge from the Opossum package can generate data for either PLR models or IRMs. The choice of these two causal models is intended to demonstrate the applicability and suitability of DL methods for various models used in the DML framework and package. Furthermore, it is insightful to consider causal models with homogeneous as well as heterogeneous treatment effects for the simulation study. Despite the fact that models using IVs are not included in the presented simulation study, it is already worth mentioning that the simulation study can be easily extended to investigate the use of DL methods in these models as well. Section 4.3 presents the details of how the implementation of these extensions can be carried out in the simulation study design.

The newly implemented DL approaches will be compared with some previously implemented ML methods in terms of performance in different scenarios (see enumeration point 2 above). First of all, a linear model trained with Lasso regularization shall be considered in the simulation study to compare the DL methods to a basic ML method whose usage is already implemented in the DML package. Additionally, a Random Forest shall be used as a comparator as it is representing a more sophisticated ML method. A Random Forest shows favourable performance especially when the nuisance functions are rather smooth functions of a limited number of variables [17]. Furthermore, the inclusion of Random Forest methods in the simulation study is justified by the good performance in various numerical examples presented by Chernozhukov et al. [17]. For both, Lasso and Random Forest methods, default parameters are used to keep the methods generic as they are used on different DGPs and data settings. It should be pointed out that tuning the parameters may lead to better results, but given the objectives (see subsection 4.2.1) this should not be part of this simulation study. Certainly, the simulation study can be extended to compare DL methods against some other common ML methods whose usage is already implemented in the DML package like for example Boosting or Ensemble methods. It is mentioned that, just as described above, the simulation study in this thesis shall be flexibly designed to allow for easy implementation of such extensions. More on this will follow in section 4.3.

As described in the third point in the above's enumeration, it is also important to specify the network architectures used in the simulation study. For each scenario, a variety of networks shall be used to also evaluate differences in the performance of DL approaches. Due to the generic implementation of DL methods in the DML package, various different network architectures can be implemented. The presented simulation study shall only highlight a few different architectures but, as described before, the study can easily be extended by adding new network architectures. In reference to Farell et al. [26], all network architectures shall be MLPs using a ReLU activation function. The architectures vary in their width and depth and thus in their approximating power (see subsection 2.4). Since the search for suitable architectures is exhaustive and prevented by computational constraints, heuristic search methods are usually used to leverage a restricted searching budget [105]. One field of such methods is the Monte Carlo Tree Search (MCTS) method as presented for example by Wang et al. [115] or Su et al.

[105]. The following simulation study considers ten selected network architectures. An overview of these networks is given in Table 3. Five of these architectures (models 5 - 9) are following the presentation of Farell et al. [26], where Monte Carlo architectures have been explored. In addition to that, five fairly simple architectures (models 0 - 4) are used in this simulation study, as they have proven a good performance in some pre-tests. This variety of networks also fits the objective of studying inference for different DL methods. In addition, the different depths and widths of the presented network architectures can be used to find possible settings where more complex models perform better than simpler models.

The architectures in Table 3 are described by indicating the number of neurons per hidden layer. The exemplary architecture $\{4, 2\}$ is representing a neural network with two hidden layers where the first hidden layer consists of four neurons and the second hidden layer consists of two neurons. The depth of the networks is varied from 1 to 6 hidden layers and the width is varied from 1 to 80 neurons per layer.

Table 3: Network Architectures

Source: own table.

Architecture Number	**Structure**
Model 0	$\{1\}$
Model 1	$\{5\}$
Model 2	$\{10\}$
Model 3	$\{10, 5\}$
Model 4	$\{10, 5, 3\}$
Model 5	$\{20, 15, 5\}$
Model 6	$\{80, 80, 80\}$
Model 7	$\{60, 30, 20, 10\}$
Model 8	$\{20, 15, 15, 10, 10, 5\}$
Model 9	$\{60, 30, 20, 20, 10, 5\}$

It is worth mentioning that the ten hardcoded network architectures are just exemplary DL methods. Additionally, the networks are not considered to be exceptionally deep. Nevertheless, the simulation study is built up in a flexible way so that it easily can be extended to any network architecture.

The parameter settings for training the neural networks shall be in the style of Whata et al. [118]. In addition to the regularization method dropout, an early stopping as described in section 2.4 is implemented and a maximum number of epochs to run before stopping the training of the neural network is set. Of course, these parameters can be tuned in order to obtain better results but due to the limited scope of this thesis, the parameters are kept the same throughout the study and for each architecture. Nevertheless, it is to be mentioned that the simulation study in this thesis shall be flexibly designed to allow for an easy adjustment of parameters and regularization methods. The used parameters are presented in Table 4.

Table 4: Parameters for Training the Neural Networks
Source: own table, in the style of [118].

Parameter	**Values**
Dropout	0.2
Learning rate	0.001
Batch size	128
Number of epochs	100
Early stopping	10

As a default optimizer for the simulation study the algorithm Adam [56] is used. Adam is an efficient learning algorithm that is robust and effective and therefore commonly used in DL [125]. For more details on the Adam algorithm, the reader is referred to Kingma et al. [56].

4.2.5 Specify Performance Measures to be estimated

Performance measures describe different numerical quantities which are used to assess the performance of a method [80]. It is important to mention that for simulation studies, being empirical experiments, the performance measures themselves are only estimated and therefore subject to errors [80]. This issue needs to be considered when evaluating the results of a simulation study on the one hand by having an estimate of uncertainty and on the other hand by thinking about the number of repetitions of the study n_{rep} [80]. The simulation sample size n_{rep} is specified in section 4.2.6.

The determination of the required performance measures results from the objectives (see subsection 4.2.1.) as well as from the defined estimands and targets (see subsection 4.2.3.). When the focus of a study is an estimand, a widely used performance measure is the bias [80]. The bias hereby describes the amount by which an estimand $\hat{\theta}_0$ deviates from the true value θ_0 on average. Another common measure is the coverage of a certain Confidence Interval (CI) [80]. In reviewing several different simulation studies, Morris et al. [80] ascertained that bias and coverage are by far the most used performance measures.

In the presented simulation study several performance measures are considered and displayed in order to support the objectives stated in subsection 4.2.1. Table 5 outlines these different measures and their design for the two causal models PLR and IRM. The design only differs in the construction of the measure but the same informative value is given for both causal models. Furthermore, a brief description of the performance measure is presented in Table 5. For the sake of clarity, the equations in Table 5 are representing the calculation of the performance measure for one simulation run unless explicitly stated by referring to the number of repetitions n_{rep}. The resulting values for one simulation run need to be averaged over all simulation runs in order to obtain a final performance measure. This representation is chosen to simplify the formulas. The equations in Table 5 which are already referring to all simulation runs represent the final performance measures as the different measures can not be broken down into individual simulation runs. The main difference between the performance measure equations for

PLR models and IRMs is that the true value of the causal parameter may vary from one simulation run to another simulation run. Therefore, $\theta_{0,j}$ describes the true underlying causal parameter, or ATE, for an individual simulation run j whilst θ_0 describes the homogeneous treatment effect which is constant for all simulation runs.

Table 5: Performance Measures
Source: own table.

Performance Measure	**Homogeneous Treatment Effect (PLR)**	**Heterogeneous Treatment Effects (IRM)**	**Description**
Absolute Bias	$\hat{\theta}_{0,j} - \theta_0$	$\hat{\theta}_{0,j} - \theta_{0,j}$	Shows the absolute deviation of an estimate from the true value, which can give an indication of the performance of the methods evaluated.
Standardized Bias	$\frac{\hat{\theta}_{0,j} - \theta_0}{SE(\hat{\theta}_{0,j})}$	$\frac{\hat{\theta}_{0,j} - \theta_{0,j}}{SE(\hat{\theta}_{0,j})}$	Can be more informative compared to the absolute bias, as the consequence of the bias depends on the size of the uncertainty in the parameter estimate [15].
Relative Bias	$\frac{\hat{\theta}_{0,j} - \theta_0}{\theta_0}$	$\frac{\hat{\theta}_{0,j} - \theta_{0,j}}{\theta_{0,j}}$	When different values of θ_0 are used for different DGPs, the relative bias allows easier comparison between values (only applicable if $\theta_0 \neq 0$ and $\theta_{0,j} \neq 0$) [80].
Average Standard Error (SE)	$\frac{1}{n_{rep}} \sum_{j=1}^{n_{rep}} SE(\hat{\theta}_{0,j})$	$\frac{1}{n_{rep}} \sum_{j=1}^{n_{rep}} SE(\hat{\theta}_{0,j})$	Averages the precision/ efficiency of the estimator for one simulation run/ one dataset over all simulation runs and therefore indicates how much the estimator varies in average.

Performance Measure	**Homogeneous Treatment Effect (PLR)**	**Heterogeneous Treatment Effects (IRM)**	**Description**
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{n_{rep}} \sum_{j=1}^{n_{rep}} (\hat{\theta}_{0,j} - \theta_0)^2}$	$\sqrt{\frac{1}{n_{rep}} \sum_{j=1}^{n_{rep}} (\hat{\theta}_{0,j} - \theta_{0,j})^2}$	Measures the accuracy of the estimator whilst taking the true value θ_0 into account, integrates variance and bias measures into one summary performance measure but is more sensitive to the choice of n_{obs} [80]. The square root is taken to transform the RMSE onto the same scale as the SE.
Empirical Deviation (ED)	$\sqrt{\frac{1}{n_{rep}-1} \sum_{j=1}^{n_{rep}} (\hat{\theta}_{0,j} - \overline{\hat{\theta}_0})^2}$	$\sqrt{\frac{1}{n_{rep}-1} \sum_{j=1}^{n_{rep}} (\hat{\theta}_{0,j} - \overline{\hat{\theta}_0})^2}$	Measures the precision/ efficiency of the estimator, not considering the true value, and indicates the long-run standard deviation of $\hat{\theta}_0$ over the n_{rep} repetitions [80]. Indicates how much the different estimators scatter around the mean estimator (only applicable if $n_{rep} > 1$)
Coverage of CI	Proportion of times the $1 - \alpha$ CIs CI_j contains the true value θ_0	Proportion of times the $1 - \alpha$ CIs CI_j contains the true value $\theta_{0,j}$	Measures a "key property for the long-run frequentist behaviour of an estimator" [80, p. 2087]. In the following, $\alpha = 5\%$ is used as it is common in research.

4.2.6 Specify Number of Simulation Repetitions

In order to estimate performance measures to an acceptable level of precision, it is necessary to run the simulation experiment several times [80]. The number of repetitions or synonymously the simulation sample size n_{rep} hereby describes the number of independent repeat runs of each factor combination [77]. In other words, the same simulation experiment is repeatedly executed. By this replication, it is on the one hand possible to obtain an estimate of the experimental error to determine whether observed differences are also statistically different [77]. On the other hand, replication allows for more precise estimands because sample means can be used [77]. Many published simulation studies report the average estimand over the n_{rep} simulations performed as a measure for the true parameter of interest [15]. The central issue in choosing n_{rep} properly is handling the uncertainty. Theoretically, it is possible to calculate a needed n_{rep} when expected values for a performance measure and the required Monte Carlo SE as an estimate of uncertainty are available [80]. Therefore, one convenient advantage of simulation studies is that the Monte Carlo SE can be assessed during the simulation experiment and n_{rep} can be increased very easily when needed [80]. This implies that the number of repetitions n_{rep} that is initially and fixed set in this study can be easily adjusted if needed. Nevertheless, it is important to mention that a larger n_{rep} implies also a larger computational time [80].

In reviewing several studies, Morris et al. [80] ascertained that most simulation studies use $n_{rep} = 1000$ as a number of repetitions. As already referenced, the simulation study should partly follow the study of Farell et al. [26] in some design parameters. Here the authors use 1000 replications for the simulation study. As well, Whata et al. [118] use 1000 replications for each scenario.

For the exploratory phase, the simulation sample size is set to $n_{rep} = 50$ in order to explore the various DGPs presented in subsection 4.2.2 in a simple way. By limiting the number of simulation runs to a reduced number, the computing effort seems to be manageable. Nevertheless, based on some pre-tests, a simulation sample size of $n_{rep} = 50$ appears to be suitable to accommodate for outliers and to derive a clear understanding of the method's behaviour for the respective DGP. For the detail phase of the simulation study a simulation sample size of $n_{rep} = 1000$ is used according to pertinent literature. It is worth mentioning that the detail phase of the study may require extensive computation time, and therefore additional computational resources may be necessary. The computational demand and the resulting implications are discussed in subsection 4.4.4.

4.2.7 Overview Planning Steps

Appendix B.1 provides an overview of the key steps involved in planning the simulation study for this thesis. This includes a description of each step, where to find more details about it in the thesis, and a summary of how the steps are taken into account in the study design.

4.3 Design of the Simulation Study

After outlining the planning of the simulation study (see section 4.2) the next step is about the construction and verification of a computer program to perform the simulation experiment [66]. As stated in section 4.1, it is intended to design separate scripts for the simulation study. In the following section, the code, the associated classes, and the functions shall be briefly described. Further details can be found in the commented library, which is available in Appendix C.9 and in the linked GitHub repository. During the development of the code for the simulation study, recurring plausibility and usage checks have been performed. As for the implementation of the DL approaches in section 3, the software framework for designing the simulation study is Python.

In order to implement the aforementioned requirements in a structured manner, the Python library `DMLSim` is created (see Appendix C.9). In the style of Morris et al. [80] and to have a straightforward and intuitive approach to minimize errors, the development of `DMLSim` and the according packages has been started small and specific for one causal model, one ML method, one DGP, and for one repetition. In the further course, the library has been built up and generalised including several built-in checks, checking that the behaviour is as expected [80]. After being satisfied that the simulation study behaves sensibly also for larger runs, it is checked for datasets generated with different seeds. Afterwards, running a few repetitions is used to check if the bias for a specific DGP is as expected. This procedure is repeated after implementing DL methods and the various considered DGPs.

`DMLSim` consists of the package `dml_nn`, which contains the developed API described in section 3 as well as a network builder that generates structured dictionaries for PLR models and IRMs with PyTorch neural networks. In addition, `DMLSim` includes the package `dml_sim`, which contains all necessary components for the execution of the planned simulation study. The complete architecture of `DMLSim` library is shown in Figure 15.

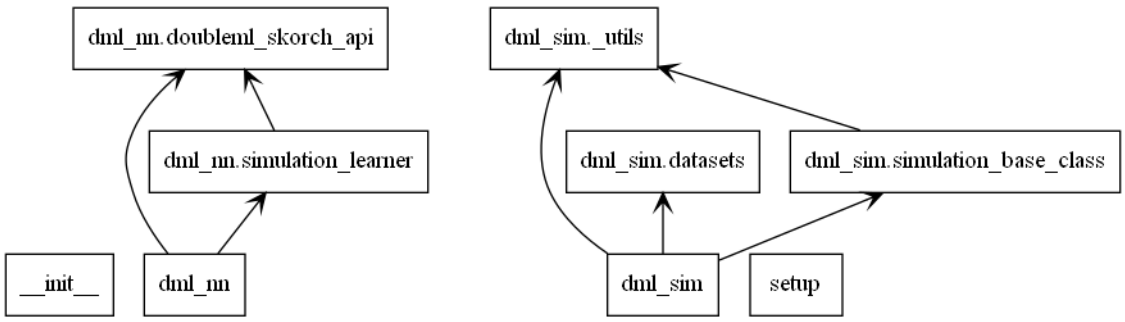


Figure 15: Packages and Modules in DMLSim

Source: own figure.

To apply neural networks in the simulation study, an instance of the `class network_builder` is used which returns the corresponding learner packages as dictionaries. The `class network_builder` class structure is shown in Figure 16. The

main object for the simulation study is the `class simulation_study` with a respective class structure as presented in Figure 17.

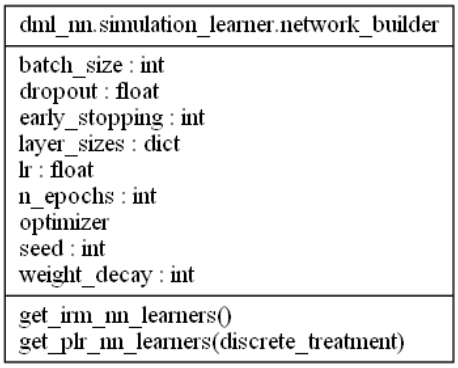


Figure 16: Class network_builder

Source: own figure.

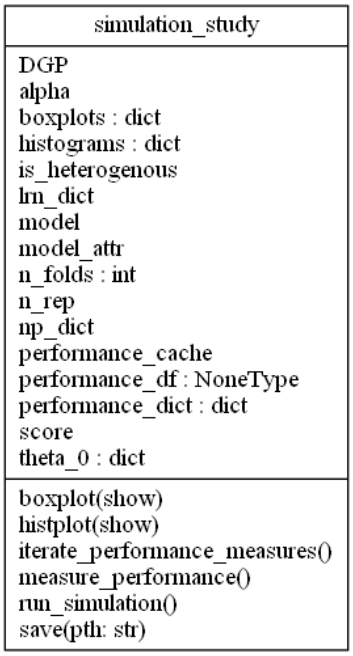


Figure 17: Class simulation_study

Source: own figure.

With the `DMLSim` library, the different scenarios of the simulation study can be performed in a generalized way as an instance of the `class simulation_study`. An example is shown in the code below:

```
# Make imports
from sklearn.base import clone
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LogisticRegressionCV

# import simulation_study object
from dml_sim.simulation_base_class import simulation_study as ssession
# import network builder
from dml_nn.simulation_learner import network_builder

from doubleml.datasets import make_plr_CCDDHNR2018
from doubleml import DoubleMLPLR

# Setup scenario
scenario_DGP = make_plr_CCDDHNR2018
folder_name = 'example_scenario'
```

```
path = r"../"  # local or cloud
scenario_model = DoubleMLPLR

#################### Simulation Settings ####################

builder = network_builder()  # Init builder with default settings

# Define general ML learner
lasso_reg = LassoCV()
lasso_cls = LogisticRegressionCV(penalty="l1", solver="liblinear")
rf_reg = RandomForestRegressor()
rf_cls = RandomForestClassifier()


# Create a structured learner dict for machine learning algorithms
learner_dict_plr_ml = {
    "lasso": {"ml_l": clone(lasso_reg), "ml_m": clone(lasso_reg)},
    "random_forests": {"ml_l": clone(rf_reg), "ml_m": clone(rf_reg)},
}

# Create a structured learner dict for neural network algorithms
learner_dict_plr_nn = builder.get_plr_nn_learners()

# Combine dicts
learner_dict_plr = {**learner_dict_plr_ml, **learner_dict_plr_nn}

# Create a dict with n_obs and dim_x for the DGP data setting
np_dict = {"n_obs": [500], "dim_x": [20]}

# Init a dml simulation session
scenario = ssession(
    model=scenario_model,
    score="partialling out",
    DGP=scenario_DGP,
    n_rep=50,
    np_dict=np_dict,
    lrn_dict=learner_dict_plr,
    alpha=1,
    is_heterogenous=False,
)

# Perform full simulation
scenario.run_simulation()

# Generate boxplots
scenario.boxplot(show=False)
```

```
# Generate histograms
scenario.histplot(show=False)

# Measure performances
scenario.measure_performance()

# Save measures and plots to a folder
scenario.save(path + folder_name)
```

Some of the DGPs, which are considered for this thesis and described in subsection 4.2.2, are already implemented in the DML package. This applies to plr_CCDDHNR2018, irm_data, and plr_turell2018. The six Opossum DGPs as described in Table 2 are available after the installation of the Opossum package through the specification of the arguments. The remainder of the DGPs as described in subsection 4.2.2 are manually implemented in the `dml_sim.datasets` module. This applies to the DGPs irm_friedman, plr_fingerhut2018, and irm_farell2021. The class for the simulation study requires the DGP as a callable function with the following keyword arguments in the following form:

```
def DGP_Wrapper(alpha=None, n_obs=500, dim_x=20, **args):
    # Generating x, y, d, theta
    return x, y, d, theta # As numpy arrays
```

The factors to vary during the presented simulation study are n_{obs} and dim_x (see 4.2.2.). The factor levels refer to the considered data setting. The individual factor levels, as presented in subsection 4.2.2, can be added to the `np_dict` in an instance of the **class** `simulation_study`.

The methods to evaluate in the simulation study are specified in the `learner_dict`. As described above, neural networks are added to the `learner_dict` after being initialized in the **class** `network_builder`. The different model architectures (see Table 3) and the literature-based parameters for training the neural networks (see Table 4) are specified as arguments of the **class** `network_builder`. The ML methods Lasso and Random Forest are defined in an instance of the **class** `simulation_study` and are appended to the `learner_dict`.

The performance measures, as described in subsection 4.2.5, are implemented as methods in the **class** `simulation_study` and are calculated according to the mathematical equations (see Table 5). For settings, where the treatment effect is fully heterogeneous, the true value of the ATE is calculated for each simulation run. Special mention should be made to the Opossum DGPs since, according to the data structure that is reported by Winkel et al. [120], the true value for the ATE is supposed to be calculated only with respect to the treatment group. For further details, it is referred to the respective code (see **class** `simulation_base_class` in Appendix C.9).

The number of simulation repetitions n_{rep} is specified when initializing a simula-

tion session for a specific scenario. For this initialization, a causal model (DML implementation for PLR or IRM), a suitable score, a DGP, a true value of the causal parameter, and a boolean indicator to state whether the treatment effect is heterogeneous are necessary arguments. Additionally, the various methods, as `learner_dict`, and the different data settings, as `np_dict`, are handed over.

When designing a simulation study it is also important to carefully consider which data and results shall be stored [15]. The specification of which data is to be saved is implemented as a method in the **`class`** `simulation_study`. To summarize, all attributes, performance measures, and histograms as well as the boxplots of the standardized bias for each method and data setting are stored.

It is worth mentioning that the seed is set several times in the **`class`** `simulation_study` in order to meet the requirement of reproducibility (see section 4.1). This is crucial to ensure that the generated datasets, and consequently, the results, can be replicated [15]. In order to make the results of the neural network models reproducible, it shall be highlighted that in addition to the `numpy` seed, the `torch` seed must also be set during an individual simulation runs.

For further descriptions, especially about class arguments, method parameters, required data types, and return values it is referred to the commented code, which is available in the respective packages and modules of the `DMLSim` library (see Appendix C.9).

As stated during the planning steps (see section 4.2) the simulation study in this thesis is built up in a flexible way. Since the different scenarios can be performed as instances of the **`class`** `simulation_study`, various different scenarios can be generated and evaluated in a straightforward way. The presented design can be easily extended with regard to new DGPs, other ML learner or neural network architectures, and various combinations of factor levels. For example, other DGPs can be added to the `dml_sim.datasets` module and be chosen as an argument for an instance of **`class`** `simulation_study`. Other ML learners can be added to the respective `learner_dict` just in the same way as shown for Lasso and Random Forest above. Additional network architectures can be added to **`class`** `network_builder`, including changing the neural network training parameters or adding more regularization methods. Various data settings can be flexibly implemented by defining the factor levels in the `np_dict`.

Furthermore, as stated in subsection 4.2.4, the simulation study can also be extended to incorporate the causal models PLIV and IIVM which are also already implemented in the DML package. To extend the present simulation study for models using IVs there are some minor changes in **`class`** `simulation_study` expected, for example when it comes to the implemented case distinctions. Since the simulation study is designed in an object-oriented, structured, and generalized way, these changes should be easily possible. The implementation of PLIV and IIVM settings in the simulation study design might be one interesting topic for future works in order to investigate the use of DL approaches also in scenarios with IVs.

4.4 Execution of the Simulation Study

Once the simulation study has been planned (see section 4.2.) and designed (see section 4.3.), the subsequent step involves executing the simulation study as a simulation experiment [66]. Thereby the aim is to detect and understand any changes in the observed behaviour of the methods which are caused by the variation of the factors [66].

The process of executing the simulation study is shown in the process diagram in Figure 18 which expands Figure 14.

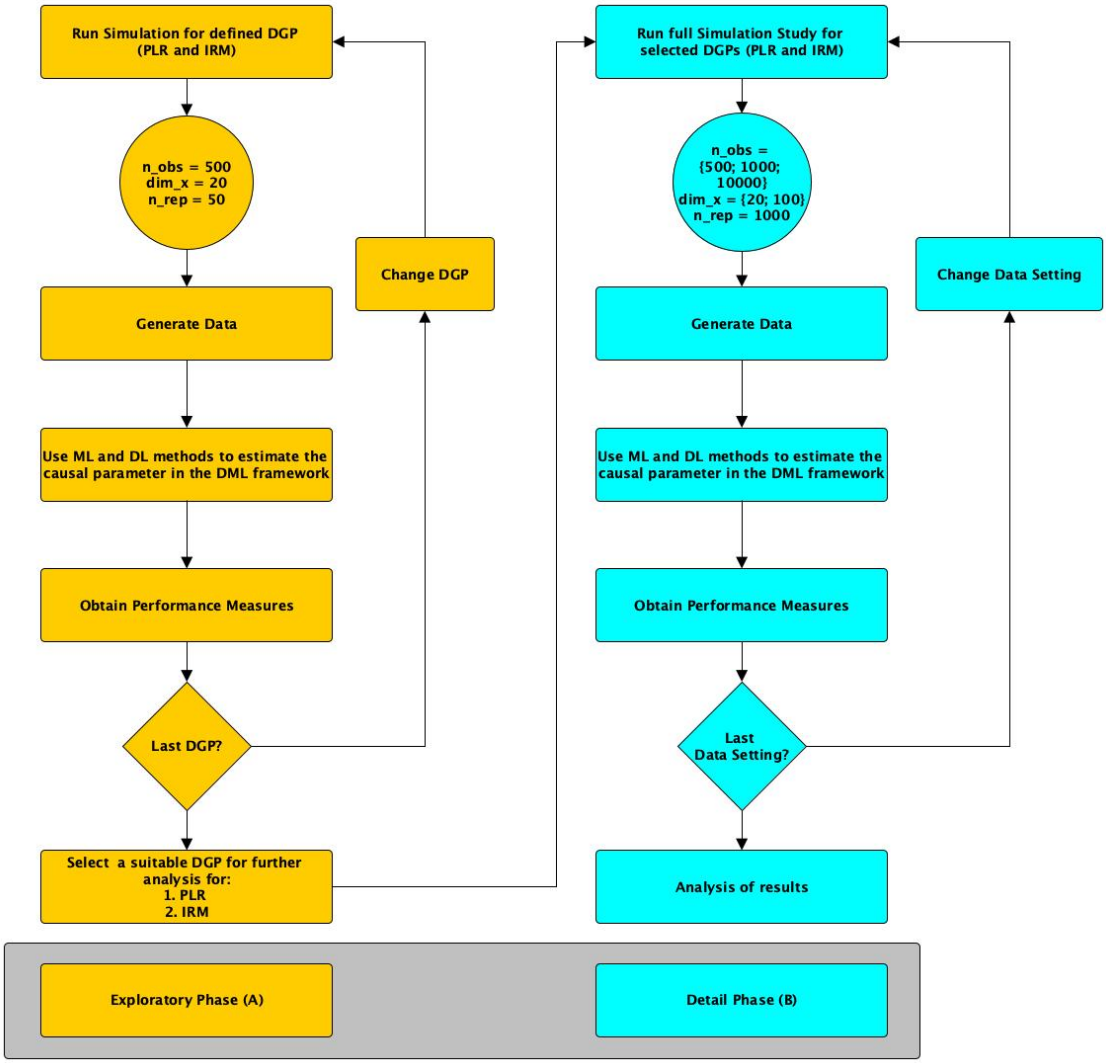


Figure 18: Process Diagram Simulation Study

Source: own figure.

Following the process diagram in Figure 18, the simulation study results in various different scenarios and settings to be considered. It should be noted that given the numerous factors that can be potentially varied, there exists an extensive range of

scenarios that could be simulated. The potential factors may also include the default parameters of DGPs or the design of techniques to deal with the generalization problems of DL approaches (see section 2.4). Due to the limited scope of this thesis, it is not possible to investigate all possible scenarios. As described in Figure 18, the exploratory phase is used to identify suitable DGPs for further analysis in order to not run the full simulation study for DGPs which prove to not be particularly suitable for the analysis with DL approaches. The exploratory phase is executed for a base case data setting and a limited number of simulation runs (see subsections 4.2.2 and 4.2.6). Besides, various parameters and design factors, which are not part of the core factors to vary as described in subsection 4.2.2, are set based on justified assumptions or are set to default values. The simulation study is implemented in a flexible way which makes an adaption of the methods, settings, parameters, and factors easy.

After running the limited simulation in the exploratory phase for all DGPs, a suitable DGP to focus on for the detail phase is chosen for both a PLR model and an IRM. As mentioned in subsection 4.2.2, the exploratory phase aims at finding a DGP for which the DL methods show good and promising performance compared to the conventional ML methods Lasso and Random Forest. The selection of DGPs to be used in the detail phase of the simulation study is based on the presented performance measures.

The detail phase is performed for all data settings described by the factor variation (see subsection 4.2.2) and for both a PLR model and an IRM with the DGP chosen for each case. In addition, the detail phase is extended in terms of the number of simulation runs (see subsection 4.2.6).

It is already worth mentioning that the detail phase is performed using High Performance Computing (HPC) resources due to the high computational demand, while the exploratory phase can be performed on local computational resources due to the limited scope. Further details on the computational demand and its implications are presented in subsection 4.4.4.

For the sake of clarity and reference, the scenarios considered are numbered and described in the following subsections. Additionally, the scenarios are explained in greater detail.

4.4.1 Exploratory Phase

Table 6 lists the twelve different scenarios for the exploratory phase of the simulation study resulting from the use of the various DGPs (see subsection 4.2.2). It is also stated to which causal model the scenario is referring.

Table 6: Scenarios in Exploratory Phase of Simulation Study

Source: own table.

Scenario	DGP	Causal Model
A01	plr_CCDDHNR2018	PLR
A02	irm_data	IRM
A03	plr_turell2018	PLR
A04	irm_friedman	IRM
A05	plr_fingerhut2018	PLR
A06	irm_farell2021	IRM
A07	opossum_1	PLR
A08	opossum_2	PLR
A09	opossum_3	IRM
A10	opossum_4	IRM
A11	opossum_5	IRM
A12	opossum_6	IRM

As stated in subsection 4.2.2 and visualized in Figure 18, the scenarios in Table 6 shall be executed for a base case setting of the factor levels. This base case is defined with $n_{obs} = 500$, $dim_x = 20$, and $n_{rep} = 50$.

The computational demand in the exploratory phase of the present simulation study is manageable due to the limited number of simulation runs and the small dataset resulting from the base case factor settings. Therefore, it is sufficient to use only limited local computing resources. As described in section 4.3, each scenario presented in Table 6 is executed as an instance of the developed `class simulation_study`. The resulting performance measures, the histograms, the boxplots, and a log file are saved accordingly. The Python source code for executing all scenarios in the exploratory phase of the simulation study is available in Appendix C.2.

4.4.2 Analysis of the Exploratory Phase and Selection of a Data Generation Process for the Detail Phase

The results of the exploratory phase are presented on the basis of the performance measures relative bias and coverage of the 95 % CI. All other performance measures can be retrieved from the log files in Appendix D.1 and the histograms, as well as the boxplots for each scenario, are presented in Appendix A.4 and A.5. The notebook summarizing and presenting the resulting performance measures is available in Appendix C.2.

The resulting relative bias for all twelve scenarios of the exploratory phase is shown in Table 7 for all PLR scenarios and in Table 8 for all IRM scenarios. The results are colour-coded, with red numbers indicating a negative relative bias and blue numbers a positive relative bias. The darker the field is filled with a certain colour, the greater the relative bias in a certain direction and therefore the greater the deviation from the true value. Additionally, Figure 19 shows the coverage for all PLR scenarios and Figure 21 shows the coverage for all IRM scenarios.

For reasons of space, the neural networks models are abbreviated as nn_0 to nn_9 and Random Forest as rf in certain tables.

PLR Scenarios

Table 7 shows that the standard ML method Lasso with default parameters proves a strong performance for most of the PLR scenarios resulting in approximately unbiased estimates of the treatment effect. An exception to this is scenario A05 where a significant bias appears. The Random Forest model shows also good performance. However, in all PLR scenarios except scenario A05 the estimates deviate more from the ground truth than for Lasso. The simple neural networks with only one hidden layer (models 0, 1, and 2) show some heavy bias in several scenarios. Especially model 0 with one neuron in the hidden layer has a poor performance with large deviations for all scenarios except scenario A03. Model 1 outperforms the standard ML methods Lasso and Random Forest in scenario A03 with regard to relative bias and shows fairly good performance in scenarios A01 and A07. Similar behaviour can be seen for neural network models with more than one hidden layer. Especially models 4 and 9 provide approximately unbiased estimations for scenario A03. It is noticeable that for scenario A03 most of the neural network models show stronger performance in providing unbiased estimates than standard ML techniques. The comparatively good performance of model 6 in scenario A05 is also striking. Whilst other learners show at least a two-digit relative bias, model 6 provides fairly good estimates. Nevertheless, it is to be noted that the standard ML method Lasso outperforms most of the DL methods for most of the scenarios. Especially for scenario A08, none of the neural network models can come close to the performance of the Lasso method. Compared to Random Forest, nearly all neural network models show similar or better performance for all scenarios.

Table 7: Relative Bias for Partial Linear Regression Scenarios in Exploratory Phase in %

Source: own table.

	A01	A03	A05	A07	A08
lasso	-0.49 %	-2.8 %	36.09 %	-1.64 %	-2.64 %
random_forests	-2.09 %	-3.43 %	23.87 %	5.77 %	-4.02 %
neural_net_model0	19.79 %	5.34 %	93.85 %	-33.21 %	-85.57 %
neural_net_model1	15.34 %	-2.24 %	97.08 %	16.29 %	-17.61 %
neural_net_model2	2.2 %	-3.03 %	108.91 %	3.53 %	-17.93 %
neural_net_model3	5.44 %	-2.12 %	87.55 %	10.12 %	-15.09 %
neural_net_model4	6.28 %	-0.78 %	87.06 %	15.52 %	-11.4 %
neural_net_model5	-0.52 %	-2.35 %	28.81 %	3.93 %	-12.94 %
neural_net_model6	-4.28 %	-3.22 %	7.28 %	-5.68 %	-11.33 %
neural_net_model7	-1.46 %	-2.85 %	17.51 %	-2.13 %	-13.34 %
neural_net_model8	-5.47 %	-2.66 %	74.16 %	7.49 %	-15.17 %
neural_net_model9	2.81 %	-1.45 %	52.93 %	6.94 %	-16.45 %

One thing that is striking in Figure 19, is the poor performance of nearly all learners in scenario A05 with regards to the coverage of the 95 % CI. Only for model 6, the 95 % CI contains the true value of the estimand in more than 50 % of the simulation runs. For

Lasso and the simple neural network models the 95 % CI does not contain the true value for a single simulation run. This is also a representation of the large relative bias for most of the models for scenario A05. Besides that, the simple neural network model 0 is not competitive to any other learner for all of the PLR scenarios with regard to the coverage. Similar to the description of the resulting relative bias, the good performance of nearly all neural network models for scenario A03 can be seen in Figure 19. With the exceptions of model 0 and model 2, all DL methods show a better coverage compared to the standard ML methods Lasso and Random Forest. For scenarios A01 and A07, especially the simple model 2 with only one hidden layer and model 5 can keep up with Lasso. For scenario A08 all neural network models show a significantly worse performance which is a similar observation as for the relative bias in Table 7.

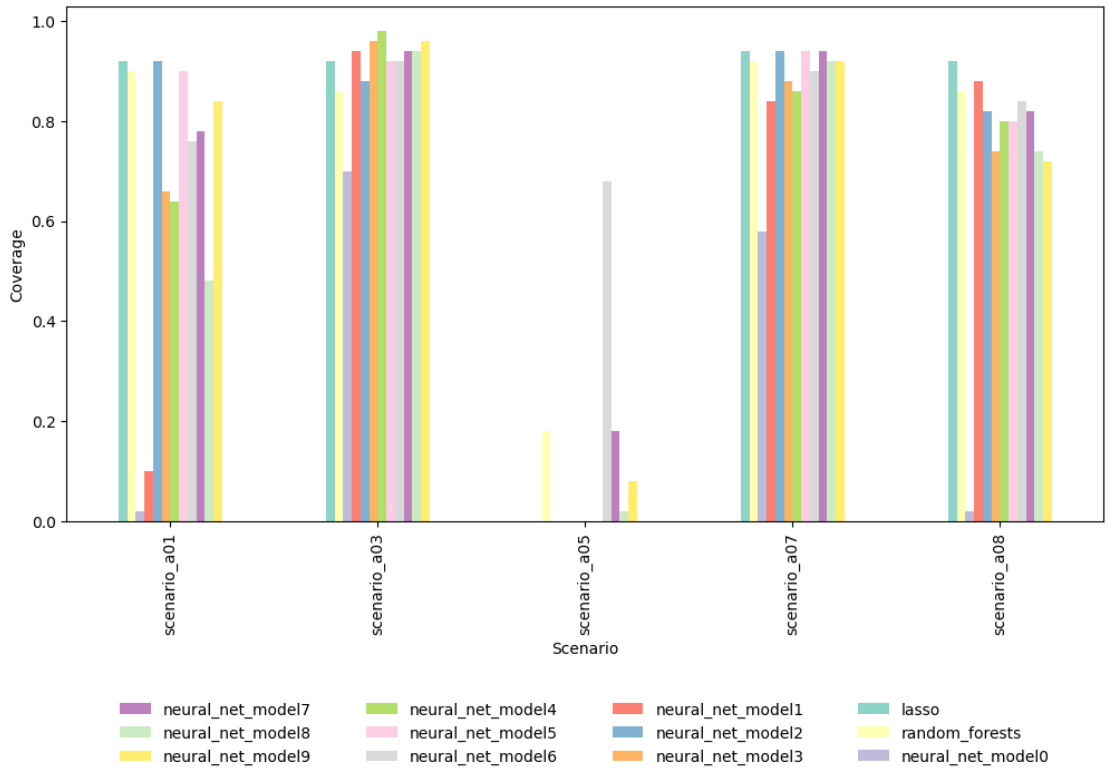


Figure 19: Coverage of 95 % Confidence Interval for Partial Linear Regression Scenarios in Exploratory Phase in %

Source: own figure.

For the RMSE (see Appendix B.2) a similar behaviour can be observed. Model 0 is not competitive with the other learners, as it has the worst accuracy for scenarios A01, A03, A07, and A08. For scenario A05 all neural network models show a high RMSE resulting in a bad accuracy except for model 6. The good performance of model 6 for scenario A05 could also be seen in Table 7 and Figure 19. For scenario A03, almost all neural network models show better accuracy than Random Forest and equal or better accuracy than Lasso. For scenarios A01, A07, and A08 the neural network models show in general worse RMSEs compared to Lasso and Random Forest. At least neural network models

5 and 7 show better performance compared to Lasso. It is noticeable that the accuracy of the Random Forest is slightly better compared to Lasso for scenarios A07 and A08. Another thing that is striking, is the comparatively high average SE for scenarios A07 and A08 (see Appendix B.3). As they are higher compared to the other PLR scenarios for each learner and approximately equal across all learners, this observation may be attributed to the DGP. It should also be emphasized that there is no strong evidence for overfitting, as the RMSE is comparable for network architectures with different complexity but similar bias for a specific scenario.

The decision for a DGP to investigate further during the detail phase of the study is mainly based on the performance measures relative bias, RMSE, and coverage of CI among the other ones. In accordance with the objectives of the simulation study, it is looked for a DGP where DL methods show promising performance especially compared to standard ML methods. For this thesis, the DGP plr_turell2018 from scenario A03 is chosen for the PLR setting of the detail phase. As seen in Table 7 and visualized in Figure 20, the neural network models with the exception of model 0 show similar or better performance in providing unbiased estimates compared to the standard ML methods Lasso and Random Forest. Also with regards to the coverage and the RMSE, this DGP seems to be interesting for further investigations in different data settings. The histograms showing the standardized bias for scenario A03 are presented in Appendix A.2.

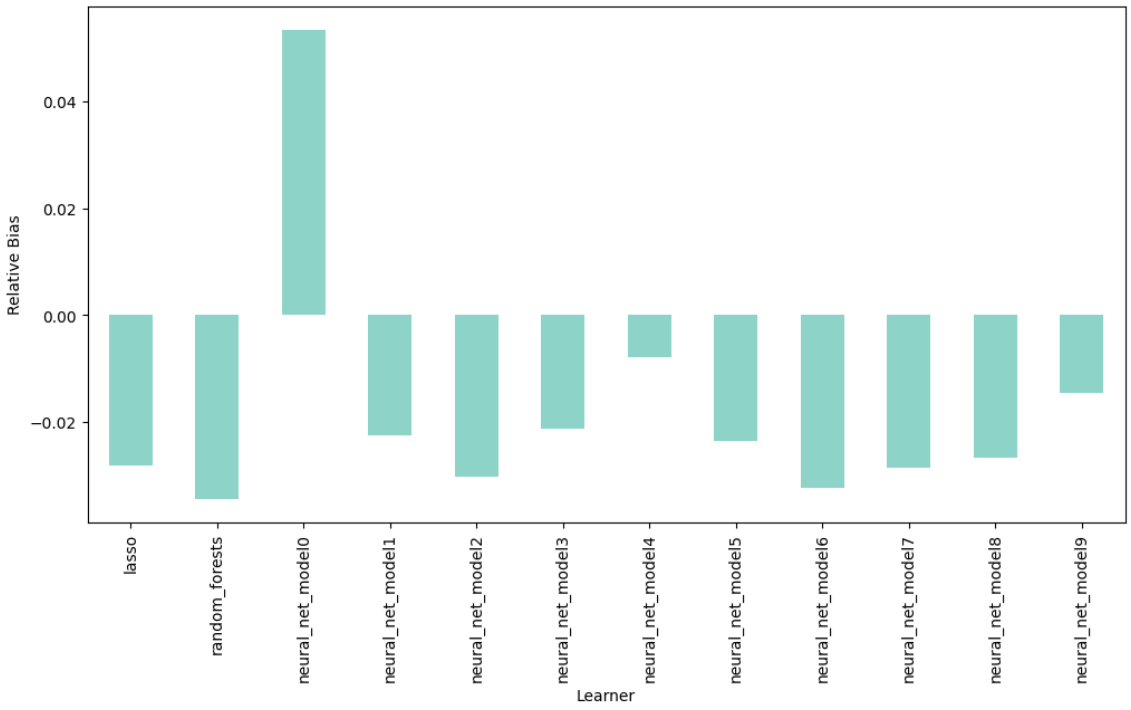


Figure 20: Relative Bias for Scenario A03 in %

Source: own figure.

Of course, other scenarios and DGPs are also interesting to be further explored, but due to

the limited scope of this work, these remain a possible direction for future work based on this thesis. Especially the DGP plr_fingerhut2018, which has been used in scenario A05, seems to be interesting for further research, as only the wide network model 6 with many neurons but a limited size of hidden layers shows a moderate performance. Particularly because larger and more complex networks show worse performance in scenario A05 compared to model 6, with regards to coverage or RMSE, this might be a starting point for some research. The research could focus on how to choose the optimal network architecture for a specific causal inference setting.

IRM Scenarios

Table 8 shows that the standard ML method Lasso with default parameters proves a strong performance for most of the IRM scenarios resulting in approximately unbiased estimates of the treatment effect. An exception to this is scenario A09 where a significant bias appears. The Random Forest model shows good performance in scenarios A04, A06, and A10 but in each case, the relative bias is larger compared to Lasso. For scenarios A09 and A12 a significant bias appears. The simplest neural network, model 0, shows a bad performance resulting in a large relative bias for all scenarios. The low complexity models 1, 2, 3, and 4 show exceptionally bad performance in scenario A04 resulting in a heavy relative bias. With exception to scenario A09, these neural network models have a significantly larger relative bias compared to the standard ML methods Lasso and Random Forest. In scenario A09 models 1, 2, 3, and 4 provide less biased estimates than Lasso and Random Forest. Furthermore, model 4 shows a good performance in scenario A11 where it outperforms Random Forest. Similar behaviour can be observed for more complex neural network models (models 5 - 9). They show comparatively good performance in scenario A09. Besides, models 6 and 7 have the best performance among all considered learners in scenario A11. It is noticeable that for scenarios A02, A04, A06, A11, and A12 all neural network models show worse performance with regard to the relative bias compared to Lasso and Random Forest.

Table 8: Relative Bias for Interactive Regression Model Scenarios in Exploratory Phase in %

Source: own table.

	A02	A04	A06	A09	A10	A11	A12
lasso	2.78 %	-0.89 %	-0.27 %	0.62 %	-0.22 %	-1.34 %	-1.88 %
rf	6.22 %	1.45 %	-0.93 %	-0.89 %	-15.89 %	1.58 %	-21.86 %
nn_0	19.4 %	-111.12 %	-178.18 %	-64.09 %	-101.03 %	-86.94 %	-124.59 %
nn_1	34.38 %	396.04 %	-26.43 %	-11.05 %	-44.68 %	-17.12 %	-54.91 %
nn_2	30.95 %	367.11 %	-13.55 %	-6.14 %	-36.66 %	-9.73 %	-43.5 %
nn_3	34.01 %	425.45 %	-18.21 %	-5.52 %	-40.28 %	-11.13 %	-49.55 %
nn_4	36.18 %	549.85 %	-7.16 %	3.07 %	-33.02 %	1.43 %	-41.6 %
nn_5	24.16 %	145.05 %	-7.4 %	-2.2 %	-29.91 %	-1.81 %	-34.21 %
nn_6	16.57 %	63.21 %	-2.7 %	0.3 %	-19.82 %	0.52 %	-19.13 %
nn_7	20.59 %	23.22 %	-5.66 %	-0.19 %	-25.04 %	-1.26 %	-26.19 %
nn_8	31.61 %	-53.67 %	-8.37 %	-0.59 %	-32.61 %	-4.94 %	-39.95 %
nn_9	26.29 %	2.7 %	-5.64 %	-1.51 %	-28.63 %	-2.93 %	-33.54 %

The coverage of the 95 % CI presented in Figure 21 shows a similar behaviour as described

above. With the exception of scenario A11, where model 8 has the highest coverage, Lasso performs among the best for each scenario. It is noticeable that only for scenarios A09 and A11 the neural network models can keep up with the performance of Lasso. In the other scenarios, especially the simple neural network models lack behind in terms of coverage. In scenarios A02 and A04, the neural network model 0 outperforms the other neural network models with regard to coverage. It is apparent that the coverage varies quite heavily not only between the different learners but also between the different scenarios and therefore between the considered DGPs. Especially in scenario A10 only a few learners show a performance where the 95 % CI contains the true value of the estimand in more than 50 % of the simulation runs.

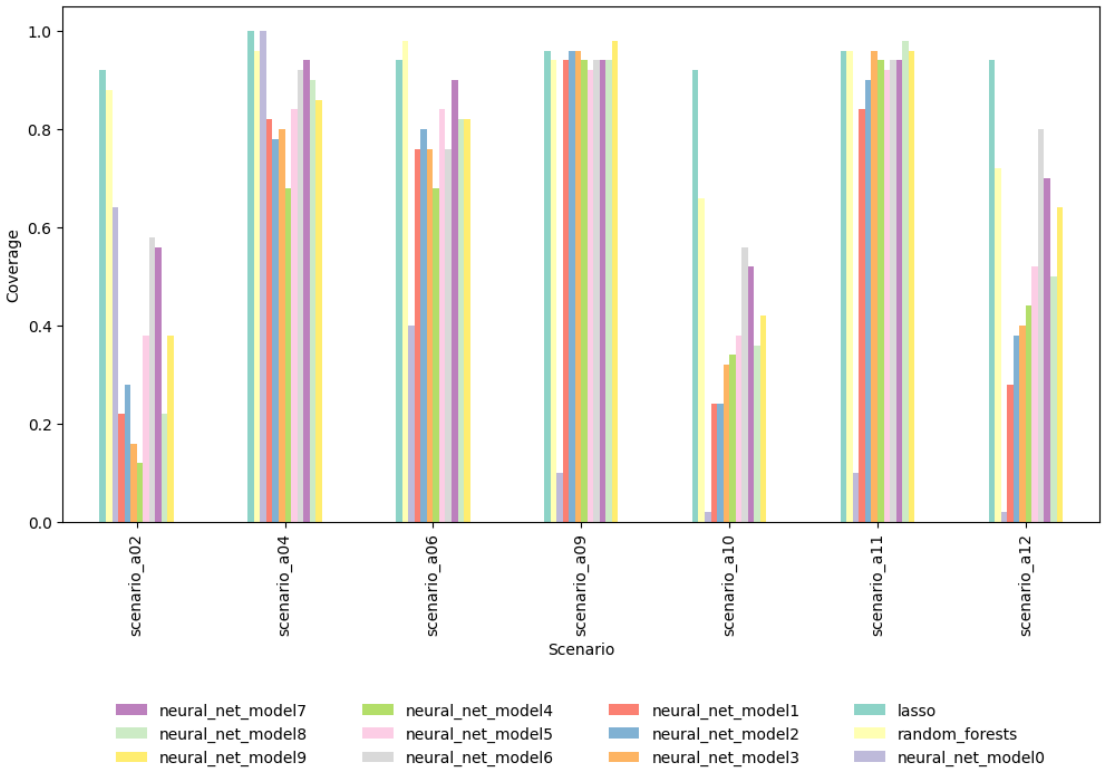


Figure 21: Coverage of 95 % Confidence Interval for Interactive Regression Model Scenarios in Exploratory Phase in %

Source: own figure.

In terms of RMSE (see Appendix B.4), one thing that is striking, is the enormous RMSE for model 0 in scenario A06. This is also reflected in a large value for the ED (see Appendix B.5) indicating that the estimations from model 0 are not precise or efficient. Besides this outlier, model 0 shows also the largest RMSE for scenarios A10, A11, and A12. In scenario A12, model 6 shows the lowest RMSE, in scenario A11 model 8 has the lowest RMSE and in scenario A09 model 4. In all three cases, the RMSE is significantly smaller compared to Lasso. Especially in scenario A09, models 1, 2, 3, 4, and 9 provide highly accurate estimates. For all other IRM scenarios either Lasso or Random Forest show the lowest RMSE. A remarkable finding is that the average SE and ED (see Appendix B.5 and

B.6) for the simple neural network models 0 - 4 are two to three times higher than those of the more complex models 5 - 9 in scenarios A04 and A06.

When choosing a suitable DGP to investigate further in the IRM setting, the decision shall mainly be based on the performance measures relative bias, RMSE, and coverage of CI among the other ones. For this thesis, the DGP opossum_3 from scenario A09 is chosen for the detail phase. As seen in Table 8 and visualized in Figure 22, the neural network models 6, 7, and 8 show much better performance in providing unbiased estimates compared to the standard ML methods Lasso and Random Forest. With regards to the coverage, the DL methods show equal performance compared to Lasso and Random Forest. Especially because of the highly accurate estimations of the neural network models, resulting in a very low RMSE, this DGP seems to be interesting for further investigations in different data settings. The histograms showing the standardized bias for scenario A09 are presented in Appendix A.3.

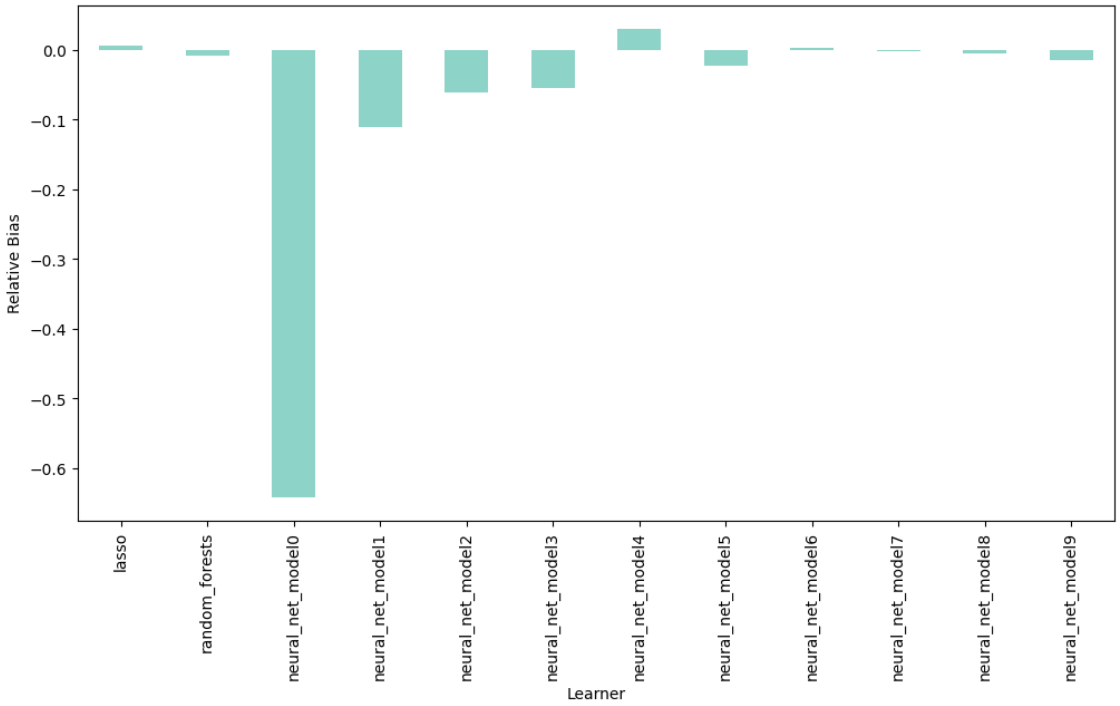


Figure 22: Relative Bias for Scenario A09 in %

Source: own figure.

Of course, other scenarios and DGPs are also interesting to be further explored, but due to the limited scope of this work, these remain a possible direction for future work based on this thesis. Especially the observation of the increased average SE and ED for the simple neural network models in scenarios A04 and A06 (see Appendix B.5 and B.6) are one interesting topic for further research. Besides, it is possible to conduct a more comprehensive analysis of the good performance of models 6 and 7 in scenario A11, where both models surpassed the two standard ML techniques.

Summary of Exploratory Phase

To summarize the exploratory phase of the simulation study it is referred back to the objectives of the simulation study (see subsection 4.2.1) and to the descriptions in subsection 4.2.2. Therefore, the exploratory phase is used to prove the successful and correct implementation of DL methods in the DML package for different settings as well as to find a setting where the use of DL methods looks promising in the context of CML. Besides, the newly implemented DL approaches are compared to ML methods for different DGPs.

First of all, it can be stated that the newly implemented DL approaches work for a great variety of different DGPs. Since no systematic deviations or other anomalies are apparent, it can be assumed that the implementation of the DL methods in the DML package is correct. Furthermore, the exploratory phase shows the flexibility of the developed software library `DMLSim` (see Appendix C.9) since various DGPs, data settings, and learners can be explored in a structured and straightforward way. In terms of comparison to formerly implemented ML methods, it can be seen that the DL methods lack behind in performance for most of the scenarios and network architectures. It is again to be noted that only one data setting with $n_{obs} = 500$ and $dim_x = 20$ for a limited number of simulation runs $n_{rep} = 50$ is considered in this exploratory phase. Despite this, certain neural network models demonstrate superiority in specific scenarios and DGPs in terms of providing unbiased estimates for causal parameters in PLR and IRM settings compared to the two ML methods Lasso and Random Forest. Therefore, two settings can be chosen where the application of DL methods seems promising and where it is interesting to gain further insight, for example, how the behaviour changes with different data settings.

4.4.3 Detail Phase

After selecting suitable DGPs for both, a PLR model and an IRM, the detail phase of the simulation study is executed for these DGPs. As stated, any changes in the observed behaviour due to varying the factors shall be detected. Varying the factors in this detail phase refers to subsection 4.2.2 and therefore to different data settings. As a result, there are twelve scenarios for the detail phase of the simulation study which are listed in Table 9.

Table 9: Scenarios in Detail Phase of Simulation Study

Source: own table.

Scenario	DGP	Causal Model	Data Setting
B01	plr_turell2018	PLR	$n_{obs} = 500$, $dim_x = 20$
B02	plr_turell2018	PLR	$n_{obs} = 1000$, $dim_x = 20$
B03	plr_turell2018	PLR	$n_{obs} = 10000$, $dim_x = 20$
B04	plr_turell2018	PLR	$n_{obs} = 500$, $dim_x = 100$
B05	plr_turell2018	PLR	$n_{obs} = 1000$, $dim_x = 100$
B06	plr_turell2018	PLR	$n_{obs} = 10000$, $dim_x = 100$
B07	opossum_3	IRM	$n_{obs} = 500$, $dim_x = 20$
B08	opossum_3	IRM	$n_{obs} = 1000$, $dim_x = 20$
B09	opossum_3	IRM	$n_{obs} = 10000$, $dim_x = 20$
B10	opossum_3	IRM	$n_{obs} = 500$, $dim_x = 100$
B11	opossum_3	IRM	$n_{obs} = 1000$, $dim_x = 100$
B12	opossum_3	IRM	$n_{obs} = 10000$, $dim_x = 100$

Since the computational demand for the detail phase of the simulation study is expected to be much larger compared to the exploratory phase, subsection 4.4.4 provides an analysis of the computational performance and implies consequences for the simulation design. The Python source code and the shell scripts for executing all scenarios in the detail phase of the simulation study are available in Appendix C.3.

4.4.4 Computational Demand and Performance

To estimate the computational time needed to run the simulation study, the number of fits of the DML model class n_{fits} is considered.

$$n_{fits} = n_r \cdot n_l \cdot n_s \tag{66}$$

In equation 66, n_r represents the number of repetitions, n_l the number of learners, and n_s the number of settings. The computing time can be estimated in a simplified way:

$$T_{sim} = T_{secondary} + \sum_{i=0}^{n_{fits}} t_i^{fit} \tag{67}$$

The time to perform the `fit` method from a DML model t_i^{fit} heavily depends on the learner `ml_m`, `ml_l`, `ml_g`, the data dimension n_{obs}, dim_x, and on the number of folds n_{folds}. When choosing the hyper-parameters of the learners, especially the number of iterations per epoch, the computing time must be considered accordingly. The computing time increases rapidly when combined with numerous repetitions. As described in subsection 4.2.6, a certain number of simulation runs n_{rep} must be achieved to reduce statistical deviations in the results. The following values are assumed for the detailed phase of the simulation study, resulting in the number of fits:

$$n_{fits} = n_{rep} \cdot n_{learners} \cdot n_{settings} = 1000 \cdot 12 \cdot 12 = 144000 \tag{68}$$

The time per run of the `fit` method for the selected learners and data settings is determined from a short pre-test to give an approximate estimate of $\tilde{t}^{fit}$.

$$\tilde{t}^{fit} = \frac{1}{n_l \cdot n_s \cdot n_r} \cdot \sum_{i=0}^{n_l} \sum_{j=0}^{n_s} \sum_{h=0}^{n_r} t_{i,j,h}^{fit} \tag{69}$$

The `tqdm` package outputs the averaged values across the learners and settings, which are shown in Table 10.

Table 10: Average Time when calling the fit() Method
(All values are given in seconds [1 s])

Source: own table.

	Data Setting				**Average**
Learner	500_20	10000_20	500_100	10000_100	
lasso	0.4608	1.45	36.54	16.97	13.8552
random_forests	2.92	101.41	10.07	484.80	149.8
neural_net_model0	6.37	40.06	6.65	33.68	21.69
neural_net_model1	5.03	35.88	3.68	19.77	16.09
neural_net_model2	3.65	32.34	1.67	17.19	13.7125
neural_net_model3	6.29	39.89	4.65	23.51	18.585
neural_net_model4	6.79	66.41	5.36	38.22	29.195
neural_net_model5	3.14	33.68	2.91	19.56	14.8225
neural_net_model6	2.18	18.25	2.01	15.15	9.3975
neural_net_model7	3.06	29.67	2.65	18.02	13.35
neural_net_model8	3.52	41.77	3.77	28.72	19.445
neural_net_model9	3.98	42.08	4.09	26.54	19.1725
Average	3.94923333	40.2408333	7.00416667	61.8441667	**28.2596**

Assuming that the calculations are performed sequentially, the resulting time requirement can be estimated with a roughly approximated time of about 28.2596 s per execution of the `fit` method and a conservatively estimated 10 minutes for ancillary activities such as initialization, data generation or similar:

$$T_{sim} \approx T_{secondary} + n_{fits} \cdot \tilde{t}^{fit} \tag{70}$$

$$T_{sim} \approx 120\,\text{min} + 144000 \cdot 28.2596\,\text{s} \tag{71}$$

$$T_{sim} \approx 67\,943.04\,\text{min} = 1132.384\,\text{h} = 47.2\,\text{d} \tag{72}$$

In order to reduce the huge computing time, the execution of the n_{rep} simulations can be carried out in parallel. For this purpose, a modified package `simulation_multiprocessing` of the simulation study design as presented in section 4.3 is developed. The `joblib.Parallel` module is used to launch separate Python worker processes to execute tasks on separate Central Processing Unit (CPU) in parallel. In this way, the computing time can be reduced depending on how many CPU cores are available. The `DMLSim` library (see Appendix C.9) contains the module `simulation_multiprocessing` which is customized and optimized to ensure that the simulation study is suitable for multiprocessing.

Due to the high computational demand shown in equation 72, HPC resources are used to perform the detail phase of the simulation study. Specifically, the Linux-Cluster Hummel, which is available via the regional computing center of the University of Hamburg, is used. The HPC-Cluster Hummel is designed to assist scientific projects with large parallel computing requirements [41]. Hummel is a multi-node Linux cluster with CentOS 7 procured as the operating system. Win-Secure Copy (SCP) 5.21 is used as a client to control the Linux cluster, which enables simple data transfer via SCP and control of the PuTTY bash sessions via Secure Shell (SSH). Python 3.9.12 is used as the module, the library `DMLSim` (see Appendix C.9) is installed via a local copy in the user's home directory on the Hummel cluster. The steps necessary to carry out the simulation study are described in the setup below:

```
-bash-4.2$ ssh front1
module: loaded site/slurm
module: loaded site/tmpdir
module: loaded site/hummel
module: loaded env/system-gcc
-bash-4.2$ module unload env
module: unloaded env/system-gcc
-bash-4.2$ module load python
module: loaded python/3.9.12
-bash-4.2$ python -m pip install --user --upgrade pip setuptools wheel
-bash-4.2$ python -m pip install --user -e /work/fwwx329/DMLSim-master
-bash-4.2$ cd $WORK
```

A Simple Linux Utility for Resource Management (SLURM) job is used to run the simulation on different nodes. SLURM is a popular job scheduler used in many Linux HPC clusters. It provides features for job submission, resource allocation, job monitoring, and job accounting. The following SLURM job is shown as an example:

```
#!/bin/bash
#SBATCH --job-name=sim_b01
#SBATCH --partition=std
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=24:00:00
#SBATCH --export=NONE
set -e
srun="srun -n1 -N1 --exclusive"
source /sw/batch/init.sh
module unload env
module load python/3.9.12
module list
pwd
date
cd /work/fwwx329/Detail_Phase
$srun python b01.py
echo "Node $(hostname -s)"
```

```
echo "TMPDIR: $TMPDIR"
exit
```

The SLURM jobs are saved as shell scripts in the working directory and transferred to the queue via the SLURM module with `sbatch b01.sh`:

```
-bash-4.2$ squeue --user fwwx329
             JOBID PART     NAME     USER ST       TIME  NODES NODELIST(REASON)
           2891043  std   sim_b12  fwwx329  R    1:37:08      1 node123
           2891042  std   sim_b11  fwwx329  R    1:37:11      1 node122
           2891041  std   sim_b10  fwwx329  R    1:37:14      1 node121
           ...
```

A short extract from the SLURM output of the job shows that the start of the execution of the different scripts was successful. For each main process, 32 Python workers were generated on CPU cores.

```
module: loaded site/slurm
module: loaded site/tmpdir
module: loaded site/hummel
module: loaded env/system-gcc
module: unloaded env/system-gcc
module: loaded python/3.9.12
Currently Loaded Modulefiles:
 1) site/slurm      2) site/tmpdir     3) site/hummel
4) python/3.9.12
/work/fwwx329/Shell_Scripts
Mon Feb  6 10:37:35 CET 2023
[Parallel(n_jobs=32)]: Using backend LokyBackend with 32 concurrent workers.
```

By running the scenarios on exclusive nodes and parallelising to 32 CPU cores per node, the estimated computing time can be reduced to:

$$n_{fits} = n_{rep} \cdot n_{learners} \cdot n_{settings} = \frac{1000}{32} \cdot 12 \cdot 1 = 375 \tag{73}$$

$$T_{sim} \approx T_{secondary} + n_{fits} \cdot \tilde{t}^{fit} = 10\,\text{min} + 375 \cdot 28.2596\,\text{s} = 186.6225\,\text{min} = 3.110\,\text{h} \tag{74}$$

However, it is to bear in mind that this consideration of the computational demand is only a rough approximation of the actual calculation times. In fact, the calculation times differ widely across the individual scenarios. As described in section 4.3, each scenario presented in Table 9 is executed as an instance of the developed simulation study `class simulation_study`. The resulting performance measures, attributes and a log file can be saved accordingly as shown in Figure 23.

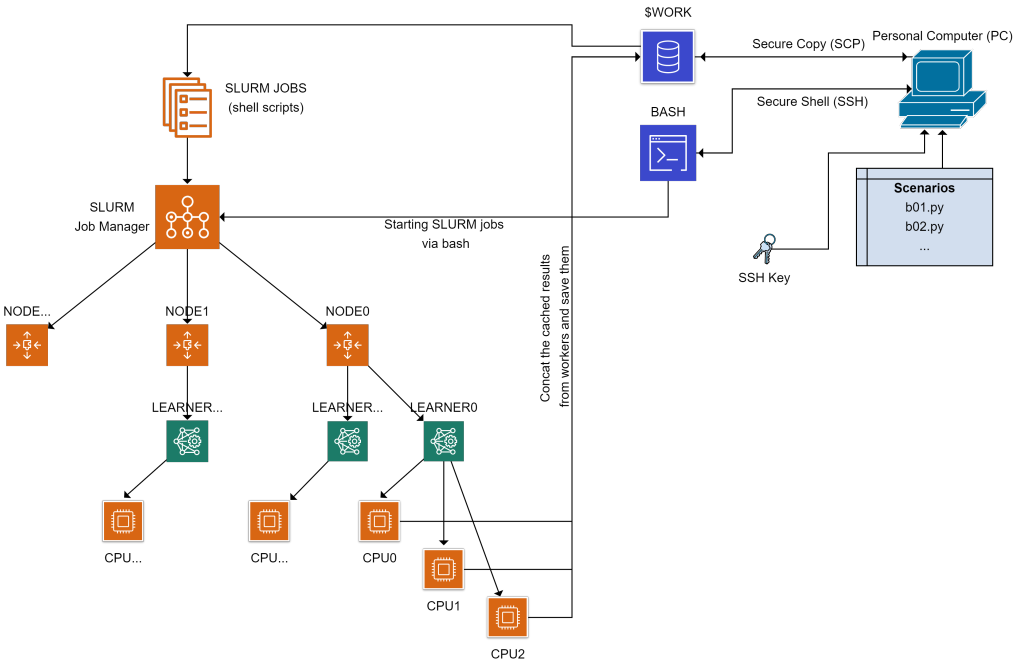


Figure 23: Workflow on the Hummel HPC Cluster

Source: own figure.

To conclude, working with Linux HPC-clusters demands a basic familiarity and comprehension of the cluster architecture, job submission, file management, job monitoring, and debugging. On the other hand, using multiprocessing and HPC resources can save a significant amount of computing time.

4.5 Analysis of the Detail Phase of the Simulation Study

The results of the detail phase of the simulation study are presented on the basis of the performance measures relative bias, RMSE, and coverage of the 95 % CI. All other performance measures can be retrieved from the log files in Appendix D.2. The histograms, as well as the boxplots for each scenario of the detail phase, are presented in the notebook `Simulation_Summary_attributes_review_Detail.ipynb` in Appendix C.3. Since the detail phase of the simulation study is carried out on the Hummel HPC cluster over several nodes in parallel, a subsequent merging of the partial results is necessary to obtain the same visualisations as in the exploratory phase. The notebook summarizing and presenting the resulting performance measures is available in Appendix C.3 `Simulation_Summary_performance_review_Detail.ipynb`.

The resulting relative bias for all six PLR scenarios of the detail phase is shown in Table 12 and for all six IRM scenarios in Table 15. The results are colour-coded, with red fields indicating a negative relative bias and blue fields a positive relative bias. The darker the field is filled with a certain colour, the greater the relative bias in a certain direction and therefore the greater the deviation of the estimand from the true value.

Additionally, Figure 26 shows the relative bias for all PLR scenarios and Figure 30 the relative bias for all IRM scenarios in a visualized way.

Besides, the RMSE is presented in Table 11 for the PLR scenarios and in Table 14 for the IRM scenarios. The resulting values are colour-coded with red fields indicating a high RMSE and green fields indicating a low RMSE for a certain column and therefore scenario. This means that the field with the highest RMSE for a certain scenario is coded in dark red whilst the field with the smallest RMSE is coded in dark green. In addition, Figures 24, 25, 28, and 29 illustrate the RMSE for a given number of covariates *dim_x* for the PLR and IRM scenarios. Comparing scenarios with the same causal model and the same number of confounding covariates provides an illustration of the development of the RMSE as datasets become larger.

Additionally, Table 13 and Figure 27 show the coverage of the 95 % CI for the PLR scenarios and Table 16 and Figure 31 show the coverage of the 95 % CI for the IRM scenarios. The performance measures in Tables 13 and 16 are colour-coded with fields with a coverage close to 100 % being filled in dark green and fields with a coverage close to 0 % being filled in dark red.

For reasons of space in certain tables the neural networks models are abbreviated as nn_0 to nn_9 and Random Forest as rf.

4.5.1 Partially Linear Regression Scenarios

Overall, the neural network models have lower RMSE values than the Lasso and Random Forest models for most of the scenarios (see Table 11). For scenarios B01, B02, B04, and B06 neural network model 4 with three hidden layers and a limited number of neurons per layer shows the best performance among all considered methods in providing accurate estimates. For scenario B03, the complex and deep neural network model 9 illustrates the best performance with regard to the RMSE. In scenario B05 the simple neural network models 0 and 1 with only one hidden layer provide accurate estimates. Nevertheless, in scenarios B01 and B04, the simple model 0 has the worst performance among all considered methods. Especially in scenario B03 where the dataset is large, Lasso and Random Forest fail to provide accurate estimates compared to the neural network models. On the other hand with increasing the number of covariates for the largest number of observations in scenario B06, Lasso and Random Forest remain at their performance level, whilst the neural network models show worse performance in each case. In summary, the results in Table 11 suggest that certain neural network models outperform other methods in accurately estimating a causal parameter across a wide range of data settings, including those with higher dimensions. For each scenario, there is at least one neural network model that outperforms Lasso and Random Forest.

Table 11: Root Mean Squared Error for Partially Linear Regression Scenarios in Detail Phase

Source: own table.

	B01	B02	B03	B04	B05	B06
lasso	0.0528	0.0407	0.0249	0.0557	0.0397	0.0242
rf	0.0601	0.0472	0.0255	0.064	0.0471	0.0296
nn_0	0.077	0.0373	0.0252	0.0788	0.0375	0.0287
nn_1	0.0495	0.0415	0.022	0.0523	0.0365	0.0269
nn_2	0.0538	0.0417	0.0206	0.057	0.0405	0.0269
nn_3	0.0513	0.04	0.0204	0.0547	0.0417	0.0255
nn_4	0.0478	0.036	0.0209	0.052	0.0382	0.0234
nn_5	0.0514	0.0402	0.0197	0.0566	0.0438	0.0267
nn_6	0.06	0.0456	0.0192	0.0732	0.0639	0.0407
nn_7	0.0538	0.0406	0.0181	0.0625	0.0469	0.0294
nn_8	0.0511	0.0403	0.0212	0.0557	0.0422	0.025
nn_9	0.0504	0.0383	0.0176	0.0562	0.0434	0.0277

When considering the dimensions in the scenarios further, an approximately linear relationship between the RMSE and the number of observations n_{obs} can be observed(see Figures 24 and 25). It is apparent that the RMSE declines for each method with increasing n_{obs} for a given number of covariates dim_x. This is reasonable as in general, more data supports obtaining highly accurate estimands.

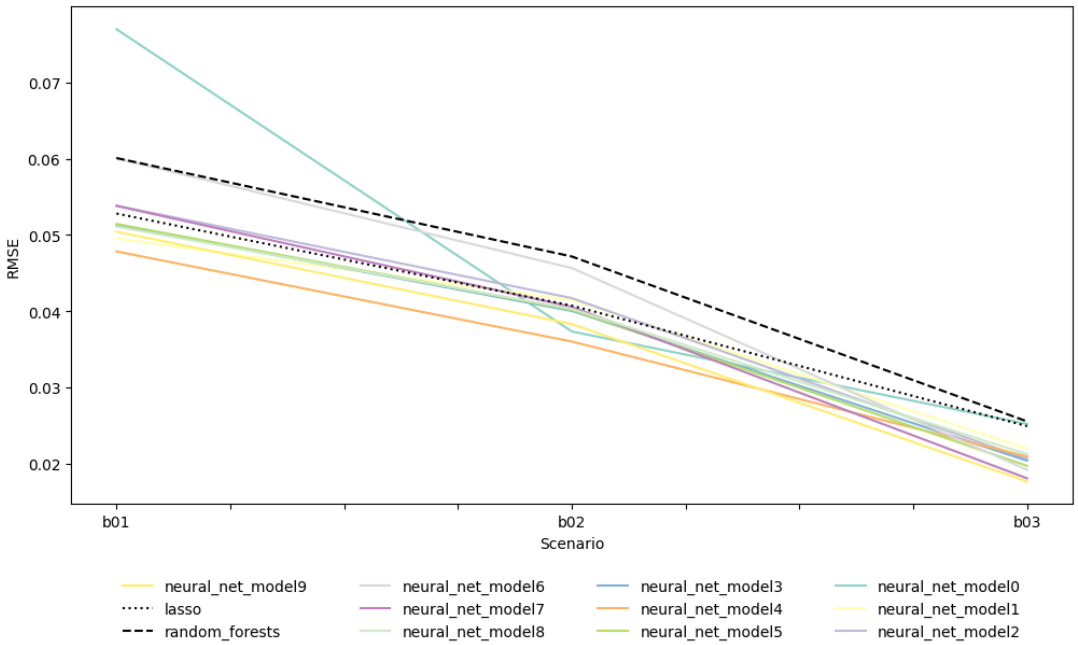


Figure 24: Root Mean Squared Error for Partially Linear Regression Scenarios with $dim_x = 20$

Source: own figure.

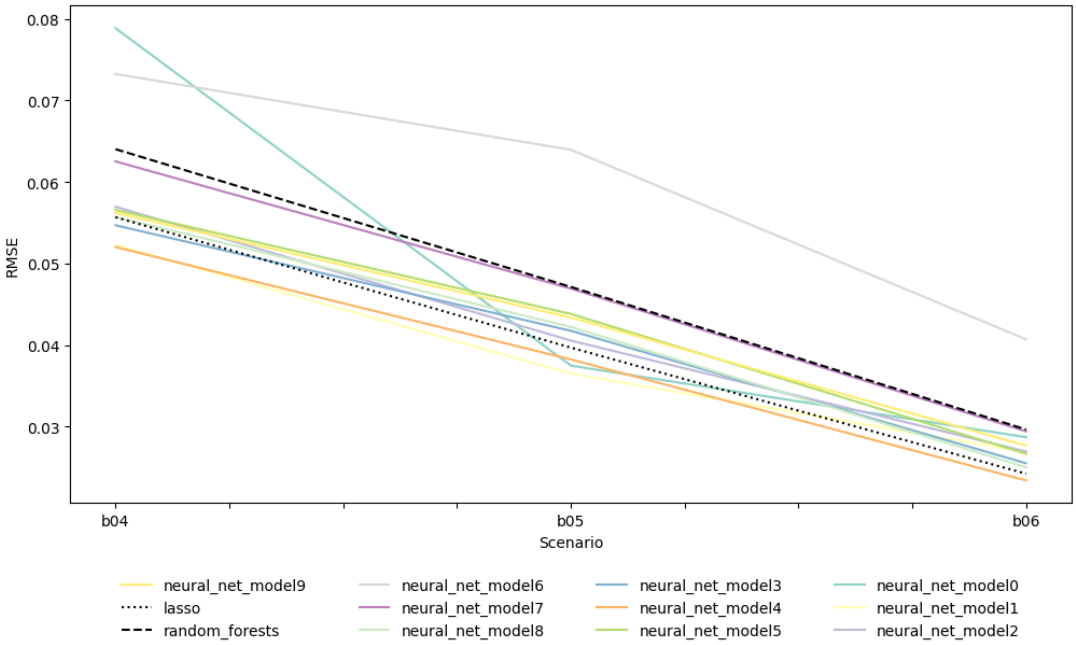


Figure 25: Root Mean Squared Error for Partially Linear Regression Scenarios with $dim_x = 100$

Source: own figure.

It shall be noted that the RMSE is sensitive to the choice of n_{obs} which is why it should not be the only performance measure that is looked to analyse the results of a simulation study [80]. Therefore, Table 12 highlights the relative bias as another performance measure for the analysis. Looking at the resulting relative bias, it shows that most of the methods have some level of bias in each scenario. In some cases, the bias is quite low (less than 1 %), while in other cases it is significantly higher (up to 6 %). In general, it is observed that nearly all methods show a negative relative bias, indicating that on average the estimated value of the treatment effect is lower than the true value. Furthermore, it can be stated that most of the methods show a relative bias of approximately 2 % for most scenarios. This indicates that all methods prove to provide approximately unbiased estimates of the causal effect in a PLR model for various data settings.

The good performance of neural network model 4 in several scenarios shows again as this model provides nearly unbiased estimates for scenarios B01 and B04, where the size of the dataset is only limited. There is no unambiguous or clear statement for all methods whether increasing the number of observations or increasing the number of covariates has a positive impact on the relative bias or not. For Lasso and Random Forest, for example, the relative bias decreases with an increasing size of the dataset. For Lasso, the relative bias is impacted positively by increasing the number of covariates for a given number of observations whilst for Random Forest the relative bias is affected in a negative way. For the neural network models, this relationship can not be clearly stated and differs for different model architectures.

Similar to the results presented for the RMSE, the simple neural network models 0 and 1 show good performance in providing unbiased estimates for scenarios B02 and B05 where the number of observations is on the mid-level. Especially for scenario B05, model 1 outperforms all other methods significantly.

Summarizing the results in Table 12, it shows that there are several neural network models that outperform the standard ML methods Lasso and Random Forest for each scenario or at least show the same performance. Anyhow, especially Lasso proves to provide a relatively consistent value for the relative bias for all considered data settings.

Table 12: Relative Bias for Partially Linear Regression Scenarios in Detail Phase in %

Source: own table.

	B01	B02	B03	B04	B05	B06
lasso	-2.41 %	-2.33 %	-2.21 %	-2.42 %	-2.08 %	-2.1 %
rf	-3.4 %	-3.08 %	-2.29 %	-3.64 %	-3.16 %	-2.72 %
nn_0	6.01 %	1.57 %	-2.25 %	5.91 %	1.36 %	-2.61 %
nn_1	-1.22 %	-2.34 %	-1.9 %	0.59 %	-0.8 %	-2.41 %
nn_2	-2.36 %	-2.39 %	-1.74 %	-1.81 %	-1.88 %	-2.42 %
nn_3	-1.6 %	-2.1 %	-1.71 %	-1.59 %	-2.08 %	-2.26 %
nn_4	-0.49 %	-1.31 %	-1.78 %	-0.21 %	-1.4 %	-2.01 %
nn_5	-1.94 %	-2.12 %	-1.62 %	-2.13 %	-2.4 %	-2.37 %
nn_6	-3.05 %	-2.8 %	-1.53 %	-4.42 %	-4.86 %	-3.84 %
nn_7	-2.21 %	-2.15 %	-1.42 %	-3.06 %	-2.79 %	-2.67 %
nn_8	-2.14 %	-2.23 %	-1.8 %	-2.42 %	-2.34 %	-2.2 %
nn_9	-1.4 %	-1.72 %	-1.38 %	-1.84 %	-2.32 %	-2.47 %

The resulting relative bias for each method and scenario is visualized in Figure 26. This illustration supports the descriptions above that there are several neural network models that show as-good-as or better performance than Lasso and Random Forest with regard to obtaining unbiased estimates for the causal effect. It is also visible that most of the methods show a negative relative bias for all scenarios. The outliers of model 0 and model 6 for certain scenarios are clearly visible, indicating that the simplest possible network architecture and a wide architecture with many neurons per hidden layer failed for these data settings. Especially the wide architecture of model 6 shows no good performance for data settings with many covariates as the values for the relative bias are up to twice as high compared to a scenario with the same number of observations but a reduced number of covariates. There is no clear picture of whether deeper neural network models with more complex architectures perform better than simpler ones. It can be seen that besides neural network model 4 the deep and complex model 9 also shows a performance among the best methods for all data settings.

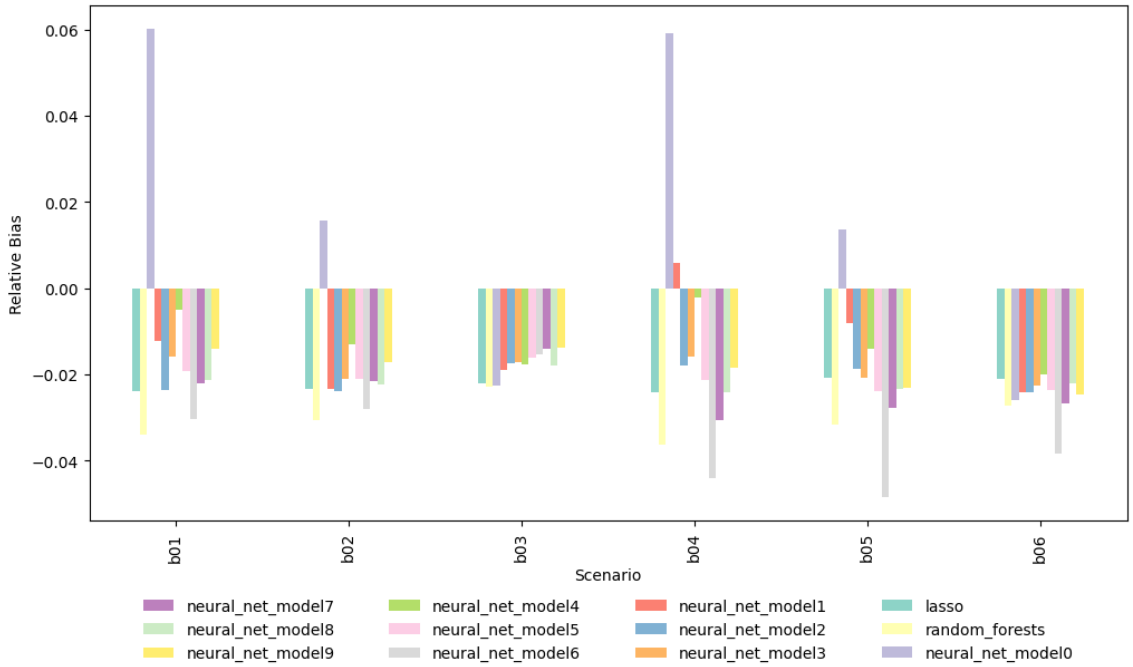


Figure 26: Relative Bias for Partially Linear Regression Scenarios in %

Source: own figure.

In addition to the RMSE and the relative bias, the coverage of the 95 % CI is analysed. The resulting values are presented in Table 13 and visualized in Figure 27. In general, it can be seen that the coverage varies quite substantially across different methods and data settings. The standard ML methods Lasso and Random Forest tend to have a lower coverage compared to most neural network methods. For all considered scenarios, one of the neural network models proves to have the best coverage. For the scenarios with a limited number of observations B01, B02, B04 and B05, the simple neural network model 1 together with models 4 and 0 shows the best performance with regard to the coverage. Almost all considered models show a coverage of around 90 % for those scenarios. For scenarios B03 and B06 with a large number of observations, the coverage is significantly lower across all considered methods. For example, in scenario B06, the 95 % CIs encompassed the true value for over fifty percent of the simulation runs solely for neural network model 4. Also for B03, the coverage is pretty low. In Figure 27 it is obvious that the coverage increases for more complex model architectures but even for the best-performing neural network model 9, the 95 % CIs only contained the true value in three out of four cases. Another notable aspect depicted in Figure 27 is that neural network model 6 lacks behind the performance of all other neural network models for all scenarios. This poor performance with regard to the coverage also reflects the descriptions that arise from analysing the relative bias and the RMSE. All three considered performance measures indicate that the neural network model 6 with a wide architecture and many neurons per hidden layer is not as suitable as other methods to perform causal inference for the given DGP.

Table 13: Coverage of 95 % Confidence Interval for Partially Linear Regression Scenarios in Detail Phase in %

Source: own table.

	B01	B02	B03	B04	B05	B06
lasso	91.3 %	88.2 %	44.2 %	90.7 %	90.0 %	47.8 %
rf	87.3 %	81.6 %	42.6 %	85.6 %	83.5 %	28.6 %
nn_0	75.6 %	93.4 %	42.7 %	75.1 %	92.4 %	31.9 %
nn_1	94.0 %	88.1 %	56.5 %	93.0 %	92.5 %	38.8 %
nn_2	91.7 %	88.5 %	62.9 %	90.2 %	89.8 %	38.1 %
nn_3	93.2 %	89.5 %	63.3 %	90.9 %	87.4 %	43.1 %
nn_4	94.4 %	92.6 %	60.3 %	92.5 %	91.1 %	51.3 %
nn_5	92.6 %	89.1 %	66.7 %	90.1 %	85.9 %	40.6 %
nn_6	88.1 %	84.4 %	67.9 %	79.6 %	66.6 %	9.6 %
nn_7	91.6 %	89.2 %	72.5 %	87.2 %	84.5 %	30.8 %
nn_8	92.4 %	89.2 %	60.6 %	91.0 %	88.4 %	45.2 %
nn_9	93.3 %	90.8 %	74.4 %	91.0 %	86.8 %	36.8 %

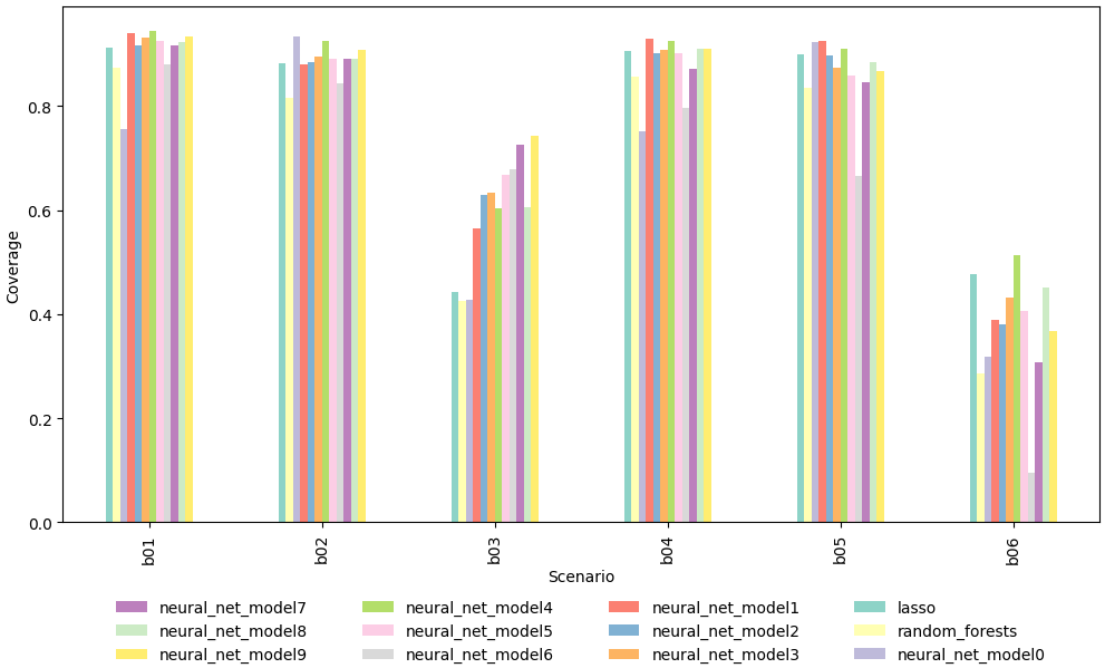


Figure 27: Coverage of 95 % Confidence Interval for Partially Linear Regression Scenarios in %

Source: own figure.

4.5.2 Interactive Regression Model Scenarios

Table 14 shows the RMSE for the IRM scenarios. One thing that strikes at first is the incredibly large values for Random Forest for scenarios B08 and B09 and for Lasso for scenario B10 that are in the order of millions. This behaviour is not as expected which is

why the raw data for these scenarios has been studied in detail (see Appendix D.2). The high values in the resulting performance measure arise only from a few simulation runs where the estimand is calculated to be tremendously high. By measuring the RMSE as described by the equation in Table 5, these large estimated values distort the resulting performance measure. Since this behaviour only appeared selectively for the stated scenarios it can be excluded that the generic and structured design of the simulation study as presented in section 4.3 is incorrect or faulty. It is assumed that the error rather originates when applying the ML method to the data, for example, due to the non-convergence of some coefficients. This error has not been ultimately fathomed and solved in this thesis which makes this one interesting topic for further research. Nevertheless, all other values seem to be appropriate and behave as expected, which is why the three learner-scenario combinations are neglected for the rest of the analysis.

Table 14: Root Mean Squared Error for Interactive Regression Model Scenarios in Detail Phase

Source: own table.

	B07	B08	B09	B10	B11	B12
lasso	0.370963	0.220284	0.073967	$63.481 \cdot 10^7$	1.269182	0.065226
rf	0.192920	$80.577 \cdot 10^6$	$32.042 \cdot 10^6$	0.160172	0.117741	0.040606
nn_0	0.766269	0.559720	0.096303	0.756460	0.562290	0.095827
nn_1	0.250254	0.190399	0.078546	0.225397	0.167636	0.096614
nn_2	0.209645	0.170836	0.074713	0.189019	0.152660	0.082179
nn_3	0.223293	0.164087	0.068704	0.190385	0.146109	0.067495
nn_4	0.191792	0.145444	0.068761	0.200621	0.132372	0.050886
nn_5	0.197233	0.144174	0.063727	0.176548	0.133530	0.056028
nn_6	0.203371	0.145983	0.067800	0.185556	0.136353	0.046173
nn_7	0.198962	0.143738	0.064649	0.185956	0.140338	0.055691
nn_8	0.195402	0.143909	0.065205	0.171192	0.133010	0.049457
nn_9	0.200232	0.146991	0.068648	0.188387	0.143141	0.049131

In general, the Random Forest method shows superior performance for most scenarios with regard to accurate estimations (see Table 14). Several neural network models, especially models 4, 5, and 6, come close to the performance of Random Forest but never reach it. It can be seen that the simple neural network architectures do not perform well for any data settings. Besides that, almost all neural network models are showing a better performance with regard to the RMSE compared to Lasso. The simplest neural network model 0 only shows relatively acceptable performance for the scenarios with a large dataset. In all other scenarios, model 0 fails to provide accurate estimates.

Similar to the PLR scenarios, an approximately linear relationship between the RMSE and the number of observations can be observed for the IRM scenarios (see Fig. 28 and 29). The aforementioned outliers are not displayed in Figure 29 and are displayed dot-wise in Figure 28. The relationship between RMSE and n_{obs} is reasonable as described above but also indicates the sensitivity of this performance measure with regard to the size of the dataset.

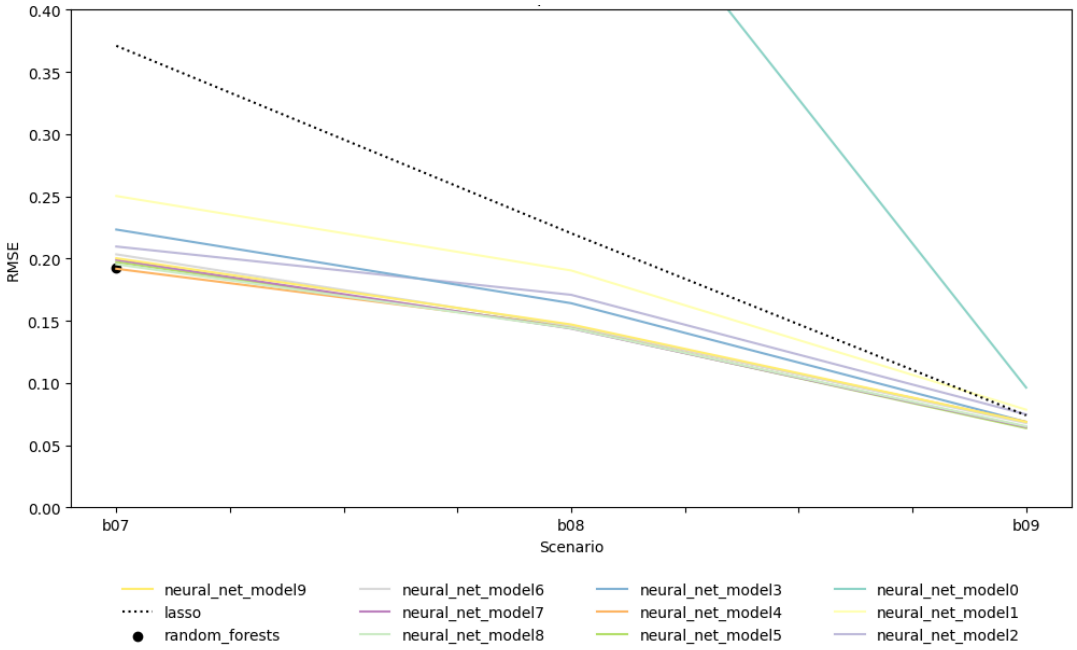


Figure 28: Root Mean Squared Error for Interactive Regression Model Scenarios with $dim_x = 20$

random_forest and nn_0 are only partially displayed.

Source: own figure.

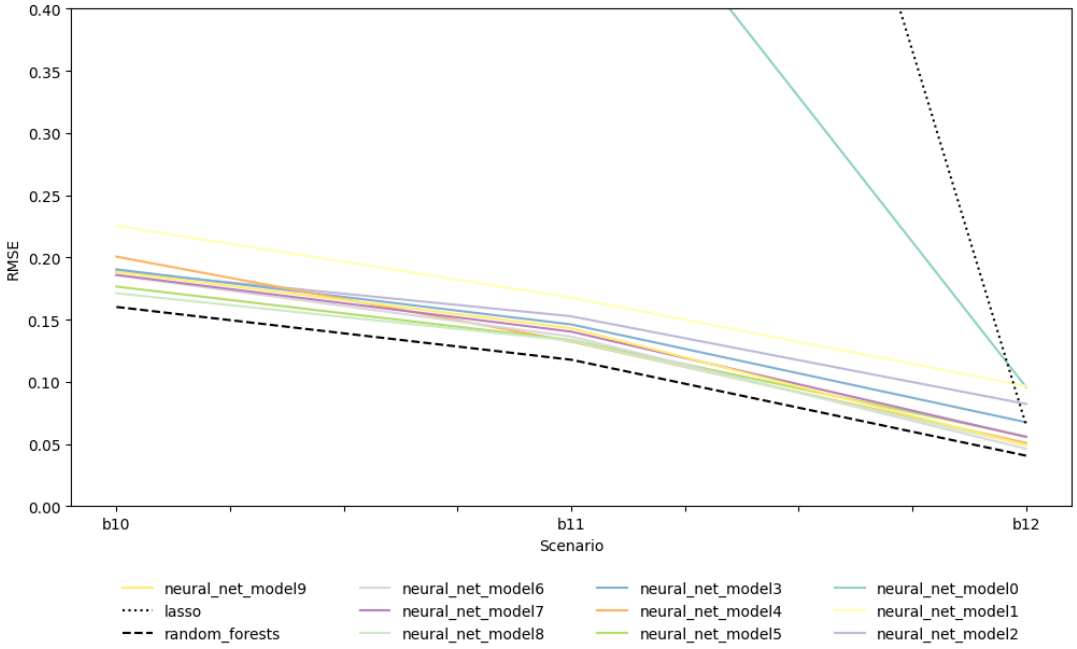


Figure 29: Root Mean Squared Error for Interactive Regression Model Scenarios with $dim_x = 100$

Lasso and nn_0 are only partially displayed.

Source: own figure.

Because of the aforementioned sensitivity of the RMSE to the number of observations, the relative bias is analysed additionally. The relative bias for the IRM scenarios is shown in Table 15 and visualized in Figure 30. In accordance with the descriptions above, one can see the same outliers that are not discussed further. In general, it is apparent that most of the resulting values for the relative bias are negative. This indicates that the estimated causal effect is smaller than the true value on average. The relative bias varies quite significantly between the different methods and scenarios. On average, the relative bias is higher compared to the PLR scenarios with around 3 % to 4 % for most of the methods and scenarios. In contrast to the PLR scenarios, there are some outliers, except for the ones already mentioned, which show a relative bias of up to 70 %. Additionally, there are several neural network models that have a two-digit relative bias and therefore fail at providing unbiased estimates of the causal parameter.

Table 15: Relative Bias for Interactive Regression Model Scenarios in Detail Phase in %

Source: own table.

	B07	B08	B09	B10	B11	B12
lasso	-2.33 %	-2.5 %	-3.21 %	$-23.561 \cdot 10^8$ %	-7.95 %	-2.75 %
rf	-3.55 %	$-25.171 \cdot 10^7$ %	$-19.608 \cdot 10^7$ %	-1.01 %	-1.07 %	-1.59 %
nn_0	-69.47 %	-50.74 %	-6.77 %	-70.49 %	-52.9 %	-7.91 %
nn_1	-15.69 %	-12.81 %	-4.42 %	-13.61 %	-10.62 %	-8.06 %
nn_2	-10.27 %	-10.4 %	-3.68 %	-7.28 %	-8.13 %	-6.42 %
nn_3	-11.24 %	-9.56 %	-2.55 %	-7.79 %	-7.0 %	-4.69 %
nn_4	-1.06 %	-5.55 %	-3.65 %	9.2 %	1.0 %	-2.85 %
nn_5	-5.78 %	-4.69 %	-2.61 %	-1.36 %	-1.49 %	-3.14 %
nn_6	-3.55 %	-2.31 %	-4.38 %	-1.76 %	-1.4 %	0.8 %
nn_7	-4.72 %	-3.22 %	-3.89 %	-2.36 %	-2.11 %	2.69 %
nn_8	-6.16 %	-5.68 %	-3.68 %	-2.76 %	-2.03 %	-2.03 %
nn_9	-5.38 %	-4.99 %	-4.55 %	-2.1 %	-2.84 %	-0.26 %

For scenarios B07 and B11, the neural network model 4 shows the best performance in providing unbiased estimates of the causal effect, outperforming Random Forest and Lasso significantly. Neglecting the outliers, Random Forest shows a good performance for all scenarios, especially with an increased number of covariates. In general, it can be observed that most of the methods improve significantly with an increasing number of covariates. For example for a given number of observations $n_{obs} = 10000$ (scenarios B09 and B12), the neural network models 6 and 9 are able to heavily reduce the relative bias with an increasing number of covariates dim_x and provide unbiased estimates. Although this applies to most of the methods, some other methods, especially the simple neural network models, show worse performance with an increased number of covariates. In Figure 30 it can be seen that especially the estimates of the models 0 - 3 are partially heavily biased.

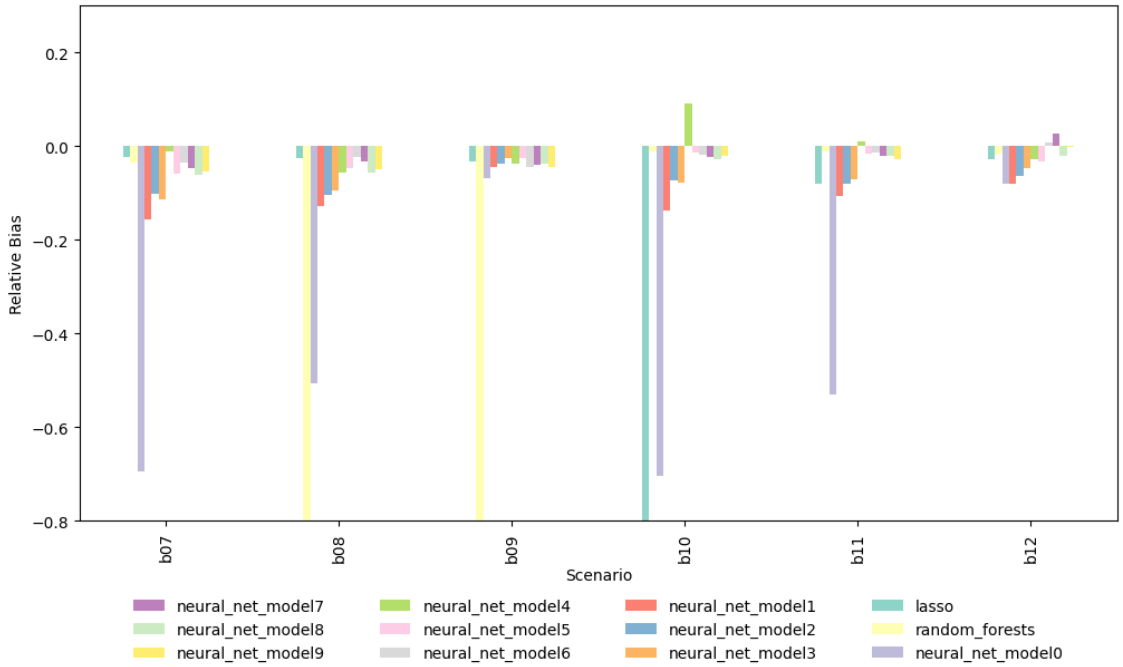


Figure 30: Relative Bias for Interactive Regression Model Scenarios in %

Source: own figure.

There is no clear picture of whether increasing the number of observations for a given number of covariates and therefore reducing the dimension improves the calculation of unbiased estimates or not. On the one hand, for certain neural network models (for example model 8), increasing the number of observations reduces the relative bias. On the other hand, for example for model 5, the same behaviour can be observed for $dim_x = 20$ but for $dim_x = 100$ the behaviour is the other way around. Summarizing the results in Table 15, it is obvious that there is at least one neural network model that outperforms the standard ML methods Lasso and Random Forest for each scenario or that shows a similar performance. Anyhow, especially Random Forest proves to provide a consistent value for the relative bias for the data settings with an increased number of covariates. There is no neural network that performs well for all given scenarios.

In addition to the RMSE and the relative bias, the coverage of the 95 % CI is analysed. The resulting values are presented in Table 16 and visualized in Figure 31. In general, it can be seen that the coverage varies quite substantially across different methods and data settings. The described outliers for Random Forest in scenarios B08 and B09 and for Lasso in scenario B10, which were visible before, are not obvious when looking at the coverage. This is because only a few simulation runs generate tremendously high estimates for the causal effect. These high estimates heavily deviate from the true value and, as a result, contribute to the high value of the resulting performance measures RMSE and relative bias. As the coverage per definition represents the proportion of times where the CIs contained the true value, this error in calculating the estimates does not destroy the interpretability of this performance measure. Therefore, the coverage is one suitable performance measure to analyse and discuss those learner-scenario combinations that

have been neglected before.

Table 16: Coverage of 95 % Confidence Interval for Interactive Regression Model Scenarios in Detail Phase in %

Source: own table.

	B07	B08	B09	B10	B11	B12
lasso	95.3 %	95.2 %	85.1 %	98.2 %	96.2 %	90.7 %
rf	91.6 %	90.6 %	74.0 %	93.6 %	92.3 %	88.9 %
nn_0	12.5 %	9.4 %	46.1 %	7.9 %	2.3 %	34.9 %
nn_1	87.1 %	81.4 %	56.2 %	91.0 %	88.9 %	28.8 %
nn_2	91.1 %	81.6 %	56.1 %	94.6 %	89.8 %	41.4 %
nn_3	90.7 %	85.0 %	63.4 %	96.3 %	91.7 %	56.6 %
nn_4	93.8 %	90.3 %	67.1 %	94.8 %	94.9 %	81.3 %
nn_5	91.0 %	88.9 %	65.9 %	94.7 %	92.6 %	72.8 %
nn_6	91.9 %	89.6 %	54.2 %	93.8 %	93.1 %	83.4 %
nn_7	93.1 %	91.5 %	56.2 %	94.5 %	93.0 %	76.1 %
nn_8	91.8 %	90.2 %	60.2 %	95.9 %	94.1 %	80.1 %
nn_9	93.6 %	90.7 %	53.8 %	94.9 %	93.3 %	85.6 %

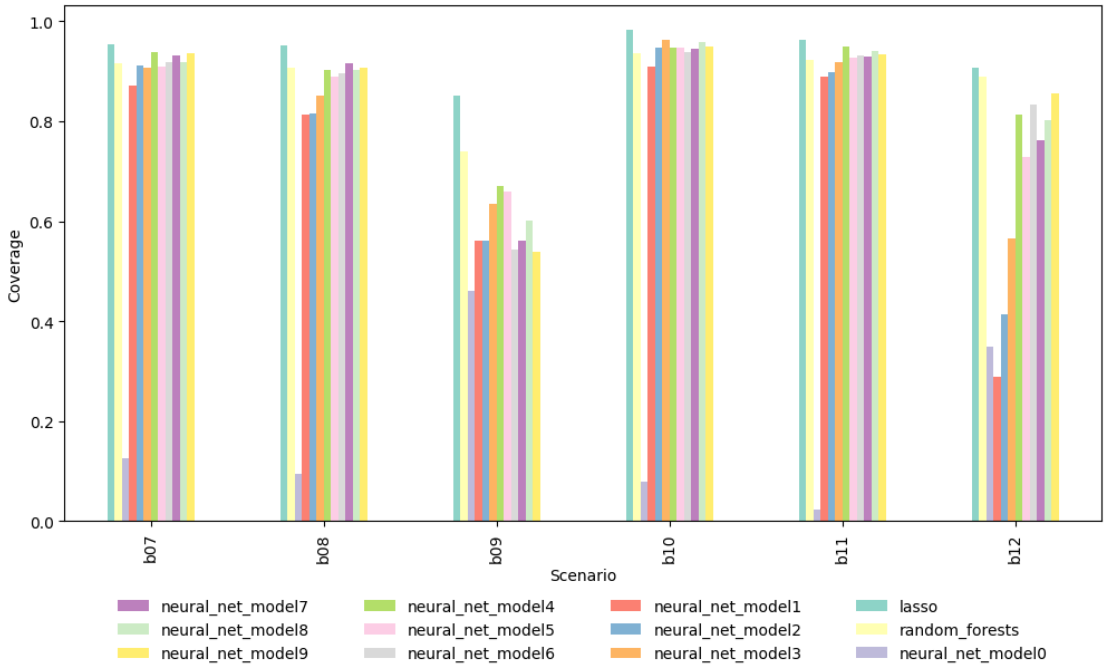


Figure 31: Coverage of 95 % Confidence Interval for Interactive Regression Model Scenarios in %

Source: own figure.

It can be seen that the simple neural network model 0 shows poor performance with regard to the coverage. Especially for the smaller datasets, the CIs only contained the true value of the causal parameter for a few simulation runs. Overall, Lasso shows the best performance in terms of coverage for all scenarios. For the scenarios with smaller datasets

(B07, B08, B10, and B11) the coverage is in each case above 95 %. For scenarios B09 and B12 with an increased number of observations, the coverage declines for all considered methods. It is noticeable that the decrease is the least for the standard ML methods Random Forest and Lasso whilst the coverage gets much lower for the neural network models. For all methods, the coverage declines with an increasing number of observations.

Only various neural network models can match the good performance of Lasso and Random Forest in some of the scenarios. Neural network model 4 shows a comparatively good performance across all scenarios. In general, the more complex neural network architectures seem to perform better than the simpler ones. In contrast to the coverage of the PLR scenarios (see Table 13), the coverage of the scenario with the largest number of observations and the number of covariates seems to be acceptable for some of the methods considered. It is striking, that the coverage is better with an increased number of covariates for a given number of observations in each case.

4.6 Summary Simulation Study

For this thesis, a simulation study was conducted to study the behaviour of the newly implemented DL approaches in the context of causal inference with the DML package. The objectives were to prove the successful implementation of these new methods for a variety of scenarios, to study the finite sample properties of neural networks in causal inference, and to compare DL approaches to other traditional ML methods. In order to plan the simulation study in a procedural manner, the structured ADEMP approach by Morris et al. [80] was followed. The simulation study was split into two phases which first allowed to analyse the performance according to the stated objectives in a reduced scope for various DGPs in an exploratory phase. This has been followed by focusing on specific DGPs for the detail phase. In the detail phase, the study was conducted to its full extent. The simulation experiment with the described requirements was designed by creating a corresponding Python library `DMLSim` that utilizes several packages and modules to enhance the adaptability and flexibility of the simulation study across all specified settings.

After planning and designing the simulation study, the two aforementioned phases are executed in the sense of a simulation experiment. Hereby it is also focused on the computational demand and the simulation is outsourced to HPC resources whenever needed.

The results of the exploratory phase reveal that for the PLR scenarios Lasso as a standard ML method shows strong performance in providing unbiased and accurate estimates of a causal effect. The performance of the different neural network models varies across the considered scenarios. Some neural network models clearly outperform the standard ML methods for a specific setting, but in general, the implemented DL approaches are falling short in terms of performance. A similar picture can be seen for the IRM scenarios where neural network models just showed a good performance in a limited number of settings. All in all, the exploratory phase revealed no systematic deviation of the neural network models but clearly showed that the performance of those

DL methods is heavily dependent on the considered DGP. Besides, the performance of the standard ML methods Lasso and Random Forest proved to be much more consistent across the considered DGP. Consequently, two DGPs where the use of DL methods seemed promising due to superior performance could be chosen for the detail phase of the simulation study.

The detail phase showed that the considered methods mainly provide unbiased and accurate estimates of the causal effect also for different data dimensions and sizes of data. It is obvious that the accuracy indicated by the RMSE is generally declining with an increased number of observations. It can be summarized that the performance of a neural network model heavily depends on its architecture and the considered setting. For instance, while neural network model 6 exhibited the poorest performance for nearly all PLR scenarios concerning relative bias, RMSE, and coverage, it demonstrated good performance for several IRM scenarios. Specifically, it resulted in the lowest relative bias for scenario B08 and produced a relative bias among the best methods for B10 to B12, with consistent outcomes. This exemplary consideration of one neural network model shows that the performance heavily differs across various data settings for the same model architecture. It is therefore striking that a suitable neural network architecture needs to be chosen upon a given data setting in order to achieve high-quality estimates.

In general, the simulation study revealed that even a simple Lasso model with standard settings shows a good performance in estimating the causal effect for various data settings. Nevertheless, there are several scenarios where specific neural network models showed a better performance. There is no indication that neural network models are in general superior in larger data settings or in settings with a higher data dimension compared to standard ML methods. Additionally, it is not visible in the results that more complex and deep neural networks generally perform better than simpler ones. Actually, even the simplest neural network model 0 with only one neuron in one hidden layer showed a fairly low relative bias in several scenarios (e.g. B02 and B05).

The results of the detail phase of the simulation study are largely consistent with other studies' findings. The observation of Chernozhukov et al. [17, p. 44] that "small sample size causes stability problems in training the neural network", could not be supported within this simulation study. Moreover, the results are compared to the findings of the two principal studies referenced in section 4.2, which this simulation study draws upon. Both, Farell et al. [26] and Whata et al. [118], find that the average coverage of the 95 % CI is between 90 % and 95 % for most of the used neural network models. Except for the scenarios with large data sizes (B03, B06, B09, and B12), this is consistent with the results of this simulation study. Furthermore, Farell et al. [26] show that there is no clear picture across various neural network models whether increasing the number of covariates affects the relative bias in a certain direction. This is in line with the findings of this study as the relative bias declines for certain neural network models with an increasing number of covariates whilst the relative bias increases for others. Consistent with the findings of Farell et al. [26], the coverage declines with an increasing number of covariates for a given number of observations for most of the neural network models. Other than that, Farell et al. [26] also obtain the result that deep and complex

neural network models are not showing a better performance per se but rather that different architectures proof to be suitable for different scenarios. Just as in the results presented by Whata et al. [118], no clear relationship between the number of observations and the bias can be seen. Whilst for some methods the bias declines from $n_{obs} = 500$ to $n_{obs} = 1000$ and increases again for $n_{obs} = 10000$, the bias declines for other methods in general with an increased n_{obs}. According to and in line with Farell et al. [26, Supplement, p.2] the coverage of the various neural network models among each other is relatively similar with "more complex architecture not exhibiting any systematic patterns of length inflation". One implication of the findings from Farell et al. [26] is that an increased bias might be expected due to the application of regularization methods for neural network models. As no differentiation has been made between regularized and non-regularized DL methods this cannot be supported or rejected within this study but remains one interesting direction for further research.

All in all the differences in the performance of the various methods are assumed to be due to the different complexity of the scenarios and the models' ability to capture the underlying patterns. Fundamentally, the data used in this study has been structured and not deliberately high-dimensional. There are several results indicating that a neural network model with a certain model architecture can produce high-quality estimates of a causal effect. The simulated data and the resulting performance measures can be one starting point for the further analysis of the finite sample properties of DL methods in the context of DML. One interesting topic for further research could be to investigate how an optimal network architecture for a given data setting can be derived.

Although much emphasis in the design of the simulation study is placed on the reproducibility of the results, the log files containing the model attributes, raw estimates, performance measures, parameters and simulation specifications are retrievable in Appendix D.1 for the exploratory phase and in Appendix D.2 for the detail phase.

5 Use Cases for Causal Machine Learning with Deep Learning Approaches

After studying the use of DL methods as learners in causal inference problems for various data settings, this section is intended to present possible use cases for CML with DL approaches.

As described in section 4.6, the simulation study revealed that for structured synthetic datasets the standard ML methods Lasso and Random Forest showed in general superior and more consistent performances compared to the various neural networks which represent the DL approaches. Despite there being several settings where specific neural network architectures outperformed the standard ML methods significantly, those approaches are in general not as good as Random Forest or especially Lasso. As stated, the performance of the neural network models heavily depends on the model architecture. This architecture needs to be chosen upon a given data setting to achieve high-quality results in estimating a causal effect.

Besides performance, computational demand is another important factor in determining which methods are suitable for which use case. As presented in Table 10, the average time when fitting the model varies substantially across the considered methods and data settings. For example for calling the `fit` method in a large data setting with $n_{obs} = 10000$ and $dim_x = 100$ each fitting of a Random Forest model takes up to 32x more time than the fitting of the fastest neural network model. Especially for the high-dimensional setting with $n_{obs} = 500$ and $dim_x = 100$ all neural network models show significantly less computational time than the ML methods Lasso and Random Forest. Of course, providing unbiased and accurate estimates of a causal effect remains a top priority when selecting an appropriate learner, but especially in the context of Big Data with large and potentially high-dimensional datasets, computational costs also play an increasingly important role. This brings along one interesting direction for use cases of DL approaches in the context of CML for Big Data settings.

Since most practical applications rely on observational data rather than data from a randomized control trial, several challenges arise (see section 2.1). In general, studying causal effects with observational data requires the adjustment with confounding factors. As described in the theoretical foundations in sections 2.5 and 2.6 there is an increasing field of research that focuses on the use of text and image data to obtain confounding factors. Text data is inherently high-dimensional and for state-of-the-art representation learning, DL methods like BERT are recommended to obtain high-quality results. Especially due to the low computational time of neural network models for more high-dimensional settings and due to the success of recent DL methods in the field of NLP, causal inference using unstructured text data to obtain confounders seem as an interesting use case for CML with DL approaches. In particular, the inductive learning of confounders to condition for all aspects of the observed text is one use case where DL approaches could be beneficial.

Similarly to that, computer vision tasks rely on state-of-the-art DL methods like BEiT to efficiently use image data. Therefore, use cases for DL approaches can be found when images are used to obtain unobserved confounders. Especially since image data is usually more granular than the actual unit of the analysis due to its resolution, it can be used together with DL approaches to construct relevant confounders.

As mentioned earlier, many potential use cases for DL approaches in CML arise from the prevalent existence of a huge amount of unstructured data, especially text and image data. In economics, this data can originate from digital platforms like for example marketplaces or social media. Due to their state-of-the-art performance in NLP and computer vision as well as their proven computational efficiency for large and potentially high-dimensional data settings, DL approaches seem to be not only suitable but inevitable for making causal inferences. In particular, DL approaches prove to be extremely useful when there are latent confounding factors obtained from text or images. The simulation study showed that standard ML methods might be sufficient as learners for tabular settings with structured data but may involve an increased computational cost.

Nevertheless, it is worth mentioning that potential use cases for DL approaches as learners in CML can also be found for structured data settings. The customization of the model architecture to the given data setting can provide superior performance compared to standard ML methods. This has been shown by the strong performance of specific neural network models for selected DGPs and data settings in the simulation study.

One of such described use cases of DL approaches in the context of CML is given exemplary and is extensively described in the following section 6 as a case study. This case study is intended to show how DL approaches can be efficiently applied to a real-world causal inference problem.

6 Case Study

This case study will focus on estimating price elasticities according to Roemheld [90] on a given data set of Amazon sales data. The case study presented and described in the subsequent sections of the thesis focuses on products in the toys and games category. Besides, there is also data available for products of the category chairs. The procedure presented in this case study is also conducted for the chairs products but is not particularly described. The respective analysis and descriptions for the chair products can be traced in the notebook in Appendix C.7.

6.1 Problem Definition

In general, elasticities are a fundamental concept used in economics to measure the responsiveness of one variable to the change in another [103]. In other words, elasticities describe how changing one variable affects another variable. More specifically, the price elasticity of demand measures the responsiveness of demand for a good or service to changes in its price [103]. The price elasticity of demand is calculated as the percentage change in the quantity demanded of a good or service divided by the percentage change in its price [103]:

$$\epsilon_{q,p} = \frac{\text{Percentage change in quantity demanded}}{\text{Percentage change in price}} \tag{75}$$

The price elasticity of demand can be either elastic or inelastic depending on the value of $\epsilon_{q,p}$. A good or service is said to have an elastic demand if the price elasticity of demand is greater than one ($|\epsilon_{q,p}| > 1$) which means that a change in price leads to a relatively larger change in quantity demanded [103]. Conversely, a good or service is said to have an inelastic demand if the price elasticity of demand is less than one ($|\epsilon_{q,p}| < 1$), meaning that a change in price will result in a relatively smaller change in the quantity demanded [103]. The sign of the price elasticity of demand describes in which direction the demand is affected by a change in price [103]. Typically, price elasticities of demand are negative [90]. This means that the quantity demanded decreases with an increasing price. The degree of the price elasticity of demand for a particular good or service depends on several determinants, including the availability and closeness of substitutes, the proportion of income spent on the good or service, and the time horizon of the analysis [103]. In general, goods and services that have close substitutes and that are considered non-essential tend to have a more elastic demand as people may switch to the alternative product when prices rise [103]. Goods and services considered essential and that have few substitutes tend to have a more inelastic demand. Besides, the income effect describes that the higher the proportion of income spent on a good or service the more one will be forced to reduce consumption when prices rise and the higher the respective price elasticity [103]. Lastly, the longer the time period after a change in price the higher the respective price elasticity of demand [103].

In addition to the price elasticity of demand, there is also price elasticity of supply that measures the responsiveness of supply to changes in the price of a good or

service [103]. Price elasticity of supply is calculated as the percentage change in the quantity of a good or service supplied divided by the percentage change in its price [103].

Understanding price elasticities is important for businesses, policymakers and researchers. Companies can use price elasticity to make informed pricing decisions, anticipate changes in demand and adjust their marketing strategies. In general, it is inevitable for a company to know the effect of a change in price on the quantity demanded [103]. For example, a retailer can use the price elasticity of demand to adjust prices in an optimal way and thereby increase revenues and potentially profits [6].

Price elasticities of demand can also be interpreted as a causal parameter. The elasticity can be used to derive how much more quantities of a good could be potentially sold when the price is, for example, decreased [90]. This refers to a counterfactual model as presented in section 2.1. If a retailer has accurate and valid estimations of the price elasticity of demand, they can use counterfactual reasoning to determine the appropriate price for a product or service [90]. Those counterfactual statements can replace the testing of various different prices in a real market situation and can give indications about how much a company can benefit from, for example, sales to clean up their inventory [90]. In summary, accurate estimates of the price elasticity of demand provide a concise overview of the complex causal system that describes individuals purchasing behaviour [90]. Therefore, the price elasticity of demand can be used as a basis for business decisions which makes the accurate estimation of the price elasticity a valuable endeavour for a company.

As described in section 2.1, one would ideally rely on a randomized control trial where the product prices are randomly adjusted and the change in quantity demanded is observed in a real market situation. This is inefficient when it comes to time and costs as the company would sell the products for non-optimal prices. Besides, the customer experience might suffer from such a randomized experiment [90]. Additionally, one major issue with such an experimental setup is that it needs to run for a long duration in order to have enough products sold for valid inference and to have generalisable results across different seasons [90]. Therefore, companies practically need to use observational data of price and quantities sold which they usually track anyhow. As described already, one needs to deal with confounding when using observational data. There are various potential confounders, for example product quality, which can influence the price as well as the quantity demanded. This relationship is shown in the causal graph in Figure 32.

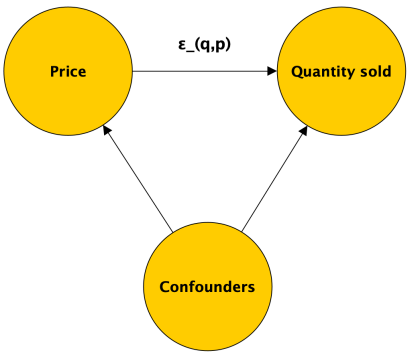


Figure 32: Causal Graph Price Elasticity of Demand
Source: own figure, in the style of [90].

The necessity to control for all relevant and available confounders makes the estimation of the price elasticity of demand one interesting use case for CML. Many potential confounders like quality, novelty, and product features can be obtained using product images and descriptive text as well as product titles. Therefore, the estimation of the price elasticity of demand with observational data provides a use case for DL approaches in the context of CML as described in section 5.

The objective of this case study is to investigate the price elasticity of demand from a given data set using the DML framework and package together with sophisticated DL approaches. Rewriting equation 75, the price (p) elasticity of demand (q) is in general defined as:

$$\epsilon_{q,p} = \frac{\partial q/q}{\partial p/p} \tag{76}$$

Defining the price elasticity of demand as in equation for 76 brings along the convenient feature that $\epsilon_{q,p}$ stays approximately constant when the price p changes [90]. Empirically, it is often a good heuristic that $\epsilon_{q,p}$ is constant in p which the derivatives themselves are not [91]. Therefore, the demand can be locally approximated as an exponential function in p [91]. A general form of a model with constant elasticity can be represented in the following way with a constant [82]:

$$q = constant \cdot p^{\epsilon_{q,p}} \tag{77}$$

As the value of $\epsilon_{q,p}$ can be difficult to estimate due to the partial derivatives, the log-log-regression is one standard way in econometrics to estimate elasticities [90]. The following is obtained by transforming equation 77:

$$\ln q = \epsilon_{q,p} \ln p + \ln constant \tag{78}$$

Equation 78 can therefore be rewritten into the following form where $\epsilon_{q,p}$ is equivalent to

equation 76 [90]:

$$\log q \sim \epsilon_{q,p} \log p + intercept \tag{79}$$

Revisiting and adjusting the PLR model as presented in subsection 2.3.1 results in:

$$\ln q = \ln p\theta_0 + g_0(X) + U, \qquad \mathbb{E}[U|X,p] = 0 \tag{80}$$

$$\ln p = m_0(X) + V, \qquad \mathbb{E}[V|X] = 0 \tag{81}$$

Due to the log-log-transformation (see equation 79), the causal parameter of the PLR model θ_0 can be interpreted in the following as the price elasticity of demand, so that $\theta_0 = \epsilon_{q,p}$ applies. The DML framework with an auxiliary function to obtain orthogonality is used to estimate θ_0. Text data, such as a product description, and image data, such as product images, are used to obtain the confounding covariates X. Using the covariates, one can construct the nuisance functions m_0 and g_0 in order to justify the assumption $\mathbb{E}[U|X,p] = 0$ by controlling for X and therefore also justify the identification of θ_0 [6].

As stated, a PLR model according to equations 80 and 81 can be used with a simple transformation to estimate elasticity. Formally, one can define the residuals in this specific case as [90]:

$$r_p = p - \mathbb{E}[p|X] \tag{82}$$

$$r_q = q - \mathbb{E}[q|X] \tag{83}$$

Both models represented by equations 82 and 83 are trained independently based on some confounding covariates X to partial out the effect of the covariates on p and q [90]. In the DML framework this is done using the nuisance functions m_0 and g_0 which are some ML models. After residualizing, one can construct the estimate of the price elasticity as [90]:

$$\hat{\theta}_0 = \frac{\partial r_q / r_q}{\partial r_p / r_p} \tag{84}$$

The price elasticity in equation 84 can be estimated by exploiting the relationship between equations 82 and 83:

$$\ln r_q \sim \hat{\theta}_0 \cdot \ln r_p + intercept \tag{85}$$

Just as described for the DML framework in section 2.3, one needs to use sample splitting in order to overcome the bias induced by overfitting. Using the DML package brings along the convenient benefit that all formally stated steps in estimating the price elasticity of demand are implemented in the backend and that an easy API can be used to approach the problem setting. Further advantages of using the DML framework and package are that appropriate regularization can be used to select suitable confounding covariates out of a potentially large set of confounders [90]. Secondly, the use of ML models has the advantage that the algorithms are flexible with regards to the functional relationship and potential interactions between confounding covariates [90].

The following process of the case study for products from the toys and games category can also be traced in the corresponding notebooks in Appendix C.6. The created modules and classes are included in the `DMLSim` library and can be traced in Appendix C.9.

6.2 Dataset

The dataset for the case study consists of data from the Amazon sales platform in the category toys and games for subcategories vehicles, cars, and trucks among others. The Amazon locale for the data is the United States of America (amazon.com). This data was collected using the Keepa software tool. Keepa is a price tracker for articles on Amazon and Amazon Marketplace. One can use Keepa to track products and to export product lists that contain prices as well as product details like reviews, sales ranks, and product descriptions.

The used data set contains variables of different data types such as the product name, the product category, the product description as text data and the price of the product and its sales ranking as numeric values. The dataset contains a total of 107 columns and 9988 rows and includes tabular, image and text data. For reference, the raw data set is available in Appendix D.3. The data date of the given data set is not known to the authors but the data is assumed to be fairly recent.

Looking at the relevant columns within the dataset, the features presented in Table 17 emerge.

Table 17: Used Features from Keepa Data

Source: own table.

Keepa variable name	**Description**	**Feature in PLR model**
`New: 30 days avg.`	Price of the product "as an integer of the respective Amazon locale's smallest currency unit" as the historical weighted mean for the last 30 days prior to the data request in $ [47]	Treatment variable p
`Sales Rank: 30 days avg.`	Sales rank of the product as the historical weighted mean for the last 30 days prior to the data request	Outcome variable q
`Title`, `Description & Features`, `Image`, etc.	Text- and image-data, detailed shown in Table 18	Confounding covariates X

Unfortunately, the given data set does not provide direct values for the number of units sold since Keepa does not provide sales estimates or has access to past sales numbers [50].

Instead, the sales rank within the product category can be used as an indicator of the quantity demanded. The sales rank is said to be the best factor for predicting sales [100]. The sales rank can be obtained in the product description and is based on Amazon sales [3]. It is updated regularly to reflect current and historical sales [3]. The sales rank is referring to the root category or top category of the product [48]. In other words, the sales rank is based on the sales of a certain product compared to other products in the same category [100]. The sales rank of a product also may differ in different Amazon marketplaces or locale [100]. The root category for the given record is toys and games and the Amazon locale is the United States of America (amazon.com) to which the sales rank refers. The lowest possible sales rank is 1 which is assigned to the best-selling product in the respective category. Since a higher sales rank indicates fewer sales, the sales rank is negatively proportional to the sales.

It is to be mentioned that there are several pay-to-use applications that estimate the average sales per month based on the sales rank and the category of a respective product (for example [37], [45], [100]). Since those applications are not transparent with their estimation methods, these sales estimators are not considered for the present case study. Because one of the drawbacks of the sales rank is that it is very fluctuating [100], the 30 days average is used (see Table 17).

To examine the given data, the negative sales rank over the price is plotted in Figure 33.

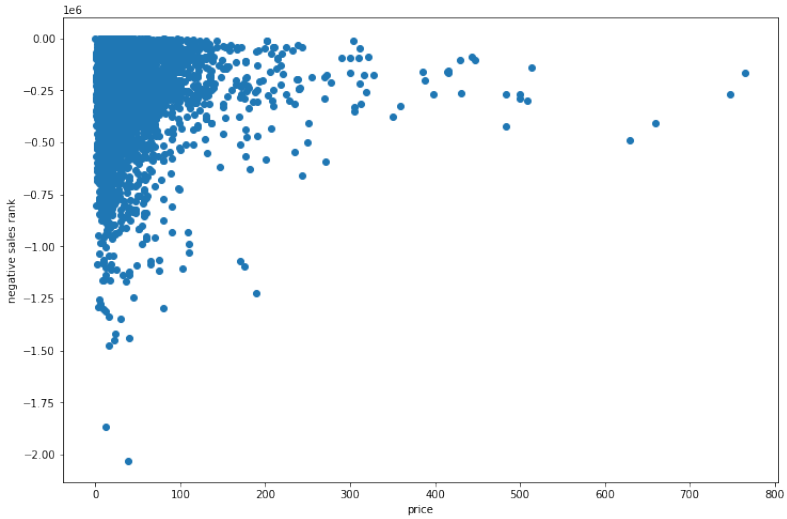


Figure 33: Negative Sales Rank over Price

Source: own figure.

Figure 34 shows the same data as Figure 33 after logarithmic transformation as described by equation 78.

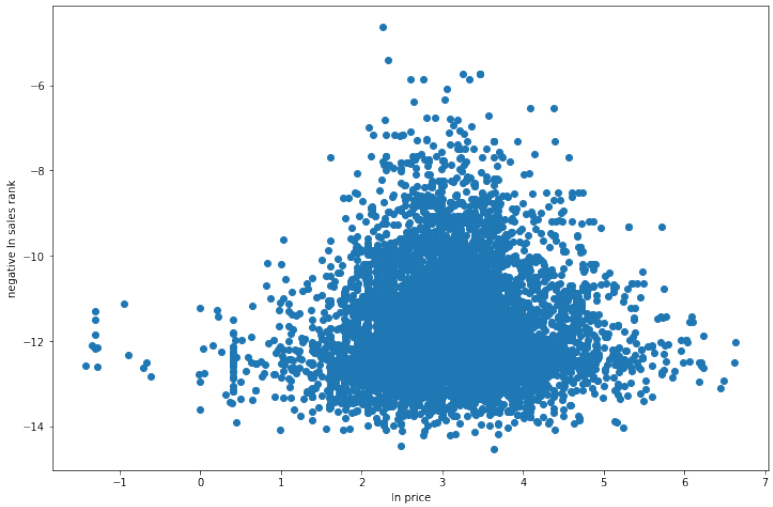


Figure 34: Logarithmic Negative Sales Rank over logarithmic Price

Source: own figure.

Additionally, Figure 35 presents the relationship between $\ln p$ and $-\ln q$ as a binned line plot with 15 bins as a visual form of de-noising.

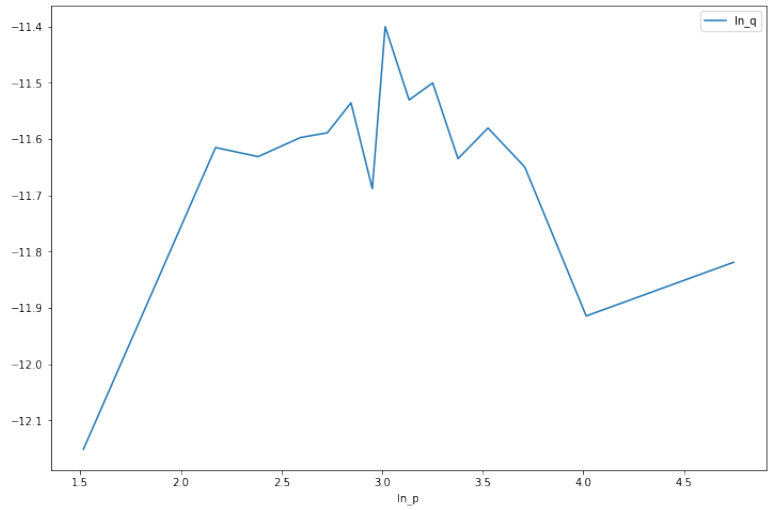
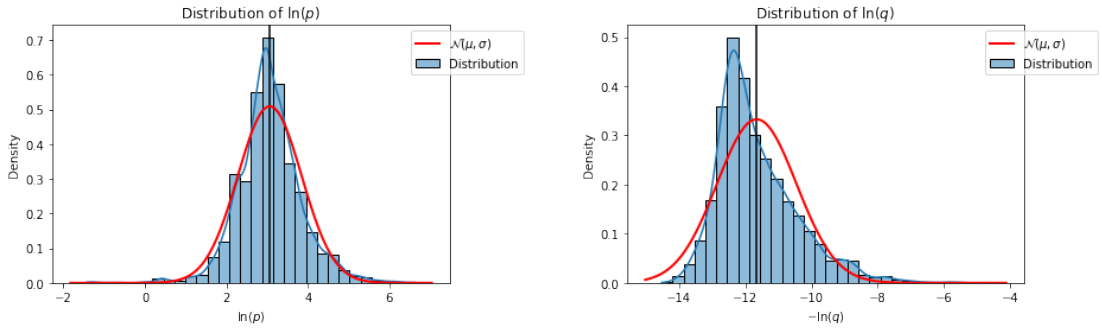


Figure 35: Logarithmic Negative Sales Rank over logarithmic Price as binned Plot

Source: own figure.

Besides, Figure 36 shows the distribution of the numeric data $\ln p$ and $\ln q$. While the distribution of the logarithmic price seems to correspond to the normal distribution, significant deviations from a normal distribution are evident in the negative logarithmic sales rank.



(a) Distribution of $\ln p$ **(b)** Distribution of $-\ln q$

Figure 36: Distribution of Numerical Variables

Source: own figure.

The used text variables for the confounding covariates X are displayed in Table 18. It should be noted that Keepa does not offer a comprehensive glossary for the variables provided, thus allowing for potential interpretation [49]. Additionally, the first product image is used as image data.

Table 18: Text Variables for Confounding Covariates

Source: own table.

Text Data	Description of Text Data
Title	Title of the product
Categories: Sub	Sub-categories of Toys & Games (e.g. race cars)
Description & Features: Description	Product description
Description & Features: Feature 1	Text field for product features
Description & Features: Feature 2	Text field for product features
Description & Features: Feature 3	Text field for product features
Description & Features: Feature 4	Text field for product features
Description & Features: Feature 5	Text field for product features
Manufacturer	Product manufacturing company
Brand	Product brand
Product Group	Product group
Model	Model number
Variation Attributes	Attribute if the product has various options
Color	Product colour
Size	Product size

In order to analyse and use the data in the further course of the case study, the datatypes are adjusted accordingly in the `pd.DataFrame` and the relevant text data (see Table 18) is concatenated into one string variable *text*.

Table 19 and Figure 37 provide an overview of the data by showing the variables for the first four observations. As described in section 6.1, the price p and the quantity demanded q are transformed with a logarithm in order to come up with the PLR

model in equations 80 and 81.

Table 19: First Rows of the transformed Dataset

	text	ln_p	-ln_q
0	Kinsmart Set of 4 McLaren 720s Toy | ...	2.992226	-12.717215
1	Pixar Cars Mack Uncle Lightning...	3.342155	-11.873867
2	2.4G Bluetooth Remote Control...	3.218076	-12.254358
3	Paw Patrol Marshall's Fire...	2.832625	-11.515951



(a) Image of Product 0



(b) Image of Product 1



(c) Image of Product 2



(d) Image of Product 3

Figure 37: Product Images

Source: own figure.

Furthermore, Table 20 presents some descriptive statistics of the numeric variables.

Table 20: Descriptive Statistics for $\ln q$ and $\ln p$
Source: own table.

	ln_p	ln_q
count	9212.000000	9212.000000
mean	3.063645	-11.655680
std	0.782881	1.198627
min	-1.427116	-14.524208
25 %	2.638343	-12.479067
50 %	2.998478	-11.939470
75 %	3.485232	-11.005079
max	6.640359	-4.624973

The descriptive analysis for the toys dataset can be traced in the notebook `Case_Study_Amazon_Toy_Data_Descriptive.ipynb` in Appendix C.6. The descriptive analysis for the chairs data is available in the corresponding notebook located in Appendix C.7.

6.3 Naive Approach

Without considering causality, the relationship between sales and price could also be seen as a coefficient in a linear regression model.

```
data[['ln_p', 'ln_q']].cov()

        ln_p            ln_q
ln_p    0.612903        0.012634
ln_q    0.012634        1.436707
```

In order to get a naive estimation as a regression coefficient, θ is estimated as in Roemheld [91]:

$$\hat{\theta}_{naive} = \frac{cov(p,q)}{var(p)} = \frac{0.012634}{0.612903} = 0.020613 \tag{86}$$

Appendix E.1 contains the proof for equation 86. In this naive approach, where the causal relationship between price and quantity demanded is neglected, the price seems to have only a weak positive correlation with the demand. Since the naively calculated elasticity is positive, the quantity demanded for a product in the category of toys and games would increase with an increasing price following the naive approach.

6.4 First Modelling Approaches

As a first step, the relationship between the covariates X, which contain text (see Table 18) and images, and the transformed outcome (demand) and treatment variable (price) will be analysed. For this purpose, several models based on Transformers as introduced

in sections 2.5 and 2.6 are used to predict the variables of interest from the covariates. Appendix C.5 contains the notebooks that are used for testing the various text and image models in the following subsections.

First, the text data shall be used in a BERT model (see section 2.5) to predict the logarithmic price and quantity demanded. The task to be examined is therefore:

$$\hat{\ln p} \sim f(X, c(X)) \tag{87}$$

$$\hat{\ln q} \sim f(X, c(X)) \tag{88}$$

In equations 87 and 88, $f(X, c(X))$ is a function of the (tokenized) text X and its context $c(X)$ with regard to the BERT architecture. As presented in section 2.5, BERT can represent any token on the basis of its bidirectional context using a pre-trained transformer encoder.

For an easy application of the pre-trained models, the Transformers library of Hugging Faces is used. First, a tokenizer function is defined:

```
from skorch.hf import HuggingfacePretrainedTokenizer

def tokenizer(data, modelname: str):
    data = main["text"]
    hf_tokenizer = HuggingfacePretrainedTokenizer(modelname, train=True)
    hf_tokenizer.fit(data)  # fits new tokenizer on data
    return hf_tokenizer.transform(data)
```

Afterwards, a general class for BERT models is created:

```
from torch import nn
from transformers import AutoModelForSequenceClassification

class BertModule(nn.Module):
    def __init__(self, name, num_labels):
        super().__init__()
        self.name = name
        self.num_labels = num_labels
        self.reset_weights()

    def reset_weights(self):
        self.bert = AutoModelForSequenceClassification.from_pretrained(
            self.name, num_labels=self.num_labels,)

    def forward(self, **kwargs):
        pred = self.bert(**kwargs)
        return pred.logits
```

With the help of these modules, different BERT models can be tested. Exemplary, one can call `'prajjwal1/bert-tiny'` as introduced by Bhargava et al. [14] and Turc et al. [110]:

```
text_token = tokenizer(data, 'prajjwal1/bert-tiny') #Get tokenized text
```

The modified skorch `class` `NeuralNetRegressorDoubleML` from the `dml_nn` module, as presented in section 3.1, (see Appendix C.9) is used to create a skorch model. An R^2 score is added as a callback.

```
from dml_nn import NeuralNetRegressorDoubleML
from skorch.callbacks import EpochScoring
from sklearn.metrics import r2_score

import torch

def r2(net, X, y):
    return r2_score(y, net.predict(X))

bertml1 = NeuralNetRegressorDoubleML(
        BertModule,
        module__name='prajjwal1/bert-tiny',
        module__num_labels=1,
        optimizer=torch.optim.AdamW,
        lr=5e-5,
        max_epochs=25,
        criterion=torch.nn.MSELoss,
        batch_size=32,
        iterator_train__shuffle=True,
        device='cuda' if torch.cuda.is_available() else 'cpu',
        callbacks=[ProgressBar(),
                   EarlyStopping(patience=3),
                   EpochScoring(r2, use_caching=False)])
```

Now the model can be fine-tuned as usual using the sklearn syntax:

```
%%time
bertml1.fit(text_token, ln_p_main)
```

6.4.1 Fine-tuning of BERT

At first, missing values have been removed from the dataset. This results in a reduction of observations from 9988 to 9212. All considered models are trained via Compute Unified Device Architecture (CUDA) on a 40 GB NVIDIA A100 Graphics Processing Unit (GPU). The hyperparameters (i.e. learning rate, epochs, and early stop settings)

were chosen individually for the different models after a short preliminary study in terms of computing time.

For the presented case study, three different BERT models are considered. Those models and their respective references as well as their sizes are presented in Table 21.

Table 21: BERT Models

Source: own table.

BERT model	Number of Parameters	Reference
`'prajjwal1/bert-tiny'`	14.5 millions	Bhargava et al. [14], Turc et al. [110] and Jiao et al. [44]
`'bert-base-uncased'`	110 millions	Devlin et al. [21]
`'distilbert-base-uncased'`	60 Millions	Sanh et al. [96]

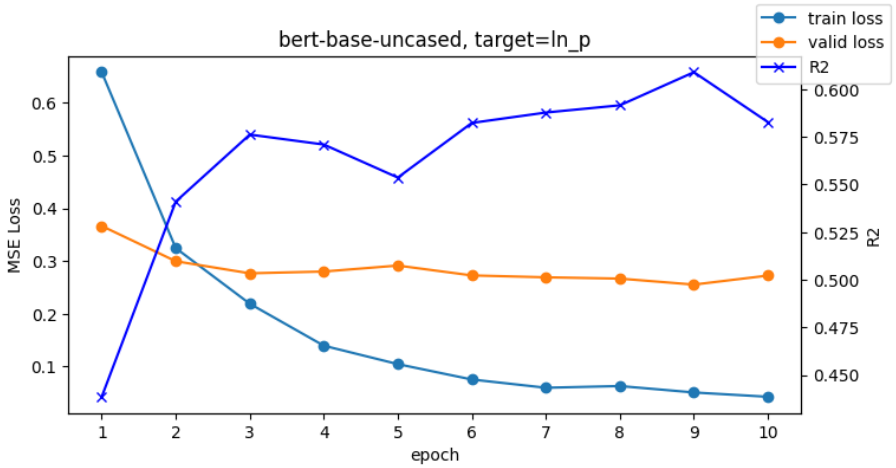
The performance metrics and duration after fine-tuning are presented in Appendix A.6 for bert-tiny, in Appendix A.7 for distilbert-base-uncased, and in Figure 38 for bert-base-uncased. The resulting R^2 scores on the holdout sample are summarized in Table 22.

Table 22: R^2 Score on Holdout Sample for BERT Models

Source: own table.

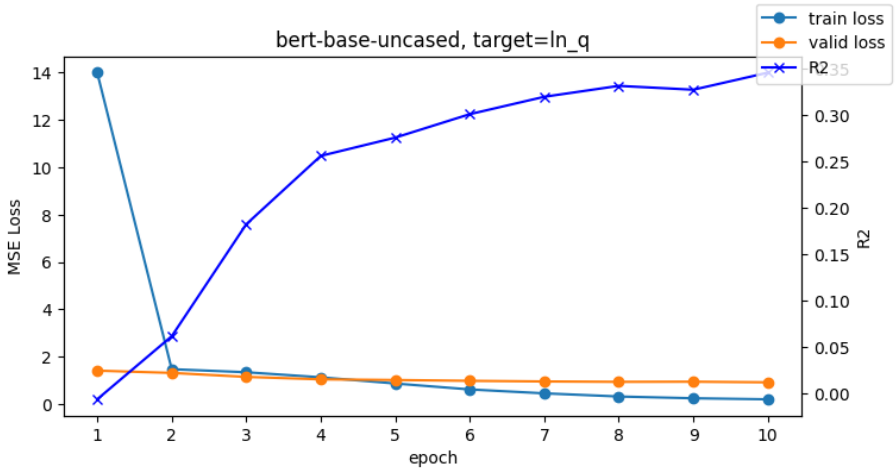
	bert-tiny	bert-base-uncased	distilbert-base-uncased
$\ln p$	0.433623	0.559022	0.534407
$\ln q$	0.192411	0.348586	0.352162

The model bert-base-uncased shows the highest R^2 score for predicting the price and a R^2 score among the best models for predicting the quantity demanded (see Table 22). On the other hand, the computing time for bert-base-uncased is higher compared to distilbert-base-uncased. The model bert-tiny has a very low computing time but also the resulting R^2 score and thus the performance is low. Despite the higher computing time, the performance of bert-base-uncased led to this BERT model being selected for the further analysis in the presented case study.



(a) bert-base-uncased on ln p
CPU times: user 25min 59s, sys: 9.25 s, total: 26min 8s

Wall time: 26min 28s



(b) bert-base-uncased on ln q
CPU times: user 23min 28s, sys: 10.3 s, total: 23min 39s

Wall time: 23min 53s

Figure 38: Performance Metrics and Duration bert-base-uncased

Source: own figure.

6.4.2 Fine-tuning of BEiT

For predicting the price and quantity demanded based on the product pictures, it shall be mainly focused on BEiT models as presented in section 2.6. In addition to the concept of BEiT, the image data shall also be processed with a ViT. ViTs are included in the BEiT model and present another approach to follow the original concept of Transformers as proposed by Vaswani et al. [113] for image recognition [23]. The concept of ViT is closely related to BERT with regard to the tokenization [23]. There are various pre-trained ViT models with different model sizes available which attained excellent results compared to state-of-the-art CNNs [23].

Just as in subsection 6.4.1, the Transformer library of Hugging Face is used. The BEiT and ViT models are trained in the same way as the BERT models in subsection 6.4.1. The dataset is previously filtered for valid images, which reduces the observations from 9212 to 9207. For the presented case study, three different BEiT models are considered. Those models and their respective references as well as their sizes are presented in Table 23.

Table 23: BEiT Models

Source: own table.

BEiT model	**Number of Parameters**	**Reference**
`'microsoft/beit-base-patch16-224-pt22k-ft22k'`	86 millions	Bao et al. [10]
`'microsoft/beit-large-patch16-224-pt22k-ft22k'`	307 millions	Bao et al. [10]
`'lysandre/tiny-vit-random'`	unknown	Lysandre [5]

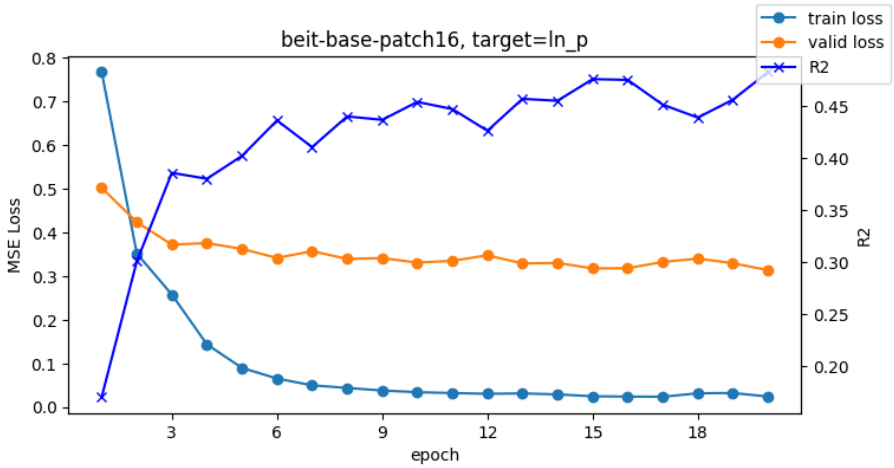
The performance metrics and duration after fine-tuning are presented in Appendix A.8 for beit-large-patch16, in Appendix A.9 for tiny-vit-random, and in Figure 39 for beit-base-patch16. The resulting R^2 scores on the holdout sample are summarized in Table 24.

Table 24: R^2 Score on Holdout Sample for BEiT Models

Source: own table.

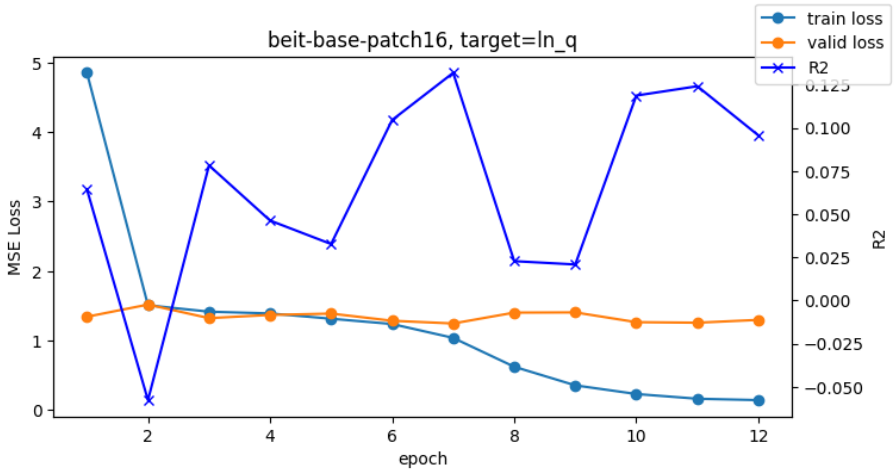
	beit-base-patch16	beit-large-patch16	tiny-vit-random
$\ln p$	0.46067	0.51102	0.03870
$\ln q$	0.08768	0.17371	-0.04967

Table 24 shows that the model tiny-vit-random is not suitable to predict the price or the quantity demanded. The models beit-base-patch16 and beit-large-patch16 show a similar R^2 for predicting the logarithmic price. The R^2 score for predicting the logarithmic demand is significantly lower for the beit-base-patch16 model compared to the beit-large-patch16. In consideration of the computing time, the model beit-base-patch16 shall be used as a BEiT model for further analysis in the presented case study since it shows the best combination of computing time and performance.



(a) beit-base-patch16 on ln p
CPU times: user 15min 1s, sys: 5min 49s, total: 20min 51s

Wall time: 18min 2s



(b) beit-base-patch16 on ln q
Stopping since valid_loss has not improved in the last 5 epochs.
CPU times: user 9min 2s, sys: 2min 45s, total: 11min 47s

Wall time: 10min 29s

Figure 39: Performance Metrics and Duration beit-base-patch16

Source: own figure.

6.5 Multimodal Models

Multimodal models are a class of ML models that can effectively handle input data from different modalities such as text, image, video and audio [86]. These models combine information from multiple modalities to improve their predictive power and achieve a better performance compared to individual modalities systems [86]. Multimodal data fusion seeks to extract and merge contextual information from multiple modalities in order to enhance decision-making [65]. This is done by leveraging the complementary strengths of each modality [65]. Multimodal models can, for example, combine semantic

knowledge gained from texts with knowledge of spatial structures gained from images to learn joint representations of images and texts [76]. In recent years, multimodal models have received considerable attention in various domains, including NLP, computer vision, speech recognition [2], and medicine [65]. This is due to their ability to integrate and utilize complementary and supplementary information in different modalities [65]. The objective of a multimodal model is to combine features from diverse modalities [63]. The architecture of these models can be diverse and includes, for example, neural networks capable of processing and analysing each modality followed by a fusion module that concatenates the information across modalities [76]. The integration of modalities in multimodal models can be achieved at different levels: input level (early fusion), intermediate representation level (middle fusion), and prediction level (late fusion) [121]. Each potential fusion level has its advantages and disadvantages and the choice of fusion level depends on the specific application and the characteristics of the data [65].

The focus of this case study is on multimodal models due to their promising results across different tasks. Utilizing multimodal data fusion is particularly suited for achieving the objective of estimating the price elasticity of demand by incorporating confounders sourced from both, text and images.

6.5.1 Identify Opportunities

In the following, the used multimodal model shall be referred to as Transformer Ensemble since it combines the two Transformer-based models BERT and BEiT to use text and image data. The high-level architecture of this model is shown in Figure 40. The integration of the two modalities occurs through a middle fusion approach, whereby the text and image data are combined at an intermediate representation level, utilizing the `pooler_output`. The `pooler_output` refers to the final hidden state of the first token in the input sequence, typically used for classification tasks [42]. This hidden state is processed by a linear layer and a *tanh* activation function, both of which are trained specifically for the next sentence prediction (classification) objective during pre-training [42]. Finally, the `pooler_output` provides a representation of the input sequence that can be used to make predictions or classify the sequence [42].

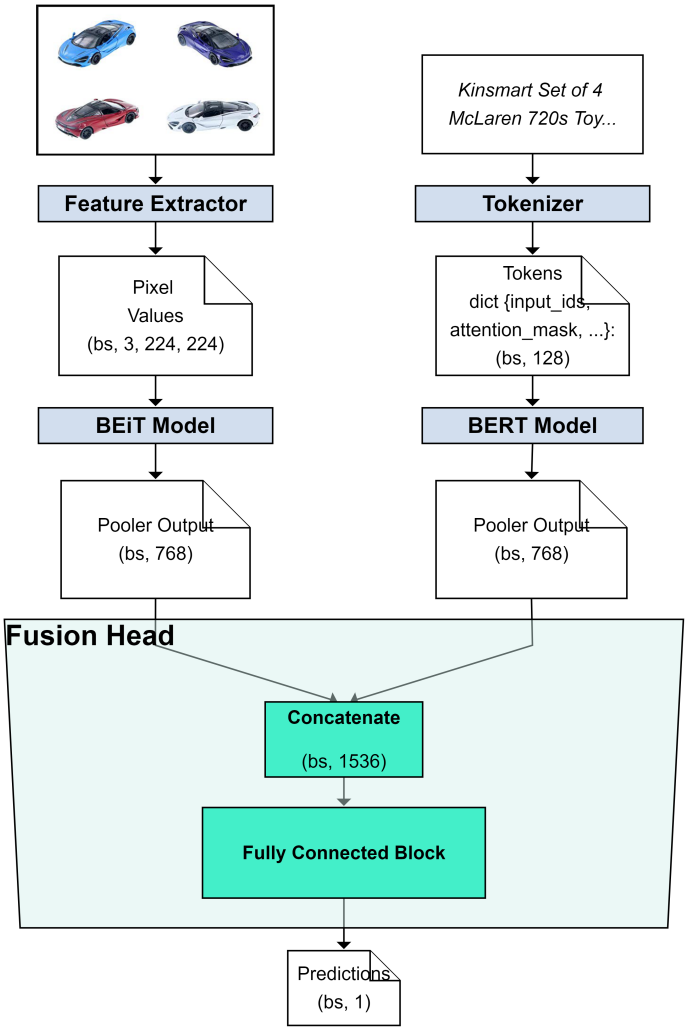


Figure 40: High-Level Multimodal Model Architecture

Source: own figure.

Either the entire multimodal model can be fine-tuned or already fine-tuned networks can be used and only the fusion head needs to be trained. It is decided that the training of the whole network should include, where appropriate, interactions between image and text features. Python classes are created for both purposes: `class FineTuned_TransformerEnsemble` for training the entire network and `class TransformerEnsemble_from_pretrained` for using the pre-trained models. Both classes use the `pooler_output` from the Huggingface Transformer models.

For an easy use of the corresponding classes, the `DMLSim` library developed in section 4.3 is extended by the package `dml_emb`.

```
import torch
from torch import nn
```

```
class FineTuned_TransformerEnsemble(nn.Module):
    def __init__(self, image_model, text_model, num_labels):
        super().__init__()
        self.image_model = BeitForImageClassification.from_pretrained(
            image_model, num_labels=num_labels,
            ignore_mismatched_sizes=True)
        self.text_model = BertForSequenceClassification.from_pretrained(
            text_model, num_labels=num_labels,)
        self.dropout = nn.Dropout(0.1)
        self.relu = nn.ReLU()
        self.pre_classifier = nn.LazyLinear(768)
        self.feature_out = nn.LazyLinear(128)
        self.classifier = nn.LazyLinear(1)

    def forward(self, input_ids, attention_mask, features):
        text_features = self.text_model.bert(input_ids, attention_mask)
        image_features = self.image_model.beit(features)
        output_text = text_features[1]
        output_image = image_features[1]
        output_cat = torch.cat([output_text,
                                output_image],
                                dim=1)
        output_cat = self.pre_classifier(output_cat)
        output_cat = self.relu(output_cat)
        pooled_output = self.dropout(output_cat)
        features = self.feature_out(pooled_output)
        logits = self.classifier(features)
        return logits, features
```

The class architecture exhibits a high degree of modularity and flexibility, enabling users to easily switch between various pre-trained transformer models for both text and images and to adjust the number of output labels for different classification tasks. The class has also been designed to handle variable-length input sequences and varying image sizes. In comparison to `class FineTuned_TransformerEnsemble` another approach is presented in `class TransformerEnsemble_from_pretrained` (see Appendix C.4) which only trains the model head according to Figure 40.

Both classes are part of the `dml_emb.TransformerEnsemble` module (see Appendix C.9). For both approaches, the saved models can be loaded with `pickle`.

6.5.2 Transformer Ensemble Fine-tuning

As described in subsection 4.4.4, the Hummel HPC cluster is used to fine-tune the Transformer Ensemble. The following Python code is presented as an example. The correct execution of the script requires an installation of the `DMLSim` library (see Appendix C.9) including the `dml_emb` package.

```
# -*- coding: utf-8 -*-

from dml_emb.FeatureRegressor import NeuralNetRegressorDoubleOut
from dml_emb.TransformerEnsemble import FineTuned_TransformerEnsemble

from skorch.callbacks import EpochScoring, ProgressBar, EarlyStopping
from sklearn.metrics import r2_score

print('model training on keepa toys. p_toys_feat_fc')

path = r'/work/fwwx329/CaseStudy'
IMG = 'microsoft/beit-base-patch16-224-pt22k-ft22k'
TXT = 'bert-base-uncased'

with open(path + r"/maindata_80.pkl", 'rb') as f:
    main = pickle.load(f)

ln_p_main = main["ln_p"].to_numpy().reshape(-1,1).astype(np.float32)
ln_q_main = main["ln_q"].to_numpy().reshape(-1,1).astype(np.float32)

def feature_extractor(data , modelname: str ):
  hf_extractor = BeitImageProcessor.from_pretrained(modelname)
  return hf_extractor(list(data['img_get']), return_tensors="pt")

def tokenizer(data , modelname: str ):
  txt = data["text"]
  hf_tokenizer = HuggingfacePretrainedTokenizer(modelname, max_length=128)
  hf_tokenizer.fit(txt)
  return hf_tokenizer.transform(txt)

features = feature_extractor(main, IMG)

text_token = tokenizer(main, TXT)

module_p = FineTuned_TransformerEnsemble(image_model=IMG,
                                         text_model=TXT,
                                         num_labels=1)

inputs = {'input_ids' : text_token['input_ids'],
          'attention_mask' : text_token['attention_mask'],
          'features' : features['pixel_values'],}

def r2(net, X, y):
    return r2_score(y, net.predict(X))
```

```
model_p = NeuralNetRegressorDoubleOut(
    module_p,
    criterion=torch.nn.MSELoss,
    optimizer=torch.optim.AdamW,
    #optimizer__amsgrad=True,
    lr=3e-5,
    max_epochs=50,
    batch_size=16, #try 16
    iterator_train__shuffle=True,
    device='cuda' if torch.cuda.is_available() else 'cpu',
    callbacks=[#ProgressBar(),
               EpochScoring(r2, use_caching=False, lower_is_better=False),
               EarlyStopping(patience=3, threshold=0.001,
                             threshold_mode='rel', lower_is_better=True,
                             load_best=True)]
)

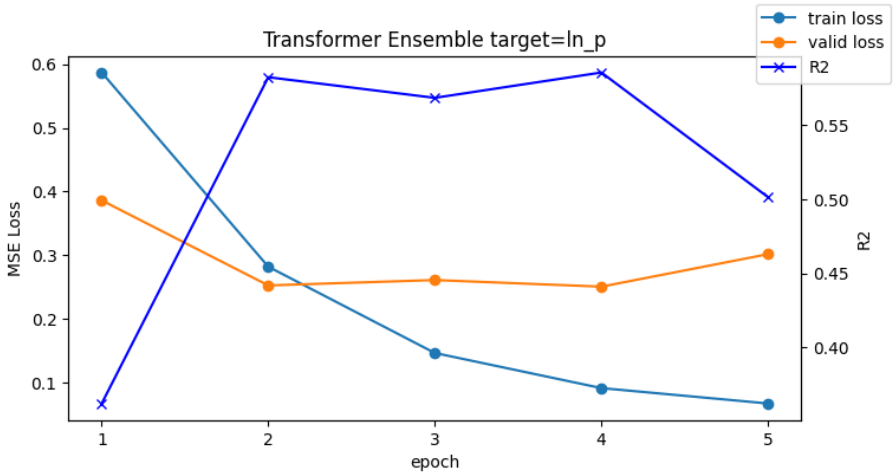
model_p.fit(inputs, ln_p_main)

with open(path + '/model_p_toys_transformer_ensemble.pkl', 'wb') as f:
    pickle.dump(model_p, f)
```

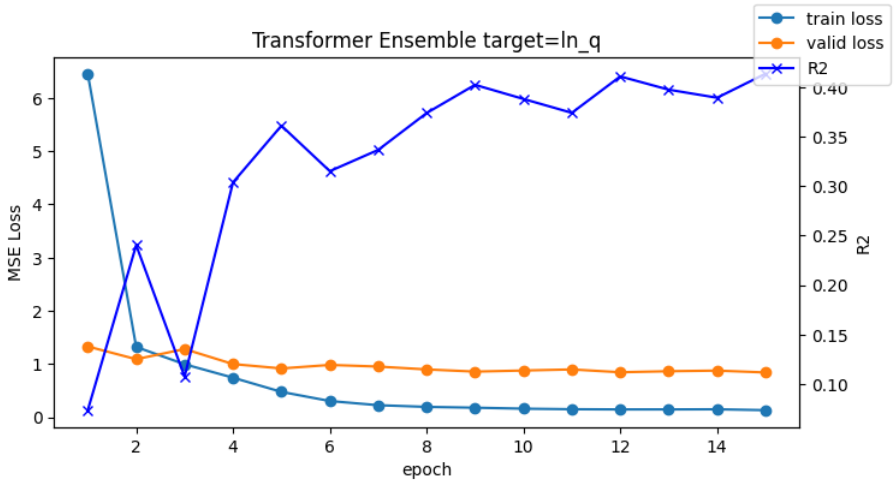
An example of execution on a GPU node on the Hummel HPC cluster is demonstrated in the following bash scrip.

```
#!/bin/bash
#SBATCH --job-name=p_toys_feat
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=10:00:00
#SBATCH --export=NONE
set -e
srun="srun -n1 -N1 --exclusive"
source /sw/batch/init.sh
module unload env
module load python/3.9.12
module list
pwd
date
cd /work/fwwx329/CaseStudy
export TRANSFORMERS_CACHE=/work/fwwx329/CaseStudy/transformer_cache
$srun python train_toys_p_model.py
echo "Node $(hostname -s)"
echo "Current TMPDIR: $TMPDIR"
exit
```

Figure 41 shows the training process of the Transformer Ensembles.



(a) Transformer Ensemble on ln p

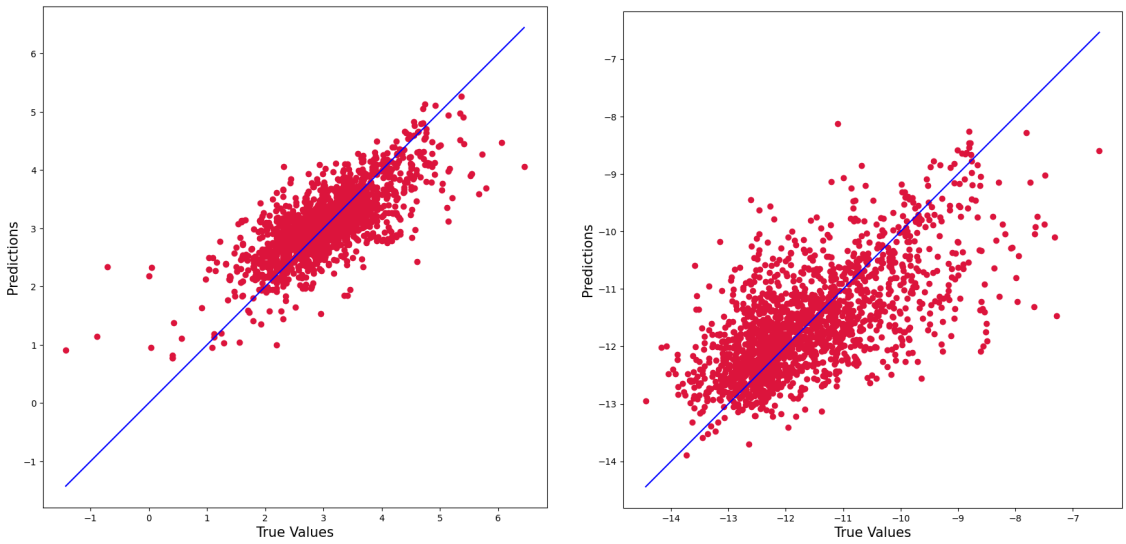


(b) Transformer Ensemble on ln q

Figure 41: Performance Metrics and Duration Multimodal Model

Source: own figure.

The performance of the Transformer Ensemble on the holdout data set is shown in Figure 42 and Table 25. It is obvious that utilizing a Transformer Ensemble model, which combines text and image modalities, results in higher R^2 scores compared to the individual BERT (see Table 22) and BEiT (see Table 24) models. This indicates that the combination of the two modalities is superior in terms of performance and therefore one suitable approach for the presented case study.



(a) true vs. predicted values on $\ln p$ **(b)** true vs. predicted values on $\ln q$

Figure 42: Performance of Multimodal Model Ensemble true vs. predicted Values
Source: own figure.

Table 25: R^2 Score on Holdout Sample for Multimodal Model
Source: own table.

	Transformer Ensemble
$\ln p$	0.5663
$\ln q$	0.4198

6.6 Causal Inference

6.6.1 Methods and Approaches

Having successfully trained and developed models that make a visible explanatory contribution to the distribution of output $-\ln q$ and treatment $\ln p$, it is possible to consider their use in the context of causal inference. There are different approaches which have obvious advantages and disadvantages. The first possible solution is to use the fusion models described in section 6.5 directly in the DML package as learners for the nuisance functions `ml_m` and `ml_l`. Using the `class NeuralNetRegressorDoubleML` described and created in section 3 (see Appendix C.9), these models could be wrapped in a compatible instance according to the DML framework. However, the unstructured data X creates a problem. The data backend of the DML package is designed for structured data and expects a 2-dimensional data structure $\mathbb{R}^{N\times M}$, i.e. a matrix, as input for X. The text (as input ids $\mathbb{R}^{128}$ and attention mask $\mathbb{R}^{128}$) and image data (as pixel values $\mathbb{R}^{3\times 224\times 224}$) are available as tensors after pre-processing. This data must, according to the skorch library, be passed as a dictionary with corresponding keys into the learner's `fit`

method:

```
tokens = tokenizer(main, TOKENIZER)
features = feature_extractor_beit(main, FEATURE_EXTRACTOR)
inputs = {'input_ids' : tokens['input_ids'],
          'attention_mask' : tokens['attention_mask'],
          'features' : features['pixel_values'],}

print(inputs['input_ids'].shape,
      inputs['attention_mask'].shape,
      inputs['features'].shape)
```

```
torch.Size([n_samples, 128])
torch.Size([n_samples, 128])
torch.Size([n_samples, 3, 224, 224])
```

The multidimensional data structure input does not allow to create a `DoubleMLData` instance. Developing a new class to handle this data could be a possible solution. However, the implementation would take too much time and effort to develop within the scope of this paper.

As another possible solution, the fusion models could generate embeddings which in turn can be used as the data basis for the `DoubleMLData` instance. This approach follows the principle of supervised representation learning or supervised feature learning. Transformer models are capable of conducting feature learning, as they acquire a comprehensive representation of their input through hidden layer(s) which is then utilized for classification or regression at the output layer. Figure 43 shows the process of the representation learning approach.

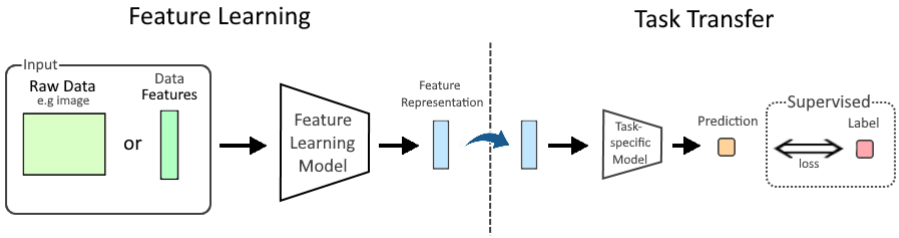


Figure 43: Feature / Representation Learning Process

Source: [27]

The fully connected block in Figure 40 (coloured in green) refers to the task-specific model in Figure 43 and the BERT and BEiT model to the feature learning model in Figure 43. With respect to the DML framework, the task-specific model can be a ML model used to learn the nuisance functions. The generated feature representations/ embeddings can be

used as covariates X. The Transformer models used for text and image processing are similar in their structure, their corresponding APIs and their outputs. The `pooler_outputs` in $\mathbb{R}^{768}$ of the models can be used to generate the embeddings. The selected solution option for the subsequent procedure is depicted in Figure 44.

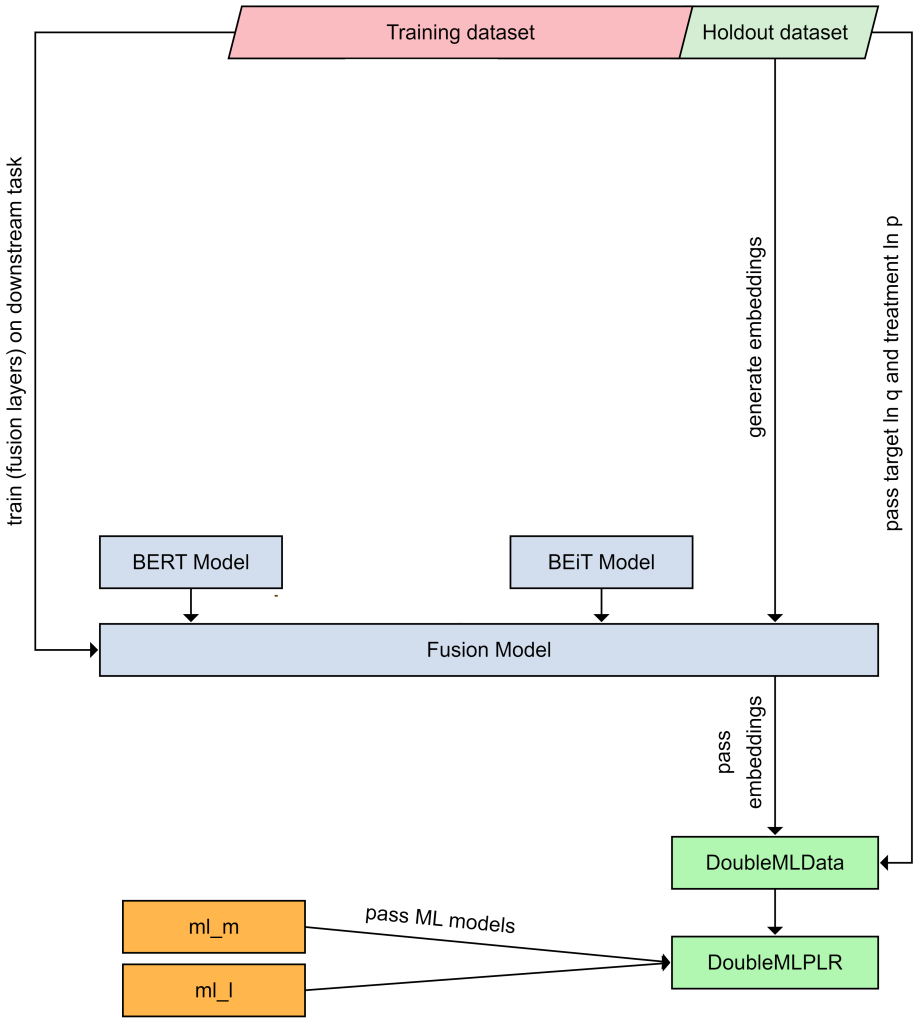


Figure 44: High-Level Overview of the Usage of Embeddings in the DoubleML Model

Source: own figure.

For this purpose, the classes in the `TransformerEnsemble` module include a linear layer `feature_out`. The output of this layer can be accessed via the method `predict_features` from the **`class`** `NeuralNetRegressorDoubleOut` in the `FeatureRegressor` module.

```
class NeuralNetRegressorDoubleOut(NeuralNetRegressor):

    def predict_features(self, X):
        y_probas = []
```

```
        for yp in super().forward_iter(X, training=False):
            yp = yp[1] if isinstance(yp, tuple) else yp
            y_probas.append(yp)
        y_proba = np.concatenate(y_probas, 0)
        return y_proba

    def get_loss(self, y_pred, y_true, *args, **kwargs):
        if isinstance(y_pred, tuple):
          logits, features = y_pred
        else:
          logits = y_pred
        return super().get_loss(logits, y_true, *args, **kwargs)
```

This class is necessary because the classes of the `TransformerEnsemble` module return two objects in their forward method. To train the `TransformerEnsemble` models, a loss function must be computed that refers to a uniquely assigned object returned by the forward method. For this reason, the `get_loss` method of the parent class is overridden. To obtain the embeddings, the features of the forward method should be returned, so this has been added to the `predict_features` method. After training a `TransformerEnsemble` model on the training dataset, the embeddings for the holdout dataset are created, shown below:

```
tokens_h = tokenizer(holdout, TOKENIZER)
features_h = feature_extractor_beit(holdout, FEATURE_EXTRACTOR)
ln_p_h = holdout["ln_p"].to_numpy().reshape(-1,1).astype(np.float32)
ln_q_h = holdout["ln_q"].to_numpy().reshape(-1,1).astype(np.float32)

inputs_h= {'input_ids' : tokens_h['input_ids'],
           'attention_mask' : tokens_h['attention_mask'],
           'features' : features_h['pixel_values']}

p_feat_h = model_p.predict_features(inputs_h) #(n_samples, 128)
q_feat_h = model_q.predict_features(inputs_h) #(n_samples, 128)

pq_feat_h = np.concatenate((p_feat_h, q_feat_h), axis = 1)
#(n_samples, 256)
```

The `TransformerEnsemble` model can be used as a feature learning model according to Figure 43.

6.6.2 First Analysis

With the text- and image information embedded in $\mathbb{R}^{256}$ as structured data, any ML model, which uses the embeddings as input, can be trained. They can be loaded into an instance of the `class DoubleMLData` and different learners can be used for the nuisance functions. An example of neural networks as learners is shown in the code below. This section describes and discusses the approach and methods using a simple split into

training and holdout datasets. The training dataset is used to train the Transformer Ensemble model. Only the holdout data is used as data in the `DoubleMLData` backend. The Transformer Ensemble model generates embeddings from the holdout dataset, which are considered as confounders. In this case, the data set was divided into 80 % training data (7366 observations) and 20 % holdout data (1841 observations).

```
from doubleml import DoubleMLData, DoubleMLPLR
from dml_nn.simulation_learner import RegressorModel
from dml_nn.doubleml_skorch_api import NeuralNetRegressorDoubleML
from sklearn.base import clone

dml_obj_pq = DoubleMLData.from_arrays(x = pq_feat_h,
                                      y = ln_q_h.reshape(-1),
                                      d = ln_p_h.reshape(-1))

neural_net = RegressorModel([256, 128, 64, 32], dropout=0.1)
nn_sk = NeuralNetRegressorDoubleML(neural_net,
                                   criterion=torch.nn.MSELoss,
                                   optimizer=torch.optim.AdamW,
                                   #optimizer__amsgrad=True,
                                   lr=4e-5,
                                   max_epochs=20,
                                   batch_size=4,
                                   verbose=0,
                                   iterator_train__shuffle=False,)

ml_m = nn_sk
ml_l = nn_sk

dml_plr_pq_nn = DoubleMLPLR(dml_obj_pq,
                            ml_l = clone(ml_l),
                            ml_m = clone(ml_m))
)
```

```
print(dml_plr_pq_nn.fit())

================== DoubleMLPLR Object ==================

------------------ Data summary      ------------------
Outcome variable: y
Treatment variable(s): ['d']
Covariates: ['X1', 'X2', 'X3', 'X4', 'X5', ...]
Instrument variable(s): None
No. Observations: 1841
```

```
------------------ Score & algorithm ------------------
Score function: partialling out
DML algorithm: dml2

------------------ Machine learner   ------------------
Learner ml_l: <dml_nn.doubleml_skorch_api.NeuralNetRegressorDoubleML...>
Learner ml_m: <dml_nn.doubleml_skorch_api.NeuralNetRegressorDoubleML...>
Out-of-sample Performance:
Learner ml_l RMSE: [[0.9470346]]
Learner ml_m RMSE: [[0.51221246]]

------------------ Resampling        ------------------
No. folds: 5
No. repeated sample splits: 1
Apply cross-fitting: True

------------------ Fit summary       ------------------
       coef   std err         t          P>|t|     2.5 %    97.5 %
d -0.217382  0.044351 -4.901352  9.517938e-07 -0.304309 -0.130455
```

In the following, instances of the `DoubleMLPLR` model were created and evaluated for different learners like Light Gradient-Boosting Machine (LGBM), Lasso, neural networks, Random Forest and an OLS regression. Table 26 shows the summary for different learners.

Table 26: Summary of DoubleMLPLR models with different learners

Source: own table.

	coef	std err	t	P>|t|	2.5 %	97.5 %
LGBM	-0.223776	0.043120	-5.189573	2.107767e-07	-0.308290	-0.139262
Lasso	-0.224109	0.042518	-5.270976	1.357002e-07	-0.307442	-0.140776
Linear reg	-0.214653	0.042703	-5.026635	4.991613e-07	-0.298349	-0.130956
Neural net	-0.217382	0.044351	-4.901352	9.517938e-07	-0.304309	-0.130455
Random Forest	-0.227581	0.043426	-5.240677	1.599888e-07	-0.312694	-0.142468

Figure 45 shows the estimated coefficients θ, i.e. the price elasticity of demand, including the 95 % CI.

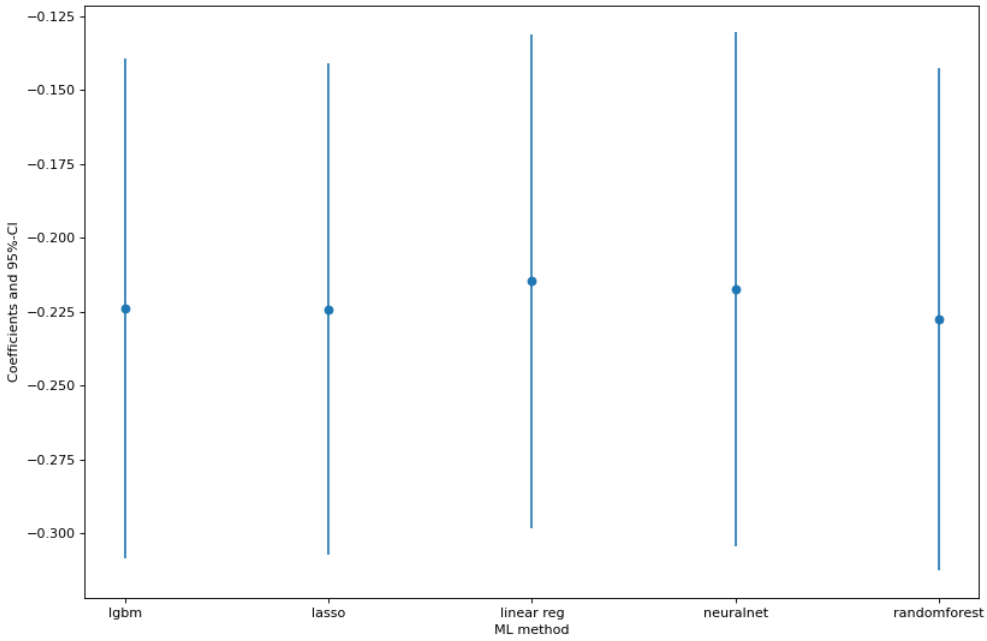


Figure 45: Estimated Coefficient and Confidence Intervals for first Analysis

Source: own figure.

Additionally, Table 27 and Figure 46 show the RMSE values of the used learners.

Table 27: Root Mean Squared Error values for considered learners in first Analysis

Source: own table.

	ml_m	ml_l
LGBM	0.515034	0.921944
Lasso	0.503499	0.892898
Linear reg	0.539271	0.964768
Neural net	0.512212	0.947035
Random Forest	0.503724	0.910609

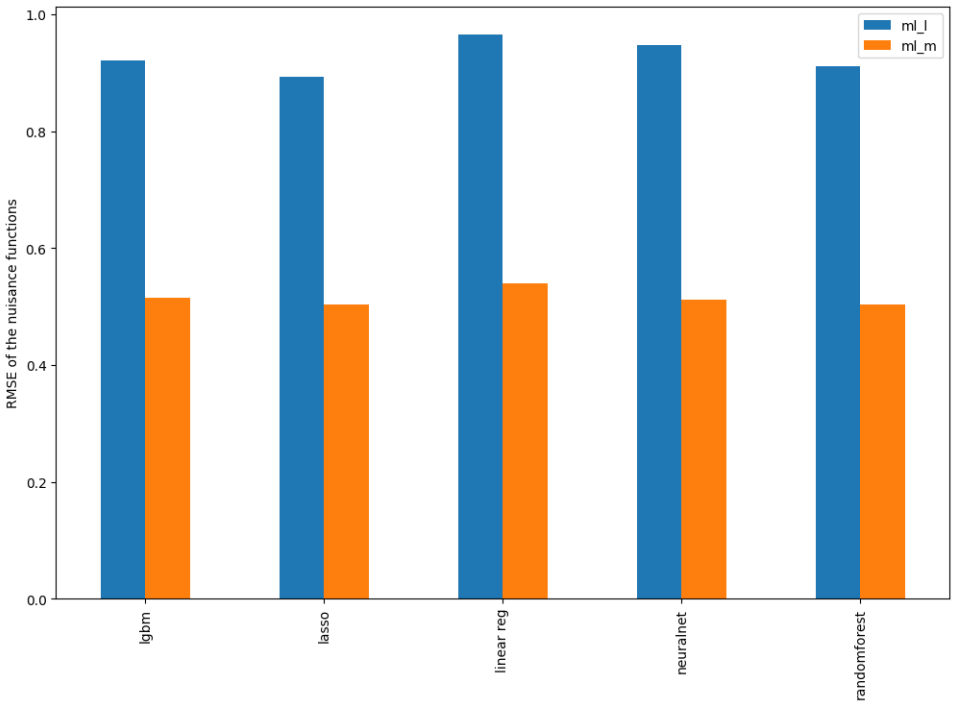


Figure 46: Root Mean Squared Error values of the used Learners in the Causal Model for first Analysis

Source: own figure.

It is apparent from the models that the estimated price elasticity of demand is approximately -0.2, suggesting that the demand is inelastic, i.e., the quantity demanded declines as the price increases. The RMSE values are comparable for all models. It is obvious that the RMSE for the nuisance function `ml_l` (quantity) are significantly larger than for `ml_m` (price) for each considered learner.

The similar RMSEs and estimated causal parameters among different learners indicate that the generated embeddings are highly effective. This effectiveness ensures that the used learners for `ml_l` and `ml_m` only need to capture an approximately linear relationship between the confounders and the treatment as well as the outcome variable. Thus, a basic OLS regression can adequately capture this relationship and provide an accurate causal parameter estimate.

The first analysis for the toys dataset can be found in the notebook `Case_Study_Amazon_Toy_Data_(First).ipynb` in Appendix C.6. The first analysis for the chairs data is available in the corresponding notebook located in Appendix C.7.

Discussion of first Results

In this case study, the authors generally aimed to estimate the price elasticity of demand using observational data and the DML approach. For the first analysis, the relationship between price and sales in the holdout dataset is briefly visualised. Figure 47 shows the

(confusing) relationship between $\ln p$ and $\ln q$ as a line graph with values clustered to de-noise the plot.

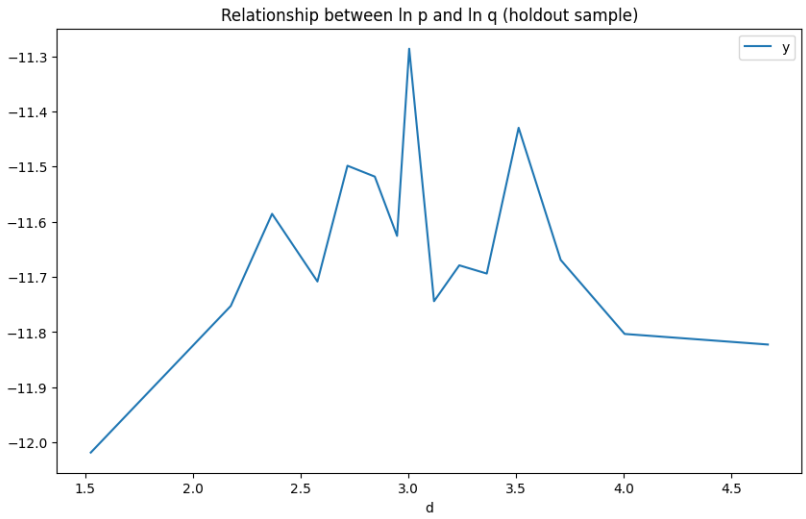


Figure 47: Relationship between Output and Treatment Variable for first Analysis

Source: own figure.

To compare the biased relationship to the unbiased relationship after residualizing the output and treatment variable, the values are mean-centred and plotted again (see Fig. 48). The slope of the graph at a point corresponds approximately to the naive estimated price elasticity. A trend line is added for a better representation of the (aggregated) price elasticity.

```
df = dml_obj_pq.data

df['d_ln_p'] = df['d'] - df['d'].mean()
df['d_ln_q'] = df['y'] - df['y'].mean()
```

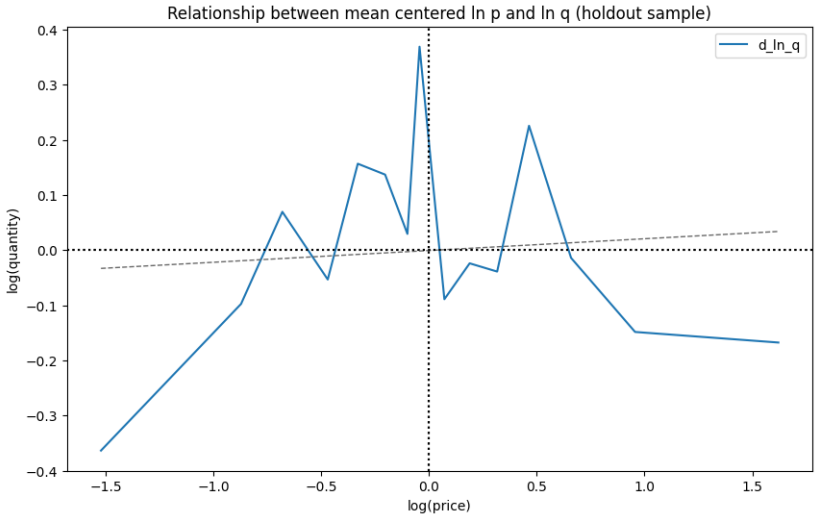


Figure 48: Relationship between Output and Treatment Variable after mean-centring for first Analysis

Source: own figure.

As already shown in section 6.3, a weak positive correlation can be identified. The slope of the trend line shown in Figure 48 is (approximately, since it represents a regression over the clustered data points) the coefficient of an OLS regression calculated in equation 86:

$$\hat{\theta}_{naive} = 0.020613 \tag{89}$$

In the following, the approach of the PLR model is explained and visualised. After fitting a model for $\ln p$ on X and $\ln q$ on X according to the DML framework, the residuals of the output and treatment variable are calculated:

$$r_p = \ln p - \hat{\ln p} \tag{90}$$

$$r_q = \ln q - \hat{\ln q} \tag{91}$$

Afterwards, an OLS Model is built:

$$r_q = constant + \theta\, r_p + \epsilon \tag{92}$$

In equation 92, θ is representing the estimated price elasticity of demand. By controlling for the confounding variables X in the separate regression models of $\ln p$ and $\ln q$, the effect of changes in price on quantity demanded can be isolated. The residuals r_p and r_q solely capture the variation in price and quantity demanded that is not explained by X. Regressing r_q on r_p allows estimating the causal effect of changes in price on quantity demanded.

```
ln_p_hat = dml_plr_pq_lasso.predictions['ml_m'].reshape(-1)
ln_q_hat = dml_plr_pq_lasso.predictions['ml_l'].reshape(-1)
```

```
df['d_ln_p_res'] = df['d'] - ln_p_hat
df['d_ln_q_res'] = df['y'] - ln_q_hat
```

```
ols = sm.OLS(df[d_ln_q_res], sm.add_constant(df['d_ln_p_res']))
res = ols.fit()
res.summary()
```

Table 28: Summary of Ordinary Least Squares Regression for first Analysis
Source: own table.

Dep. Variable:	d_ln_q_res	**R-squared:**	0.016
Model:	OLS	**Adj. R-squared:**	0.015
Method:	Least Squares	**F-statistic:**	29.85
Date:	Fri, 31 Mar 2023	**Prob (F-statistic):**	5.31e-08
Time:	12:44:45	**Log-Likelihood:**	-2388.9
No. Observations:	1841	**AIC:**	4782.
Df Residuals:	1839	**BIC:**	4793.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	-0.0008	0.021	-0.040	0.968	-0.041	0.040
d_ln_p_res	-0.2241	0.041	-5.463	0.000	-0.305	-0.144

It should be noted that the regression coefficient obtained from the OLS regression on the residuals in Table 28 matches with the result obtained from the `DoubleMLPLR` instance that utilizes Lasso as a learner, as shown in Table 26.

To de-noise the data, a binned plot is created again and a trendline, which represents (approximately, since it represents a regression over the clustered data points) the OLS regression in equation 92 , is added (see Fig. 49).

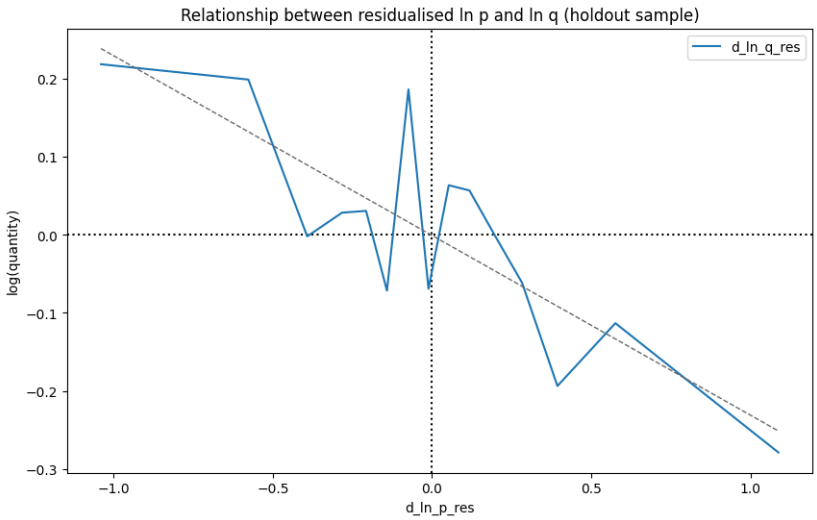


Figure 49: Relationship between Output and Treatment Variable after Residualizing for first Analysis

Source: own figure.

After residualization, a negative correlation is clearly recognisable (see Fig. 49). The centred and residualized relationship are again plotted in Figure 50 to highlight the differently directed correlation.

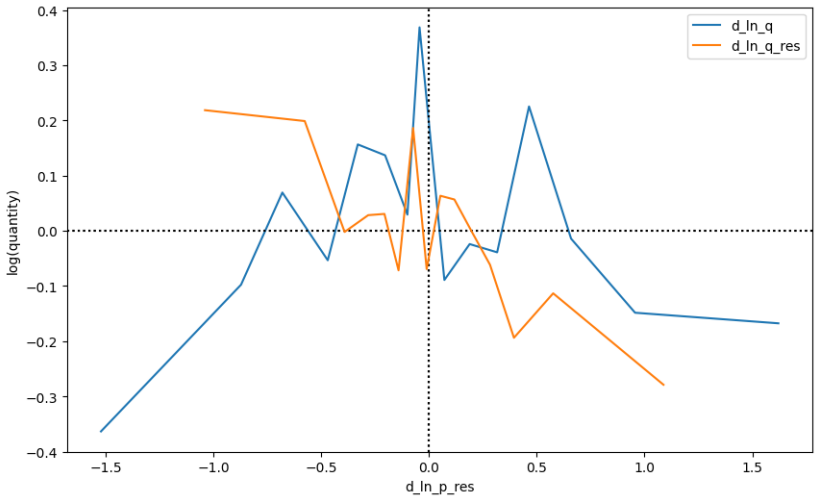


Figure 50: Comparison between centred and residualized Relationships for first Analysis

Source: own figure.

In conclusion, Table 29 shows the estimates of the naive approach and the DML approach (related to Lasso models for the nuisance functions).

Table 29: Price Elasticity of Demand for Naive and Causal Estimation for first Analysis
Source: own table.

	Price Elasticity
Naive Estimate	0.0206
Causal Estimate Lasso	-0.2241

The initial examination of causality with a limited number of observations in the holdout sample produced different directed results for estimating the price elasticity of demand when compared to the naive approach (see Table 29). Moreover, investigating the residuals demonstrated the appropriateness of employing the DML framework for this case study. Ultimately, it is crucial to incorporate confounding factors identified from text and images to obtain accurate and effective causal estimates for determining the price elasticity of demand. To utilize the entire available data set instead of only a portion, the process from the first analysis should be expanded to encompass a full data analysis in the subsequent section.

6.6.3 Full Analysis

In this section, the complete data set with 9207 observations is used to estimate the causal parameter θ, i.e. the price elasticity of demand. For this purpose, as in subsection 6.6.2, Transformer Ensemble models are trained on a training data set and the embeddings are generated from a holdout data set. To avoid data loss, the principle of cross-fitting is used. For this purpose, the **class** `CrossEmbeddings` is implemented in the `CrossEmbeddings` module in the `dml_emb` package.

```
import pandas as pd
import numpy as np
from sklearn.base import clone
from skorch.hf import HuggingfacePretrainedTokenizer
from transformers import BeitImageProcessor
from sklearn.model_selection import KFold

class CrossEmbeddings:
    def __init__(
        self,
        dataset,
        text_col,
        image_col,
        d_col,
        y_col,
        n_folds,
        aux_d,
        aux_y,
        txt_str,
```

```
        img_str,
    ):
        self.dataset = dataset.reset_index()
        self.text_col = text_col
        self.image_col = image_col
        self.d_col = d_col
        self.y_col = y_col
        self.n_folds = n_folds
        self.aux_d = aux_d
        self.aux_y = aux_y
        self.txt_str = txt_str
        self.img_str = img_str

        self.txt_tokens = None
        self.img_features = None
        self.d = None
        self.y = None
        self.emb = {fold: None for fold in range(self.n_folds)}

    def _tokenizer(self, data, modelname: str):
        txt = data[self.text_col]
        hf_tokenizer = HuggingfacePretrainedTokenizer(modelname,
                                                      max_length=128)
        hf_tokenizer.fit(txt)
        return hf_tokenizer.transform(txt)

    def _feature_extractor(self, data, modelname: str):
        hf_extractor = BeitImageProcessor.from_pretrained(modelname)
        return hf_extractor(list(data[self.image_col]),
                            return_tensors="pt")

    def _prepare_data(self):
        self.txt_tokens = self._tokenizer(self.dataset, self.txt_str)
        self.img_features = self._feature_extractor(self.dataset,
                                                    self.img_str)

    def slice_input(self, idx):
        inputs = {
            "input_ids": self.txt_tokens["input_ids"][idx],
            "attention_mask": self.txt_tokens["attention_mask"][idx],
            "features": self.img_features["pixel_values"][idx],
        }
        return inputs

    def fit_and_predict_embeddings(self):
        self._prepare_data()
```

```
        for fold, (train_idx, output_idx) in enumerate(
            KFold(n_splits=self.n_folds,
                  shuffle=True,
                  random_state=1234).split(self.dataset)):
            print(f"Fold {fold+1} from {self.n_folds}")
            aux_y_tmp = clone(self.aux_y)
            aux_d_tmp = clone(self.aux_d)
            y_tmp = (
                self.dataset.loc[train_idx, self.y_col]
                .to_numpy()
                .reshape(-1, 1)
                .astype(np.float32)
            )
            d_tmp = (
                self.dataset.loc[train_idx, self.d_col]
                .to_numpy()
                .reshape(-1, 1)
                .astype(np.float32)
            )
            temp_inputs = self.slice_input(train_idx)
            print(" Fit target aux model")
            aux_y_tmp.fit(temp_inputs, y_tmp)
            y_emd_tmp = aux_y_tmp\
                    .predict_features(self.slice_input(output_idx))
            del aux_y_tmp
            print(" Fit treatment aux model")
            aux_d_tmp.fit(temp_inputs, d_tmp)
            d_emd_tmp = aux_d_tmp\
                    .predict_features(self.slice_input(output_idx))
            del aux_d_tmp
            emb_tmp = np.concatenate((d_emd_tmp, y_emd_tmp), axis=1)
            self.emb[fold] = {idx: emb_tmp[x]
                              for x, idx
                              in enumerate(output_idx)}
        print("Finished")

    def get_embedded_df(self):
        d = {}
        for f in range(len(self.emb)):
            d = {**d, **self.emb[f]}
        self.dataset["emb"] = self.dataset.index.map(d)
        return self.dataset.set_index("index")

    def get_embeddings(self):
        return np.stack(self.dataset["emb"].to_numpy())
```

The **class** `CrossEmbeddings` creates the embeddings according to the cross-fitting principle as described in subsection 2.3.4. In the full data analysis, embeddings for the complete data set with $K = 3$ folds are generated. Therefore, three models each for target and treatment variables are generated and fitted to $\frac{2}{3}$ (6138 observations) of the entire data set. The embeddings are subsequently created from the remaining portion ($\frac{1}{3}$, 3069 observations) of the data set. For this purpose, a GPU node of the Hummel HPC Cluster is used as described in subsection 4.4.4. With the use of the `dml_emb` package, the code needed to generate the embeddings is clearly arranged:

```
from dml_emb.CrossEmbeddings import CrossEmbeddings
from dml_emb.FeatureRegressor import NeuralNetRegressorDoubleOut
from dml_emb.TransformerEnsemble import FineTuned_TransformerEnsemble

print('embedding generation for amazon toy data')

path = r'/work/fwwx329/CaseStudy'
IMG = 'microsoft/beit-base-patch16-224-pt22k-ft22k'
TXT = 'bert-base-uncased'

with open(r"/work/fwwx329/CaseStudy/dataset_toys.pkl", 'rb') as f:
    main = pickle.load(f)

module_q = FineTuned_TransformerEnsemble(image_model=IMG,
                                         text_model=TXT,
                                         num_labels=1)

module_p = FineTuned_TransformerEnsemble(image_model=IMG,
                                         text_model=TXT,
                                         num_labels=1)

def r2(net, X, y):
    return r2_score(y, net.predict(X))

model_q = NeuralNetRegressorDoubleOut(
    module_q,
    criterion=torch.nn.MSELoss,
    optimizer=torch.optim.AdamW,
    #optimizer__amsgrad=True,
    lr=3e-5,
    max_epochs=10,
    batch_size=16, #try 16
    iterator_train__shuffle=True,
    device='cuda:1' if torch.cuda.is_available() else 'cpu',
    callbacks=[#ProgressBar(),
               EpochScoring(r2, use_caching=False, lower_is_better=False),
```

```
                EarlyStopping(patience=3, threshold=0.001,
                              threshold_mode='rel', lower_is_better=True,
                              load_best=True)]
)
model_p = NeuralNetRegressorDoubleOut(
    module_p,
    criterion=torch.nn.MSELoss,
    optimizer=torch.optim.AdamW,
    #optimizer__amsgrad=True,
    lr=3e-5,
    max_epochs=8,
    batch_size=16, #try 16
    iterator_train__shuffle=True,
    device='cuda:0' if torch.cuda.is_available() else 'cpu',
    callbacks=[#ProgressBar(),
                EpochScoring(r2, use_caching=False, lower_is_better=False),
                EarlyStopping(patience=3, threshold=0.01,
                              threshold_mode='rel', lower_is_better=True,
                              load_best=True)]
)

ce = CrossEmbeddings(dataset = main,
                     text_col = 'text',
                     image_col = 'img_get',
                     d_col = 'ln_p',
                     y_col = 'ln_q',
                     n_folds = 3,
                     aux_d = model_p,
                     aux_y = model_q,
                     txt_str = TXT,
                     img_str = IMG)

ce.fit_and_predict_embeddings()
df = ce.get_embedded_df()

with open(path + '/toys_embedded_df.pkl', 'wb') as f:
    pickle.dump(df.drop(columns=['img_get']), f)
    #Drop image column to significantly reduce size of the pickled object.
```

The SLURM output in Appendix C.8 shows the training progress of the `class CrossEmbeddings` instance. Only the results obtained using the methods described in the previous sections are presented below. Table 30 shows the summary of five PLR models with different learners.

Table 30: Summary of DoubleMLPLR models with different Learners for full Analysis

Source: own table.

	coef	std err	t	P>|t|	2.5 %	97.5 %
LGBM	-0.198589	0.019760	-10.050284	9.160309e-24	-0.237317	-0.159861
Lasso	-0.190294	0.019579	-9.719162	2.498296e-22	-0.228668	-0.151919
Linear reg	-0.193276	0.019500	-9.911839	3.697752e-23	-0.231495	-0.155058
Neural net	-0.181464	0.019693	-9.214812	3.118246e-20	-0.220061	-0.142867
Random Forest	-0.195180	0.019819	-9.848067	6.987484e-23	-0.234025	-0.156335

Figure 51 visualizes the estimated coefficients for the price elasticity including the 95 % CI.

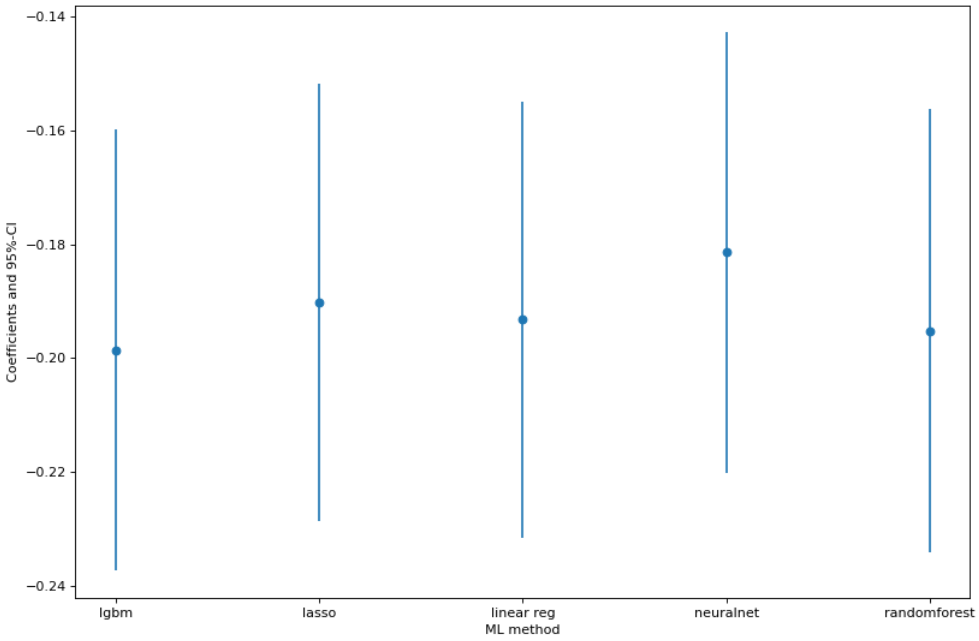


Figure 51: Estimated Coefficient and Confidence Intervals for full Analysis

Source: own figure.

Additionally, Figure 52 and Table 31 are showing the RMSE values of the used learners.

Table 31: Root Mean Squared Error values for considered learners in full Analysis

Source: own table.

Model	ml_m	ml_l
LGBM	0.497630	0.956515
Lasso	0.495906	0.948835
Linear reg	0.500585	0.957042
Neural net	0.501776	0.973961
Random Forest	0.493365	0.948052

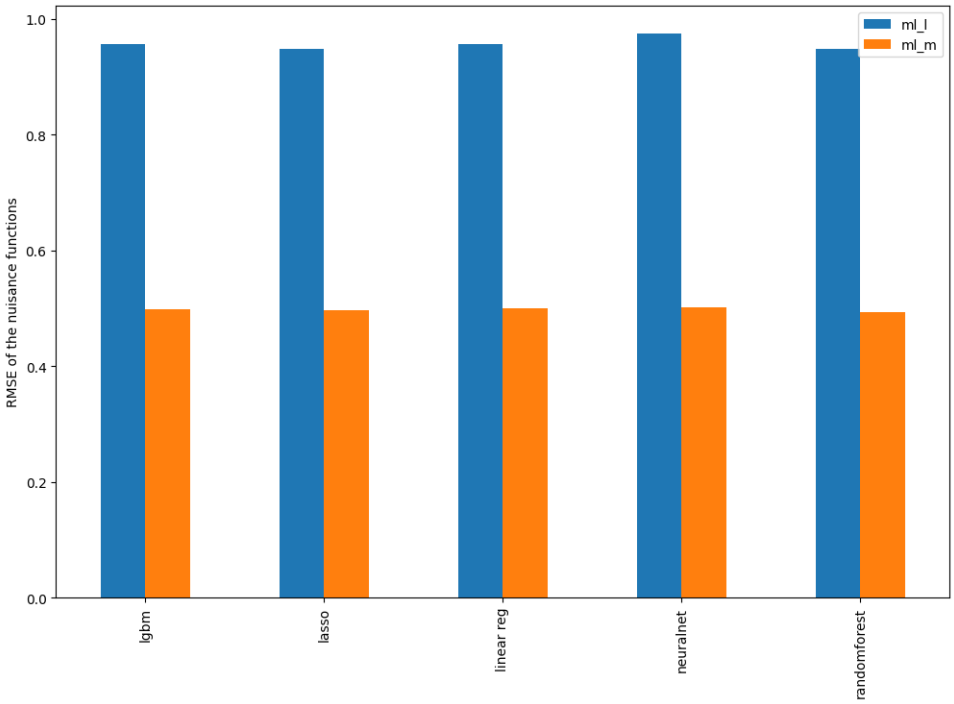


Figure 52: Root Mean Squared Error values of the used Learners in the Causal Model for full Analysis

Source: own figure.

The relationship between the target and the treatment variable is shown in Figures 53 and 54 (mean-centred).

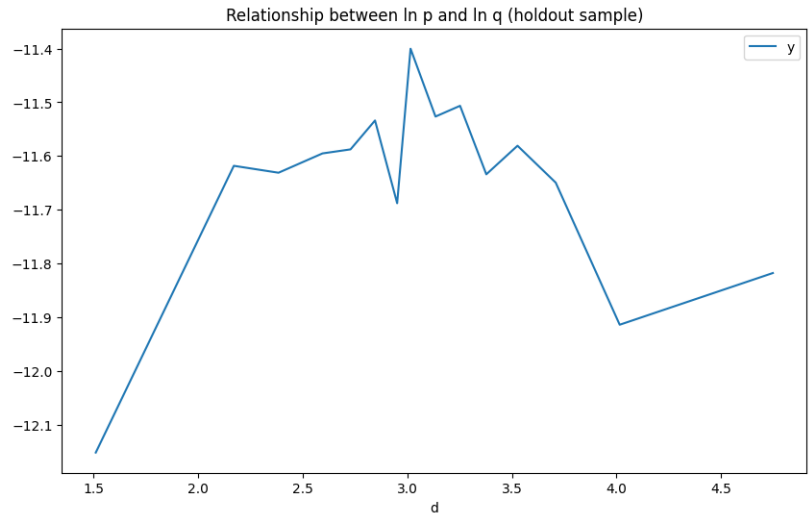


Figure 53: Relationship between Output and Treatment Variable for full Analysis

Source: own figure.

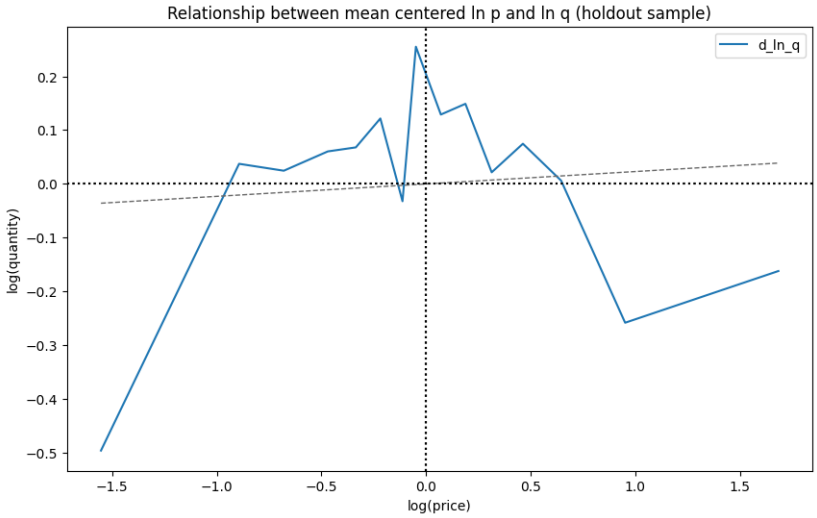


Figure 54: Relationship between Output and Treatment Variable after mean-centring for full Analysis

Source: own figure.

Following the descriptions in subsection 6.6.2, the Lasso predictions from the `DoubleMLPLR` instance are used to calculate the residuals r_q and r_p. Finally, the relationship after residualizing is shown in Figure 55.

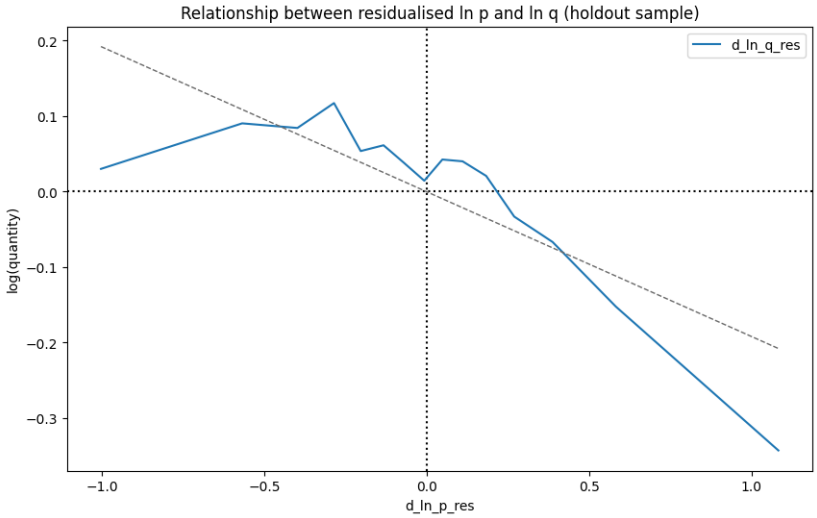


Figure 55: Relationship between Output and Treatment Variable after Residualizing for full Analysis

Source: own figure.

The OLS Regression results of r_q on r_q are presented in Table 32. It is again to be noted that the coefficient for the residualized treatment (see Table 32) corresponds to the estimator of the `DoubleMLPLR` instance that utilizes Lasso as a learner (see Table 30).

Table 32: Summary of Ordinary Least Squares Regression for full Analysis

Source: own table.

Dep. Variable:	d_ln_q_res	**R-squared:**	0.010
Model:	OLS	**Adj. R-squared:**	0.010
Method:	Least Squares	**F-statistic:**	91.96
Date:	Tue, 04 Apr 2023	**Prob (F-statistic):**	1.12e-21
Time:	09:59:17	**Log-Likelihood:**	-12535.
No. Observations:	9207	**AIC:**	2.507e+04
Df Residuals:	9205	**BIC:**	2.509e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	-0.0003	0.010	-0.032	0.974	-0.020	0.019
d_ln_p_res	-0.1903	0.020	-9.590	0.000	-0.229	-0.151

The full analysis for the toys dataset can be found in the notebook `Case_Study_Amazon_Toy_Data_(Full).ipynb` in Appendix C.6. The full analysis for the chairs data is available in the corresponding notebook located in Appendix C.7.

6.7 Summary Case Study

Summarizing the results of the case study, Table 33 shows the estimates of the naive approach and the presented DML approach (related to Lasso models for the nuisance functions) for the first and the full analysis.

Table 33: Price Elasticity of Demand for Naive and Causal Estimation
Source: own table.

	Price Elasticity
Naive Estimate	0.0206
Causal Estimate Lasso (first)	-0.2241
Causal Estimate Lasso (full)	-0.1903

It is apparent that the direction and magnitude of the causal effect estimated for the first and the full data analysis are similar and contrary to the naive estimate (see Table 33). The naive estimate indicates that demand increases as the price increases, whereas the causal estimates demonstrate a decrease in quantity demanded with an increase in price. All three estimates show that demand is inelastic.

To summarize the approach of the full data analysis, the `CrossEmbedding` method was used to generate embeddings from the multimodal transformer model, enabling the utilization of the entire dataset for estimating the price elasticities. This approach offers several benefits, including the ability to utilize a larger sample size, resulting in more reasonable and robust estimates.

Upon examining the coefficient estimates, it can be observed that the models trained on the full dataset using `CrossEmbedding` exhibit slightly smaller coefficients compared to the models trained on the holdout set in the first analysis (as described in subsection 6.6.2). This suggests that the models trained on the full dataset are more conservative in predicting the treatment effect. This conservatism could stem from the larger sample size allowing the models to better capture the true treatment effect, leading to more precise coefficient estimates.

The standard errors for the models in the full analysis (see subsection 6.6.3) are also smaller compared to the models trained on the holdout set. This could be because the larger sample size reduced the amount of noise in the data, allowing the models to estimate the variability in the data more accurately.

The absolute t-values for the models trained on the full dataset are also larger compared to the models trained on the holdout set. This indicates that the models trained on the full dataset have more statistically significant coefficients, which is likely due to the larger sample size.

All in all, the results suggest that using `CrossEmbeddings` to generate embeddings

from the entire dataset leads to more reasonable and statistically significant coefficient estimates, likely due to the larger sample size.

Employing a multimodal transformer model to create embeddings from confounding variables, and implementing a cross-fitting technique to address possible overfitting and data loss, highlights the efficacy of this method. This approach proves effective in capturing complex and nonlinear relationships between multimodal variables and overcoming limited data availability.

The analysis reveals consistent findings across different methods used for estimating the price elasticity of demand for toys. The results indicate a negative and statistically significant relationship between price and quantity demanded, in line with economic theory. The estimated treatment effect θ (i.e., the coefficient for the treatment variable $\ln p$) ranges from -0.1815 to -0.1986, depending on the learning method used, with standard errors ranging from 0.0195 to 0.0198. These findings have implications for toy manufacturers and retailers, including the ability to forecast the impact of price changes on sales and make informed decisions about pricing strategies.

The study showed that controlling with confounders significantly changed the estimation of price elasticities (see Table 33). The findings demonstrated that the approach of using DML with multimodal transformer models and embeddings can lead to reasonable estimates of price elasticities from observational data. Besides, the findings of this case study also suggest that the approach of using multimodal transformer models and embeddings for confounders may have applicability in other contexts where similar complexities and unstructured confounding variables are present. This approach has, therefore, the potential to be useful in fields such as economics, (dynamic-) pricing, and marketing, where accurate estimation of price elasticities is crucial for making informed decisions.

In general, this case study explored the use of the DML package to estimate price elasticities from observational data. The study employed a multimodal transformer model to include confounders in the estimation process. Furthermore, the study investigated the use of embeddings from these models as covariates for the partialling out approach using the `DoubleMLPLR` model. The case study also pointed out the importance of using CML methods, as the naive approach would have led to a contrary interpretation which is not reasonable.

7 Conclusion

7.1 Summary and Discussion of Results

This thesis contributes to the growing body of literature on CML by implementing and investigating the use of DL approaches for causal inference in the context of the DML framework. In this thesis, an object-oriented software implementation is developed that allows the application of DL approaches to estimate a causal effect while using the DML package. One presented strategy is to use neural networks as ML learners in the DML framework to estimate the nuisance functions. A second strategy is to use DL models for representation learning, i.e. to learn effective low-dimensional representations/ embeddings from unstructured and high-dimensional multimodal data. These embeddings can then serve as confounding factors within the DML framework.

A simulation study proves the valid implementation of neural networks as learners for different causal models, DGPs, and data settings. In addition, the study gives valuable insights into the finite sample properties of neural networks in semiparametric causal inference and examines the performance of DL approaches compared to standard ML methods. Furthermore, the simulation study offers, due to its design, a flexible software implementation for future research. Following the results of the simulation study and the implications arising from the theoretical foundations, this thesis additionally seeks to determine potential use cases and focuses on one of such use cases during the case study. For the case study, the price elasticity of demand is estimated as a causal parameter based on observational data. The theoretical implication to control for relevant confounders brings along the necessity to include text and image data. In the case study, advanced DL approaches are used to learn embeddings from text and image data in order to adjust for those confounders and to provide high-quality causal estimates for the price elasticity of demand. The case study shows that the use of DL approaches is not only beneficial but rather inevitable when estimating a causal effect in some real-world economic context where unstructured data is present.

Regarding the implementation of DL approaches as learners in the existing DML package, this thesis provides and proves an object-oriented solution using the popular DL frameworks PyTorch and Keras. According to the findings, the existing wrappers, that enable compatibility between the DL frameworks and scikit-learn, need some minor modifications in the data dimension to meet the requirements of the DML package. Both implementations are validated and show that the provided solution can be used to obtain correct estimates of a causal effect in a basic linear setting.

Proceeding further, the results of the simulation study highlight that the presented implementation of DL approaches as learners in the DML framework works for a great variety of different DGPs since no systematic deviations or other anomalies compared to standard ML methods are apparent. The exploratory phase of the simulation study further shows that the DL methods lack behind standard ML methods in terms of performance for most of the considered scenarios and network architectures. Nevertheless, the exploratory phase reveals scenarios where DL ap-

proaches generally show superior performance in providing unbiased estimates for a causal effect and where the application of such DL methods, therefore, seems promising.

The detail phase of the simulation study further investigates the finite sample properties of using DL approaches as learners in the context of causal inference for two selected DGPs in more detail. It shows that the performance of a DL model heavily depends on the interaction of its architecture and the considered setting. In general, the simulation study exposes that standard ML methods prove to have a good and consistent performance in providing unbiased and accurate estimates of a causal parameter for settings with structured, synthetic data but that there are also specific scenarios and settings where DL methods outperform the other methods. Concluding, no systematic superiority of using DL approaches as learners in the DML framework for specific data settings or data dimensions is obvious from the resulting measures. All in all, the results of the simulation study are mostly in line with the findings of the limited existing literature on CML with DL approaches that investigated the use of DL approaches in such a causal inference context.

Another finding of the simulation study is that the use of DL approaches as a learner brings along significant benefits when it comes to the computational time for large and high-dimensional data settings. Together with the theoretical explanations of image and text data and its use in causal inference settings, potential use cases for DL approaches in CML are presented.

One of such use cases is specifically explored in the case study. The case study focuses on using DL approaches for representation learning in order to adjust for confounding images and texts. Specifically, the price elasticity of demand is estimated as a causal parameter on the basis of observational sales data from Amazon. Sophisticated DL models are applied to product text and product images to generate effective embeddings which ultimately can be used in the DML package to estimate the price elasticity of demand. The case study highlights that controlling for confounders observed from unstructured, multimodal data allows estimating reasonable values for the price elasticity of demand from observational data. Furthermore, the case study shows that the choice of a ML learner has no significant impact on the resulting estimate or the accuracy when highly effective embeddings have been generated.

The explanations in the case study can serve as an exemplary guide on how to effectively use DL approaches for learning effective representations of unstructured confounding variables in a CML context. Furthermore, the case study also provides guidance on how to use generated embeddings in the DML package for some real-world economic causal inference problems. The first analysis indicates that the considered approach can effectively capture complex and nonlinear relationships between multimodal variables. The full data analysis utilizes the cross-fitting method to overcome the issues of overfitting bias and limited availability of data.

The case study shows that there is a great opportunity to obtain accurate causal estimates from observational data by adjusting for latent confounding factors in texts and

images. This opportunity arises from the increasing availability of text and image data, as well as improved and sophisticated DL methods for learning representations of these data. While the usefulness of image and text data for causal inference is indisputable, the combination of NLP and computer vision with causal inference tackles several topics and problems of current research. Using DL approaches in the DML package provides one possible and promising solution for this purpose which will even become more prominent in the future.

7.2 Limitations and Future Research

It is important to note that the results and implications presented are subject to some limitations due to the limited scope of this work. Besides, several interesting and relevant directions for future research emerge from this thesis. Future research can, therefore, build on this thesis and the software implementations presented to gain and evolve a better understanding of the use of DL approaches in the context of CML.

For the first limitation, it should be noted that the simulation study only presents results for the two causal models PLR and IRM. There are no scenarios investigated using causal models that contain IVs. Secondly, only selected neural network models with different architectures are used to represent the DL approaches in the simulation study. It is worth mentioning that no considered network architecture is particularly deep due to constraints in computational resources. Furthermore, only a limited number of selected DGPs, factor variations, and data settings are explored during the simulation study. Another limitation of the simulation study arises from causal evaluation with synthetic data in general. Following Keith et al. [51], using synthetic data brings along many degrees of freedom induced by the researcher which can ultimately limit the effectiveness of such estimations. Additionally, Parikh et al. [81] state that DGPs with synthetic data are oversimplified and may tend to favour certain methods over others a priori.

For the case study, limitations in the present thesis are at first the quality of the data available. For example, the actual sales of a certain product are not available in the given data set which is why the sales rank is used as the best factor for predicting sales. Besides, it is possible that not all relevant confounding factors could be observed by the product text and image. Another limitation arises due to computational constraints. In the presented thesis, it is only possible to examine a limited number of text and image representation models which are also restricted in their size. A further limitation is the generalizability of the resulting estimates for the price elasticity of demand since only an online market situation has been observed. Therefore, the resulting price elasticity might not be transferable to other economic settings. According to Keith et al. [51] another limitation is given due to the robustness of the results with regard to the model specifications. It is stated that the causal effects from an analysis including language models might be sensitive to hyperparameter tuning or random restarts [51].

As mentioned earlier, the research in this thesis presents an additional endeavour to explore the use of DL approaches in the context of causal inference. Out of the limitations presented above, several interesting and relevant directions for future research

arise.

The simulation study has been designed and implemented using the structured Python library `DMLSim` which is easily adaptable and expandable. By expanding the existing simulation study design, several topics that were not covered in this thesis due to its limited scope can be easily studied. These include investigating the use of DL approaches for PLIV and IIVM models, studying the performance of particularly deep neural networks or exploring various other DGPs and data settings. To overcome the limitations of synthetic data, Keith et al. [51] suggest to use also DGPs with semi-synthetic data which can be one topic for further research. Another relevant direction for future research is the adaption and alignment of the neural network architecture to a specific data setting since the simulation study shows that the performance of a DL model for a given data setting heavily depends on its architecture.

Directions for future research regarding the presented case study are first and foremost the development of a data backend in the DML package which allows the use of multimodal and unstructured data. This would be an interesting basis to investigate the direct use of sophisticated DL approaches as learners in the DML package. Another direction for future research is the examination of other fusion approaches.

In summary, despite the limitations and the possible directions for further research, this thesis provides valuable insights into the use of DL approaches for CML by providing a software implementation to easily apply such methods in the DML package and by providing some initial study on the performance of such methods in various settings. Ultimately, the inevitable use of DL approaches in a real-world economic context is highlighted in the case study.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. "TensorFlow, large-scale machine learning on heterogeneous systems." (Nov. 2015), [Online]. Available: https://github.com/tensorflow/tensorflow (visited on 01/07/2023).

[2] C. Akkus, L. Chu, V. Djakovic, S. Jauch-Walser, P. Koch, G. Loss, C. Marquardt, M. Moldovan, N. Sauter, M. Schneider, R. Schulte, K. Urbanczyk, J. Goschenhofer, C. Heumann, R. Hvingelby, D. Schalk, and M. Aßenmacher, *Multimodal deep learning*, Jan. 12, 2023. DOI: 10.48550/arXiv.2301.04856.

[3] Amazon. "About amazon best sellers rank - amazon-kundenservice." (), [Online]. Available: https://www.amazon.com/gp/help/customer/display.html?nodeId=525376 (visited on 02/28/2023).

[4] S. Athey, G. W. Imbens, and S. Wager, *Approximate residual balancing: De-biased inference of average treatment effects in high dimensions*, Jan. 31, 2018. DOI: 10.48550/arXiv.1604.07125.

[5] No-author. "Lysandre/tiny-vit-random · hugging face." (), [Online]. Available: https://huggingface.co/lysandre/tiny-vit-random (visited on 03/28/2023).

[6] P. Bach. "DoubleML tutorial materials - r: Estimation of price elasticities with double/debiased machine learning." (Sep. 14, 2022), [Online]. Available: https://github.com/DoubleML/tutorial_materials/blob/16f44404a308457d1c040ccb52078c7a360856ec/notebooks/Demand/R_elasticity_analysis.ipynb (visited on 02/28/2023).

[7] P. Bach, V. Chernozhukov, M. S. Kurz, and M. Spindler, "DoubleML - an object-oriented implementation of double machine learning in python," *Journal of Machine Learning Research*, vol. 23, no. 53, pp. 1–6, 2022. DOI: 10.48550/arXiv.2104.03220.

[8] P. Bach, V. Chernozhukov, M. S. Kurz, and M. Spindler, *DoubleML – an object-oriented implementation of double machine learning in r*, Aug. 4, 2022. DOI: 10.48550/arXiv.2103.09603.

[9] J. Banks, Ed., *Discrete-event system simulation*, 5th ed, Pearson, 2010, 622 pp.

[10] H. Bao, L. Dong, S. Piao, and F. Wei, *BEiT: BERT pre-training of image transformers*, Sep. 3, 2022. DOI: 10.48550/arXiv.2106.08254.

[11] K. Battocchi, E. Dillon, M. Hei, G. Lewis, P. Oka, M. Oprescu, and V. Syrgkanis. "EconML: A python package for ML-based heterogeneous treatment effects estimation." (2019), [Online]. Available: https://github.com/microsoft/EconML (visited on 02/07/2023).

[12] P. Beaumont, B. Horsburgh, P. Pilgerstorfer, A. Droth, R. Oentaryo, S. Ler, H. Nguyen, G. A. Ferreira, Z. Patel, and W. Leong. "CausalNex." (Oct. 2021), [Online]. Available: https://github.com/quantumblacklabs/causalnex (visited on 02/07/2023).

[13] A. Belloni, V. Chernozhukov, I. Fernández-Val, and C. Hansen, "Program evaluation and causal inference with high-dimensional data," *Econometrica*, vol. 85, no. 1, pp. 233–298, 2017. DOI: 10.3982/ECTA12723.

[14] P. Bhargava, A. Drozd, and A. Rogers, *Generalization in NLI: Ways (not) to go beyond simple heuristics*, Oct. 4, 2021. DOI: 10.48550/arXiv.2110.01518.

[15] A. Burton, D. G. Altman, P. Royston, and R. L. Holder, "The design of simulation studies in medical statistics," *Statistics in Medicine*, vol. 25, no. 24, pp. 4279–4292, 2006. DOI: 10.1002/sim.2673.

[16] M. Cannas and B. Arpino, "A comparison of machine learning algorithms and covariate balance measures for propensity score matching and weighting," *Biometrical Journal*, vol. 61, no. 4, pp. 1049–1072, 2019. DOI: 10.1002/bimj.201800132.

[17] V. Chernozhukov, D. Chetverikov, M. Demirer, E. Duflo, C. Hansen, W. Newey, and J. Robins, "Double/debiased machine learning for treatment and structural parameters," *The Econometrics Journal*, vol. 21, no. 1, pp. C1–C68, Feb. 1, 2018. DOI: 10.1111/ectj.12097.

[18] H. A. Chipman, E. I. George, and R. E. McCulloch, "BART: Bayesian additive regression trees," *The Annals of Applied Statistics*, vol. 4, no. 1, pp. 266–298, Mar. 2010. DOI: 10.1214/09-AOAS285.

[19] R. Cobzaru, S. Jiang, K. Ng, S. Finkelstein, R. Welsch, and Z. Shahn, "State of the art causal inference in the presence of extraneous covariates: A simulation study," *AMIA Annual Symposium Proceedings*, vol. 2021, pp. 334–342, Feb. 21, 2022.

[20] O. Dalle, "On reproducibility and traceability of simulations," in *Proceedings of the Winter Simulation Conference*, ser. WSC '12, Berlin, Germany: Winter Simulation Conference, Dec. 9, 2012, pp. 1–12. DOI: 10.1109/WSC.2012.6465284.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, May 24, 2019. DOI: 10.48550/arXiv.1810.04805.

[22] Y. Dodge, Ed., *The Oxford dictionary of statistical terms*, First published in paperback 2006, Oxford: Oxford University Press, 2006, 498 pp.

[23] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, Jun. 3, 2021. DOI: 10.48550/arXiv.2010.11929.

[24] N. Egami, C. J. Fong, J. Grimmer, M. E. Roberts, and B. M. Stewart, *How to make causal inferences using texts*, Feb. 6, 2018. DOI: 10.48550/arXiv.1802.02163.

[25] J. E. Engelberg and C. A. Parsons, "The causal impact of media in financial markets," *The Journal of Finance*, vol. 66, no. 1, pp. 67–97, Feb. 2011. DOI: 10.1111/j.1540-6261.2010.01626.x.

[26] M. H. Farrell, T. Liang, and S. Misra, "Deep neural networks for estimation and inference," *Econometrica*, vol. 89, no. 1, pp. 181–213, 2021. DOI: 10.3982/ECTA16901.

[27] Fgpacini. "Feature learning diagram." (Oct. 8, 2022), [Online]. Available: https://commons.wikimedia.org/wiki/File:Feature_Learning_Diagram.png (visited on 04/21/2023).

[28] N. Fingerhut, M. Sesia, and Y. Romano, *Coordinated double machine learning*, Jun. 2, 2022. DOI: 10.48550/arXiv.2206.00885.

[29] J. H. Friedman, "Multivariate adaptive regression splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1–67, Mar. 1991. DOI: 10.1214/aos/1176347963.

[30] J. H. Friedman, E. Grosse, and W. Stuetzle, "Multidimensional additive spline approximation," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 2, pp. 291–301, Jun. 1983. DOI: 10.1137/0904023.

[31] M. Gentzkow, B. Kelly, and M. Taddy, "Text as data," *Journal of Economic Literature*, vol. 57, no. 3, pp. 535–574, Sep. 2019. DOI: 10.1257/jel.20181020.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (Adaptive computation and machine learning). Cambridge, Massachusetts: The MIT Press, 2016, 775 pp.

[33] B. R. Gordon, R. Moakler, and F. Zettelmeyer, *Close enough? a large-scale exploration of non-experimental approaches to advertising measurement*, Oct. 4, 2022. DOI: 10.48550/arXiv.2201.07055.

[34] O. Goudet, D. Kalainathan, P. Caillou, I. Guyon, D. Lopez-Paz, and M. Sebag, *Learning Functional Causal Models with Generative Neural Networks*. 2018. DOI: 10.1007/978-3-319-98131-4.

[35] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, p. 424, Aug. 1969. DOI: 10.2307/1912791.

[36] R. L. Harrison, C. Granja, and C. Leroy, "Introduction to monte carlo simulation," presented at the Fifth International Summer School on Nuclear Physics Methods and Accelerators in Biology and Medicine, Bratislava (Slovakia), 2010, pp. 17–21. DOI: 10.1063/1.3295638.

[37] Helium10. "Amazon sales estimator," Helium 10. (), [Online]. Available: https://www.helium10.com/tools/free/amazon-sales-estimator/ (visited on 03/03/2023).

[38] M. A. Hernán, J. Hsu, and B. Healy, "A second chance to get causal inference right: A classification of data science tasks," *CHANCE*, vol. 32, no. 1, pp. 42–49, Jan. 2, 2019. DOI: 10.1080/09332480.2019.1579578.

[39] D. R. C. Hill, "Theory of modelling and simulation: Integrating discrete event and continuous complex dynamic systems: Second edition by b. p. zeigler, h. praehofer, t. g. kim, academic press, san diego, CA, 2000.," *International Journal of Robust and Nonlinear Control*, vol. 12, no. 1, pp. 91–92, 2002. DOI: 10.1002/rnc.610.

[40] P. W. Holland, "Statistics and causal inference," *Journal of the American Statistical Association*, vol. 81, no. 396, pp. 945–960, Dec. 1986. DOI: 10.1080/01621459.1986.10478354.

[41] HPC-Team. "High-Performance Computing (HPC)." (Oct. 8, 2022), [Online]. Available: https://www.rrz.uni-hamburg.de/services/hpc.html (visited on 01/27/2023).

[42] HuggingFace. "Transformers - API - main class - model outputs." (), [Online]. Available: https://huggingface.co/docs/transformers/main_classes/output (visited on 03/28/2023).

[43] C. T. Jerzak, F. Johansson, and A. Daoud, *Estimating causal effects under image confounding bias with an application to poverty in africa*, Feb. 15, 2023. DOI: 10.48550/arXiv.2206.06410.

[44] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, *TinyBERT: Distilling BERT for natural language understanding*, Oct. 15, 2020. DOI: 10.48550/arXiv.1909.10351.

[45] Jungle-Scout. "Amazon sales estimator - predict your sales | rank calculator," Jungle Scout. (2023), [Online]. Available: https://www.junglescout.com/estimator/ (visited on 02/28/2023).

[46] KDnuggets. "PyTorch or TensorFlow? comparing popular machine learning frameworks," KDnuggets. (Feb. 24, 2022), [Online]. Available: https://www.kdnuggets.com/pytorch-or-tensorflow-comparing-popular-machine-learning-frameworks.html (visited on 01/07/2023).

[47] Keepa. "Statistics object," Keepa Community Forums. (Sep. 2016), [Online]. Available: https://discuss.keepa.com/t/glossary-where/12694 (visited on 02/28/2023).

[48] Keepa. "What's the category the SalesRank data is referring to?" Keepa Community Forums. (Sep. 28, 2016), [Online]. Available: https://discuss.keepa.com/t/whats-the-category-the-salesrank-data-is-referring-to/1383/2?u=jessica (visited on 02/28/2023).

[49] Keepa. "Glossary, where?" Keepa Community Forums. (Apr. 19, 2021), [Online]. Available: https://discuss.keepa.com/t/glossary-where/12694 (visited on 03/03/2023).

[50] Keepa. "Average sales per month and per day," Keepa Community Forums. (May 31, 2022), [Online]. Available: https://discuss.keepa.com/t/average-sales-per-month-and-per-day/13990 (visited on 02/28/2023).

[51] K. A. Keith, D. Jensen, and B. O'Connor, *Text and causal inference: A review of using text to remove confounding from causal estimates*, May 1, 2020. DOI: 10.48550/arXiv.2005.00649.

[52] "Keras: Deep learning for humans." (Jan. 7, 2023), [Online]. Available: https://github.com/keras-team/keras (visited on 01/07/2023).

[53] N. Ketkar and J. Moolayil, "Introduction to PyTorch," in *Deep Learning with Python*. Berkeley, CA: Apress, 2021, pp. 27–91. DOI: 10.1007/978-1-4842-5364-9_2.

[54] E. Kiciman, S. Counts, and M. Gasser, "Using longitudinal social media analysis to understand the effects of early college alcohol use," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 12, no. 1, Jun. 15, 2018. DOI: 10.1609/icwsm.v12i1.15012.

[55] T. Kim. "Why is a regression coefficient covariance/variance," Cross Validated. (Dec. 12, 2021), [Online]. Available: https://stats.stackexchange.com/q/555855 (visited on 03/31/2023).

[56] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, Jan. 29, 2017. DOI: 10.48550/arXiv.1412.6980.

[57] M. C. Knaus, *A double machine learning approach to estimate the effects of musical practice on student's skills*, Jan. 9, 2019. DOI: 10.48550/arXiv.1805.10300.

[58] M. C. Knaus, "Double machine learning based program evaluation under unconfoundedness," *The Econometrics Journal*, vol. 25, no. 3, pp. 602–627, Sep. 14, 2022. DOI: 10.1093/ectj/utac015.

[59] M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl, "Mlr3: A modern object-oriented machine learning framework in r," *Journal of Open Source Software*, vol. 4, no. 44, p. 1903, Dec. 11, 2019. DOI: 10.21105/joss.01903.

[60] H. Langen and M. Huber, *How causal machine learning can leverage marketing strategies: Assessing and improving the performance of a coupon campaign*, Apr. 22, 2022. DOI: 10.48550/ARXIV.2204.10820.

[61] B. K. Lee, J. Lessler, and E. A. Stuart, "Improving propensity score weighting using machine learning," *Statistics in Medicine*, vol. 29, no. 3, pp. 337–346, 2010. DOI: 10.1002/sim.3782.

[62] H.-C. Lee, M. Danieletto, R. Miotto, S. T. Cherng, and J. T. Dudley, "Scaling structural learning with NO-BEARS to infer causal transcriptome networks," in *Biocomputing 2020*, Dec. 2019, pp. 391–402. DOI: 10.1142/9789811215636_0035.

[63] S. Lee and M. Rho, "Multimodal deep learning applied to classify healthy and disease states of human microbiome," *Scientific Reports*, vol. 12, Jan. 17, 2022. DOI: 10.1038/s41598-022-04773-3.

[64] S.-H. Lin and M. A. Ikram, "On the relationship of machine learning with causal inference," *European Journal of Epidemiology*, vol. 35, no. 2, pp. 183–185, Feb. 2020. DOI: 10.1007/s10654-019-00564-9.

[65] J. Lipkova, R. J. Chen, B. Chen, M. Y. Lu, M. Barbieri, D. Shao, A. J. Vaidya, C. Chen, L. Zhuang, D. F. K. Williamson, M. Shaban, T. Y. Chen, and F. Mahmood, "Artificial intelligence for multimodal data integration in oncology," *Cancer Cell*, vol. 40, no. 10, pp. 1095–1110, Oct. 10, 2022. DOI: 10.1016/j.ccell.2022.09.012.

[66] F. Lorig, "Foundations and methods of simulation," in *Hypothesis-Driven Simulation Studies*. Wiesbaden: Springer Fachmedien, 2019, pp. 13–48. DOI: 10.1007/978-3-658-27588-4_2.

[67] Y. Luo, J. Peng, and J. Ma, "When causal inference meets deep learning," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 426–427, Aug. 2020. DOI: 10.1038/s42256-020-0218-x.

[68] E. Manzoor. "Controlling for text in causal inference." (2021), [Online]. Available: https://nlp-css-201-tutorials.github.io/nlp-css-201-tutorials/docs/NLP+CSS201%20-%20Controlling%20for%20Text%20in%20Causal%20Inference.pdf (visited on 02/07/2023).

[69] E. Manzoor, G. H. Chen, D. Lee, and M. D. Smith, *Influence via ethos: On the persuasive power of reputation in deliberation online*, Jun. 1, 2020. DOI: 10.48550/arXiv.2006.00707.

[70] Microsoft-Research. "Econml.dml.DML — econml 0.14.0 documentation." (2022), [Online]. Available: https://econml.azurewebsites.net/_autosummary/econml.dml.DML.html#econml.dml.DML (visited on 02/07/2023).

[71] Microsoft-Research. "Introduction to causal inference." (2022), [Online]. Available: https://econml.azurewebsites.net/spec/causal_intro.html (visited on 02/07/2023).

[72] Microsoft-Research. "Machine learning based estimation of heterogeneous treatment effects — econml 0.14.0 documentation." (2022), [Online]. Available: https://econml.azurewebsites.net/spec/motivation.html (visited on 02/07/2023).

[73] Microsoft-Research. "Orthogonal/double machine learning — econml 0.14.0 documentation." (2022), [Online]. Available: https://econml.azurewebsites.net/spec/estimation/dml.html (visited on 02/07/2023).

[74] Microsoft-Research. "Overview — econml 0.14.0 documentation." (2022), [Online]. Available: https://econml.azurewebsites.net/spec/overview.html (visited on 02/07/2023).

[75] P. Migdal and R. Jakubanis. "Keras or PyTorch as your first deep learning framework," deepsense.ai. (Jun. 26, 2018), [Online]. Available: https://deepsense.ai/keras-or-pytorch/ (visited on 01/07/2023).

[76] S. Miller, J. Howard, P. Adams, M. Schwan, and R. Slater, "Multi-modal classification using images and text," *SMU Data Science Review*, vol. 3, no. 3, Jan. 11, 2021.

[77] D. C. Montgomery, *Design and analysis of experiments*, Eighth edition. Hoboken, NJ: John Wiley & Sons, Inc, 2013, 730 pp.

[78] R. Moraffah, P. Sheth, M. Karami, A. Bhattacharya, Q. Wang, A. Tahir, A. Raglin, and H. Liu, "Causal inference for time series analysis: Problems, methods and evaluation," *Knowledge and Information Systems*, vol. 63, no. 12, pp. 3041–3085, Dec. 1, 2021. DOI: 10.1007/s10115-021-01621-0.

[79] S. L. Morgan and C. Winship, *Counterfactuals and Causal Inference: Methods and Principles for Social Research*, 1st ed. Cambridge University Press, Jul. 30, 2007. DOI: 10.1017/CBO9780511804564.

[80] T. P. Morris, I. R. White, and M. J. Crowther, "Using simulation studies to evaluate statistical methods," *Statistics in Medicine*, vol. 38, no. 11, pp. 2074–2102, May 20, 2019. DOI: 10.1002/sim.8086.

[81] H. Parikh, C. Varjao, L. Xu, and E. T. Tchetgen, "Validating causal inference methods," in *Proceedings of the 39th International Conference on Machine Learning*, PMLR, Jun. 28, 2022, pp. 17 346–17 358. DOI: 10.48550/arXiv.2202.04208.

[82] R. Pedace, *Econometrics for dummies* (–For dummies). Hoboken, NJ: John Wiley & Sons Inc, 2013, 342 pp.

[83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. DOI: 10.48550/arXiv.1201.0490.

[84] V. S. Pugachev, "CHAPTER 8 - ESTIMATION OF DISTRIBUTIONS," in *Probability Theory and Mathematical Statistics for Engineers*, V. S. Pugachev, Ed., Pergamon, Jan. 1, 1984, pp. 273–310. DOI: 10.1016/B978-0-08-029148-2.50012-4.

[85] PyTorch-Contributors. "PyTorch documentation — PyTorch 1.13 documentation." (2022), [Online]. Available: https://pytorch.org/docs/stable/index.html (visited on 01/07/2023).

[86] A. Rahate, R. Walambe, S. Ramanna, and K. Kotecha, "Multimodal co-learning: Challenges, applications with datasets, recent advances and future directions," *Information Fusion*, vol. 81, pp. 203–239, May 2022. DOI: 10.1016/j.inffus.2021.12.003.

[87] V. Ramachandra, *Deep learning for causal inference*, Feb. 28, 2018. DOI: 10.48550/arXiv.1803.00149.

[88] V. Ramachandra, *Causal inference for climate change events from satellite image time series using computer vision and deep learning*, Oct. 24, 2019. DOI: 10.48550/arXiv.1910.11492.

[89] P. M. Robinson, "Root-n-consistent semiparametric regression," *Econometrica*, vol. 56, no. 4, p. 931, Jul. 1988. DOI: 10.2307/1912705.

[90] L. Roemheld. "Causal inference in the wild: Elasticity pricing." (Jul. 8, 2021), [Online]. Available: https://towardsdatascience.com/causal-inference-example-elasticity-de4a3e2e621b (visited on 02/26/2023).

[91] L. Roemheld. "Example case for causal inference in ecommerce pricing." (Jul. 8, 2021), [Online]. Available: https://github.com/larsroemheld/causalinf_ex_elasticity/blob/f102ec3acf6f1cb6a2a6bc70c0914f51a4f783c7/elasticity_dml.ipynb (visited on 03/28/2023).

[92] P. R. Rosenbaum and D. B. Rubin, "The central role of the propensity score in observational studies for causal effects," *Biometrika*, vol. 70, no. 1, pp. 41–55, Apr. 1, 1983. DOI: 10.1093/biomet/70.1.41.

[93] D. B. Rubin, "Estimating causal effects of treatments in randomized and nonrandomized studies.," *Journal of Educational Psychology*, vol. 66, no. 5, pp. 688–701, Oct. 1974. DOI: 10.1037/h0037350.

[94] J. Runge, S. Bathiany, E. Bollt, G. Camps-Valls, D. Coumou, E. Deyle, C. Glymour, M. Kretschmer, M. D. Mahecha, J. Muñoz-Marí, E. H. van Nes, J. Peters, R. Quax, M. Reichstein, M. Scheffer, B. Schölkopf, P. Spirtes, G. Sugihara, J. Sun, K. Zhang, and J. Zscheischler, "Inferring causation from time series in earth system sciences," *Nature Communications*, vol. 10, no. 1, p. 2553, Jun. 14, 2019. DOI: 10.1038/s41467-019-10105-3.

[95] S. Sanchez, *Work Smarter, Not Harder: Guidelines for Designing Simulation Experiments*. Jan. 1, 2005, vol. 2005, 14 pp. DOI: 10.1109/WSC.2005.1574241.

[96] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter*, Feb. 29, 2020. DOI: 10.48550/arXiv.1910.01108.

[97] J. Schmidt-Hieber, "Nonparametric regression using deep neural networks with ReLU activation function," *The Annals of Statistics*, vol. 48, no. 4, Aug. 1, 2020. DOI: 10.1214/19-AOS1875.

[98] P. Schwab, L. Linhardt, and W. Karlen, *Perfect match: A simple method for learning representations for counterfactual inference with neural networks*, May 27, 2019. DOI: 10.48550/arXiv.1810.00656.

[99] SciKeras-Developers. "Welcome to SciKeras's documentation! — SciKeras 0.9.0 documentation." (2020), [Online]. Available: https://www.adriangb.com/scikeras/stable/ (visited on 01/27/2023).

[100] SellerApp. "Understanding the amazon best seller rank - SellerApp." (2023), [Online]. Available: https://www.sellerapp.com/amazon-best-seller-rank (visited on 02/28/2023).

[101] A. Sharma and E. Kiciman. "Tutorial on causal inference and counterfactual reasoning." (2018), [Online]. Available: https://causalinference.gitlab.io/kdd-tutorial/ (visited on 02/07/2023).

[102] A. Skrondal, "Design and analysis of monte carlo experiments: Attacking the conventional wisdom," *Multivariate Behavioral Research*, vol. 35, no. 2, pp. 137–167, Apr. 2000. DOI: 10.1207/S15327906MBR3502_1.

[103] J. Sloman, D. Garratt, J. Guest, and E. Jones, *Economics for business*, Seventh Edition. Harlow, United Kingdom: Pearson Education, 2016, 684 pp.

[104] J. A. Sokolowski and C. M. Banks, *Modeling and simulation for analyzing global events*. Hoboken, N.J: Wiley, 2009, 205 pp.

[105] X. Su, T. Huang, Y. Li, S. You, F. Wang, C. Qian, C. Zhang, and C. Xu, *Prioritized architecture sampling with monto-carlo tree search*, Mar. 22, 2021. DOI: 10.48550/arXiv.2103.11922.

[106] J. Terra. "Pytorch vs tensorflow vs keras: Here are the difference you should know," Simplilearn.com. (Sep. 22, 2022), [Online]. Available: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article (visited on 01/07/2023).

[107] Y. Tian and Y. Zhang, "A comprehensive survey on regularization strategies in machine learning," *Information Fusion*, vol. 80, pp. 146–166, Apr. 1, 2022. DOI: 10.1016/j.inffus.2021.11.005.

[108] M. Tietz, D. Nouri, and B. Bossan. "Skorch: A scikit-learn compatible neural network library that wraps PyTorch." (2017), [Online]. Available: https://skorch.readthedocs.io/en/stable/ (visited on 01/24/2023).

[109] A. J. Tiffin. "Machine learning and causality: The impact of financial crises on growth," IMF. (Nov. 1, 2019), [Online]. Available: https://www.imf.org/en/Publications/WP/Issues/2019/11/01/Machine-Learning-and-Causality-The-Impact-of-Financial-Crises-on-Growth-48722 (visited on 11/20/2022).

[110] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, *Well-read students learn better: On the importance of pre-training compact models*, Sep. 25, 2019. DOI: 10.48550/arXiv.1908.08962.

[111] A. Turrell. "Econometrics in python part i - double machine learning, markov wanderer: A blog on economics, science, coding and data." (2018), [Online]. Available: http://aeturrell.com/2018/02/10/econometrics-in-python-partI-ML/ (visited on 12/04/2022).

[112] Various-Authors. "Papers about causal inference and language." (Nov. 12, 2022), [Online]. Available: https://github.com/causaltext/causal-text-papers (visited on 11/12/2022).

[113] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, Dec. 5, 2017. DOI: 10.48550/arXiv.1706.03762.

[114] S. Voigt, F. Giulio-Tonolo, J. Lyons, J. Kučera, B. Jones, T. Schneiderhan, G. Platzeck, K. Kaku, M. K. Hazarika, L. Czaran, S. Li, W. Pedersen, G. K. James, C. Proy, D. M. Muthike, J. Bequignon, and D. Guha-Sapir, "Global trends in satellite-based emergency mapping," *Science*, vol. 353, no. 6296, pp. 247–252, Jul. 15, 2016. DOI: 10.1126/science.aad8728.

[115] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca, *AlphaX: eXploring neural architectures with deep neural networks and monte carlo tree search*, Oct. 1, 2019. DOI: 10.48550/arXiv.1903.11059.

[116] A. Weise and R. Levitan, *Decoupling entrainment from consistency using deep neural networks*, Nov. 3, 2020. DOI: 10.48550/arXiv.2011.01860.

[117] D. Westreich, J. Lessler, and M. J. Funk, "Propensity score estimation: Neural networks, support vector machines, decision trees (CART), and meta-classifiers as alternatives to logistic regression," *Journal of Clinical Epidemiology*, vol. 63, no. 8, pp. 826–833, Aug. 1, 2010. DOI: 10.1016/j.jclinepi.2009.11.020.

[118] A. Whata and C. Chimedza, "Evaluating uses of deep learning methods for causal inference," *IEEE Access*, vol. 10, pp. 2813–2827, 2022. DOI: 10.1109/ACCESS.2021.3140189.

[119] T. C. Williams, C. C. Bach, N. B. Matthiesen, T. B. Henriksen, and L. Gagliardi, "Directed acyclic graphs: A tool for causal studies in paediatrics," *Pediatric Research*, vol. 84, no. 4, pp. 487–493, Oct. 2018. DOI: 10.1038/s41390-018-0071-3.

[120] J. Winkel and T. Krebs. "Data generating process simulation: The opossum package." (Aug. 18, 2019), [Online]. Available: https://humboldt-wi.github.io/blog/research/applied_predictive_modeling_19/data_generating_process_blogpost// (visited on 12/04/2022).

[121] P. Xu, X. Zhu, and D. A. Clifton, *Multimodal learning with transformers: A survey*, Jun. 13, 2022. DOI: 10.48550/arXiv.2206.06488.

[122] L. Yao, Z. Chu, S. Li, Y. Li, J. Gao, and A. Zhang, "A survey on causal inference," *ACM Transactions on Knowledge Discovery from Data*, vol. 15, no. 5, 74:1–74:46, May 10, 2021. DOI: 10.1145/3444944.

[123] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022 022, Feb. 2019. DOI: 10.1088/1742-6596/1168/2/022022.

[124] Y. Yuan, X. Ding, and Z. Bar-Joseph, *Causal inference using deep neural networks*, Nov. 24, 2020. DOI: 10.48550/arXiv.2011.12508.

[125] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into deep learning*, Jul. 29, 2022. DOI: 10.48550/arXiv.2106.11342.

Appendix

List of Appendices

A Figures . . . XIV
A.1 Combined Histogram . . . XIV
A.2 Histograms for Exploratory Phase Scenario A03 . . . XV
A.3 Histograms for Exploratory Phase Scenario A9 . . . XVII
A.4 Histograms for all Scenarios in Exploratory Phase . . . XVIII
A.5 Boxplots for all Scenarios in Exploratory Phase . . . XVIII
A.6 Performance Metrics and Duration bert-tiny . . . XIX
A.7 Performance Metrics and Duration distilbert-base-uncased . . . XX
A.8 Performance Metrics and Duration beit-large . . . XXI
A.9 Performance Metrics and Duration tiny-vit-random . . . XXII
B Tables . . . XXIII
B.1 Overview Planning Steps . . . XXIII
B.2 Root Mean Squared Error for Partial Linear Regression Model Scenarios in Exploratory Phase . . . XXV
B.3 Average Standard Error for Partial Linear Regression Model Scenarios in Exploratory Phase . . . XXV
B.4 Root Mean Squared Error for Interactive Regression Model Scenarios in Exploratory Phase . . . XXVI
B.5 Empirical Deviation for Interactive Regression Model Scenarios in Exploratory Phase . . . XXVI
B.6 Average Standard Error for Interactive Regression Model Scenarios in Exploratory Phase . . . XXVII
C Code . . . XXVII
C.1 Minimal Code Example for DoubleML theoretical Foundations (Section 2.3.) . . . XXVII
C.2 Simulation Study Exploratory Phase - Source Code and Notebook XXVII
C.3 Simulation Study Detail Phase - Source Code and Notebooks . . . XXVII
C.4 Class TransformerEnsemble_from_pretrained . . . XXVIII
C.5 Text and Image Model Testing . . . XXVIII
C.6 Case Study Toys Data - Code and Notebooks . . . XXVIII
C.7 Case Study Chairs Data - Code and Notebooks . . . XXIX
C.8 SLURM output of the training progress of class CrossEmbeddings XXIX
C.9 DMLSim Library . . . XXXI
D Data . . . XXXI
D.1 Log Files Simulation Study Exploratory Phase . . . XXXI
D.2 Log Files Simulation Study Detail Phase . . . XXXI
D.3 Keepa Raw Data . . . XXXI
E Proofs . . . XXXI
E.1 Regression Coefficient . . . XXXI

GitHub Repository

In addition to the formal digital appendix, the GitHub repository contains the `DMLSim` library as well as links to the most relevant notebooks used in the case study for an easy access.

A Figures

A.1 Combined Histogram

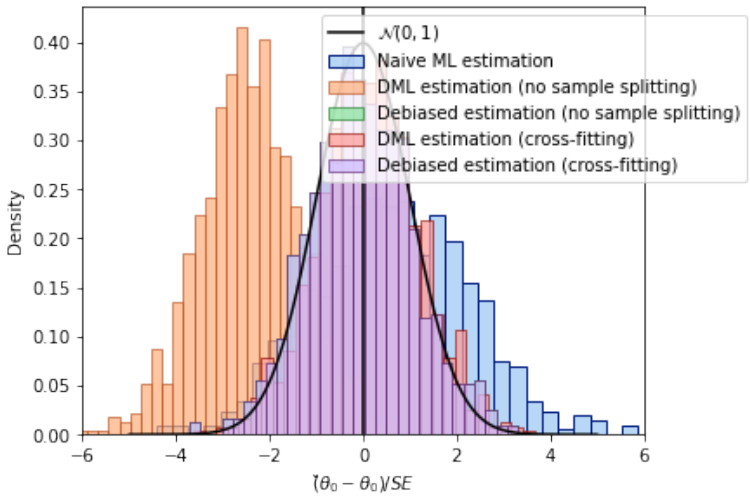
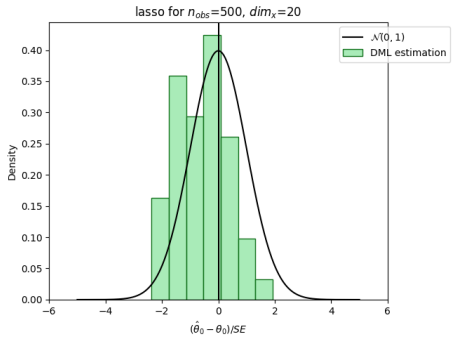


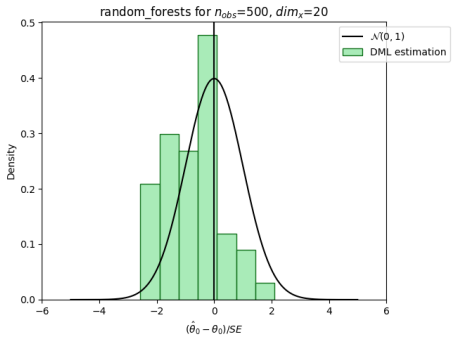
Figure 56: Combined Histogram with different Estimation Approaches

Source: own figure.

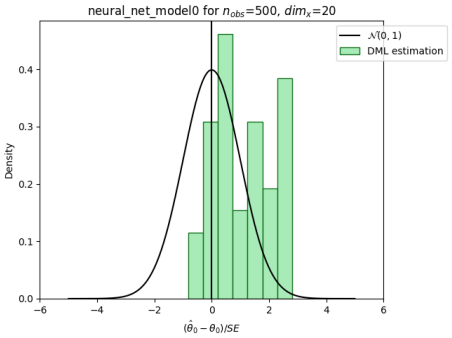
A.2 Histograms for Exploratory Phase Scenario A03



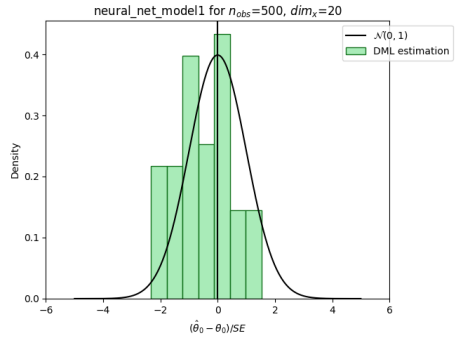
(a) Lasso



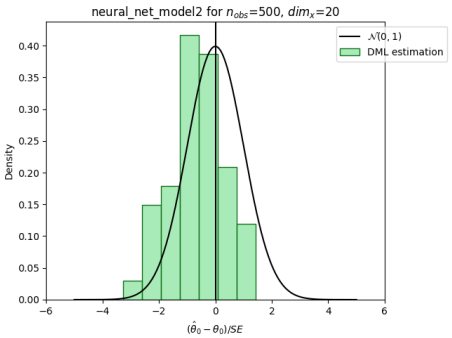
(b) Random Forest



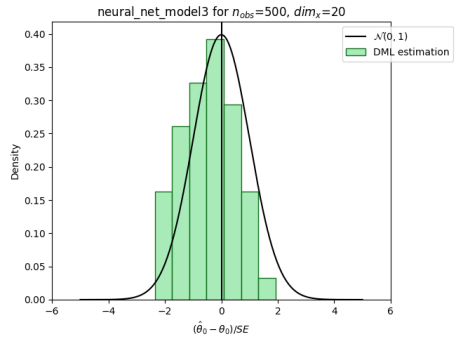
(c) Neural Network Model 0



(d) Neural Network Model 1



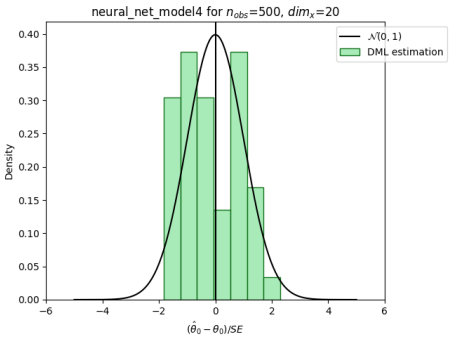
(e) Neural Network Model 2



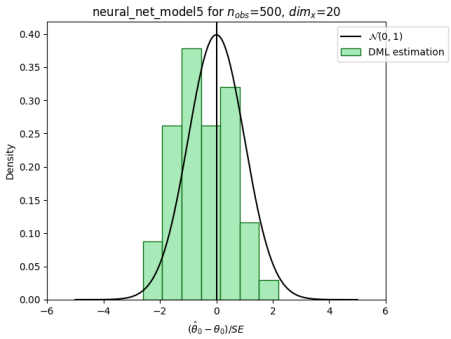
(f) Neural Network Model 3

Figure 57: Histograms for Exploratory Phase Scenario A3 - Part 1

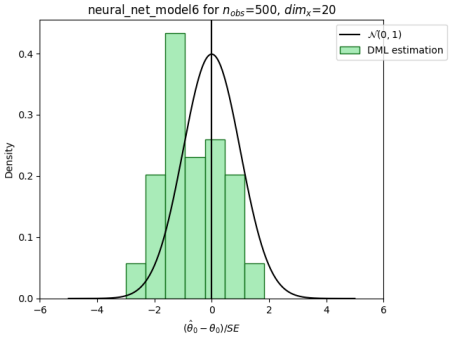
Source: own figure.



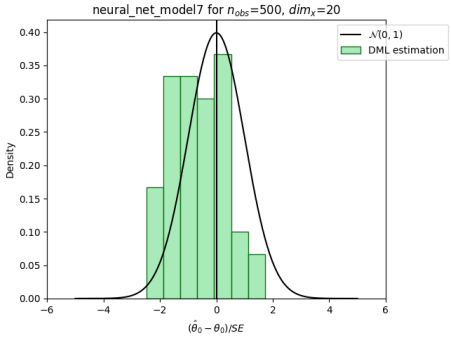
(a) Neural Network Model 4



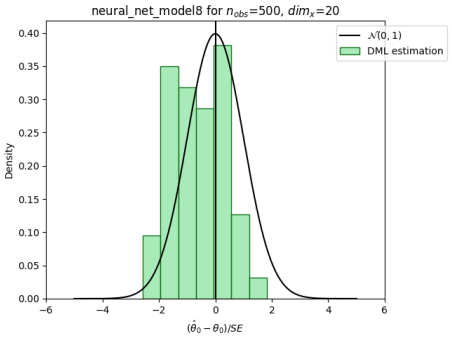
(b) Neural Network Model 5



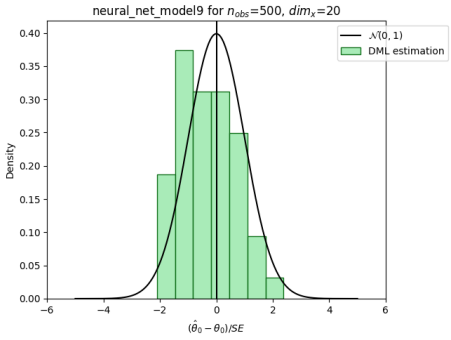
(c) Neural Network Model 6



(d) Neural Network Model 7



(e) Neural Network Model 8

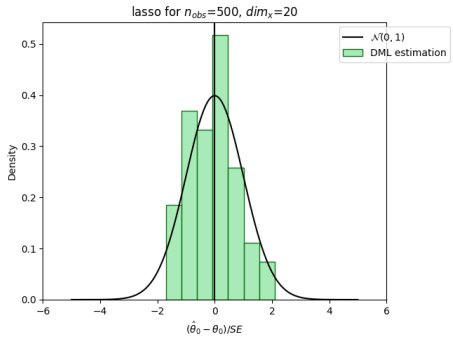


(f) Neural Network Model 9

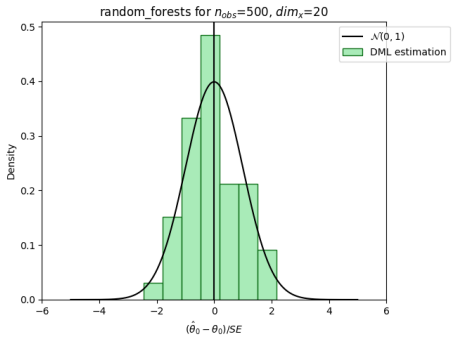
Figure 58: Histograms for Exploratory Phase Scenario A3 - Part 2

Source: own figure.

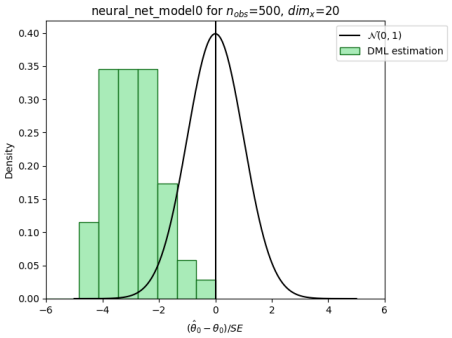
A.3 Histograms for Exploratory Phase Scenario A9



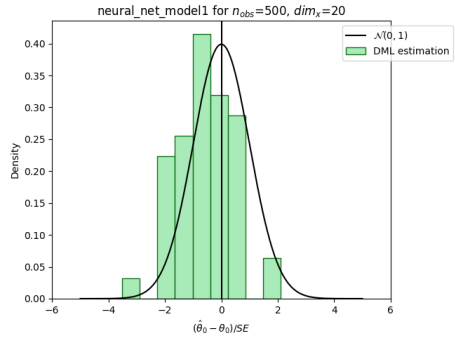
(a) Lasso



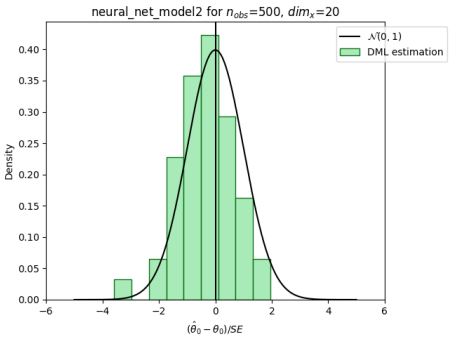
(b) Random Forest



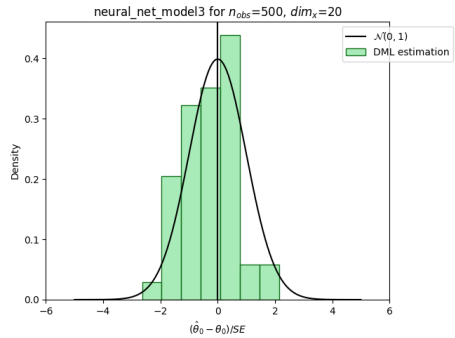
(c) Neural Network Model 0



(d) Neural Network Model 1



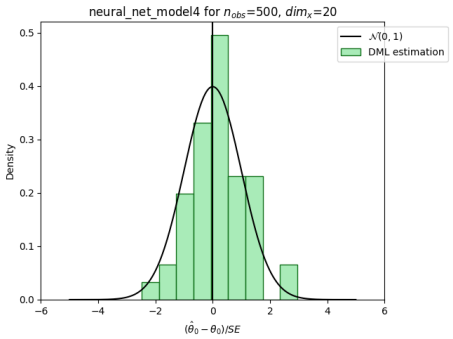
(e) Neural Network Model 2



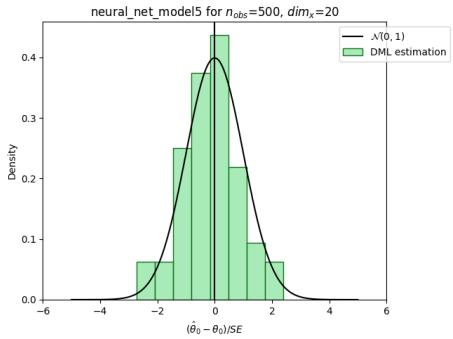
(f) Neural Network Model 3

Figure 59: Histograms for Exploratory Phase Scenario A9 - Part 1

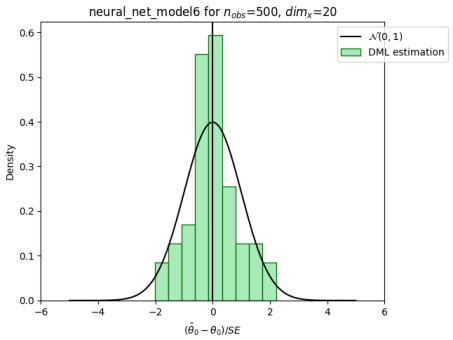
Source: own figure.



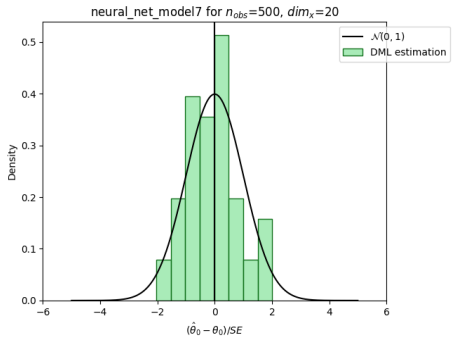
(a) Neural Network Model 4



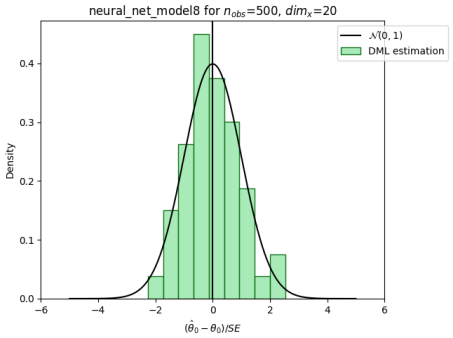
(b) Neural Network Model 5



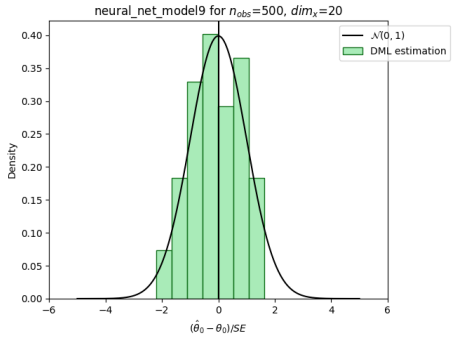
(c) Neural Network Model 6



(d) Neural Network Model 7



(e) Neural Network Model 8



(f) Neural Network Model 9

Figure 60: Histograms for Exploratory Phase Scenario A9 - Part 2

Source: own figure.

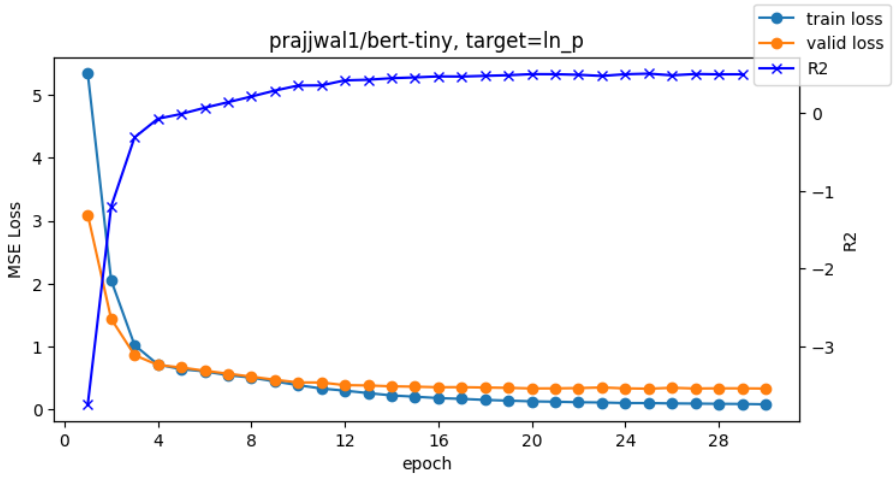
A.4 Histograms for all Scenarios in Exploratory Phase

The histograms for each considered method for all scenarios of the exploratory phase of the simulation study are available in the digital appendix.

A.5 Boxplots for all Scenarios in Exploratory Phase

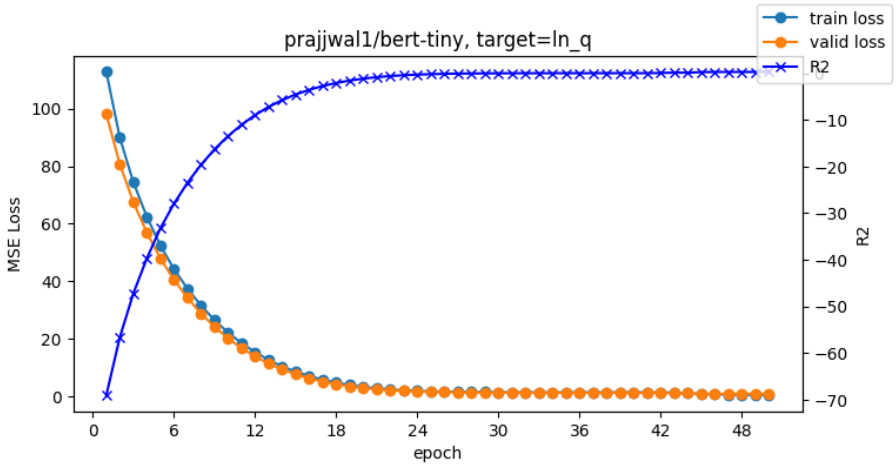
The boxplots for each considered method for all scenarios of the exploratory phase of the simulation study are available in the digital appendix.

A.6 Performance Metrics and Duration bert-tiny



(a) bert-tiny on ln p
Stopping since valid_loss has not improved in the last 5 epochs.
CPU times: user 1min 52s, sys: 2.1 s, total: 1min 54s

Wall time: 2min 8s

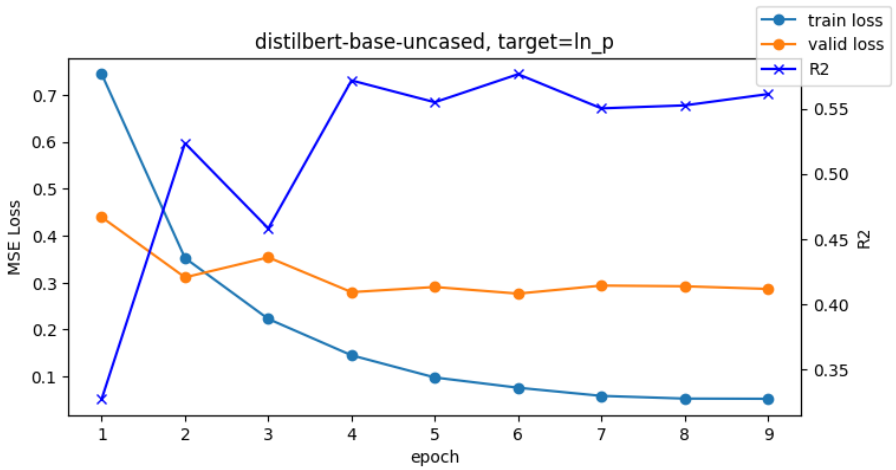


(b) bert-tiny on ln q
CPU times: user 3min 7s, sys: 1.14 s, total: 3min 9s

Wall time: 3min 13s

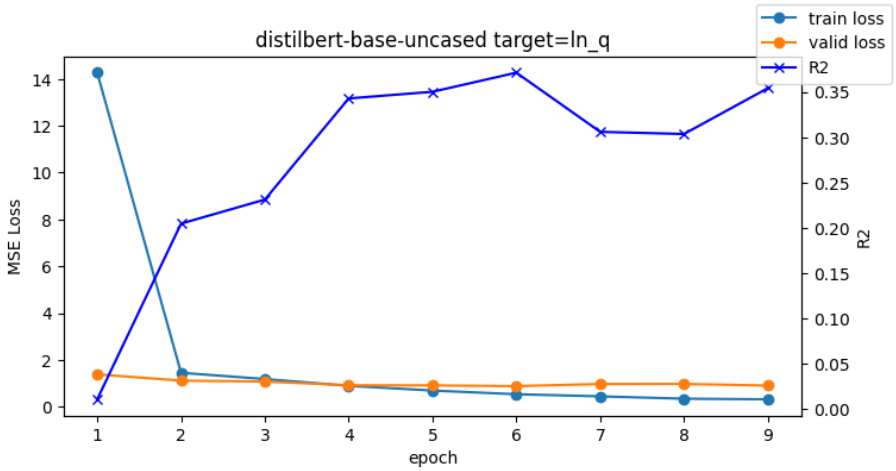
Figure 61: Performance Metrics and Duration bert-tiny
Source: own figure.

A.7 Performance Metrics and Duration distilbert-base-uncased



(a) distilbert-base-uncased on ln p
Stopping since valid_loss has not improved in the last 3 epochs.
CPU times: user 10min 43s, sys: 2.57 s, total: 10min 46s

Wall time: 10min 55s



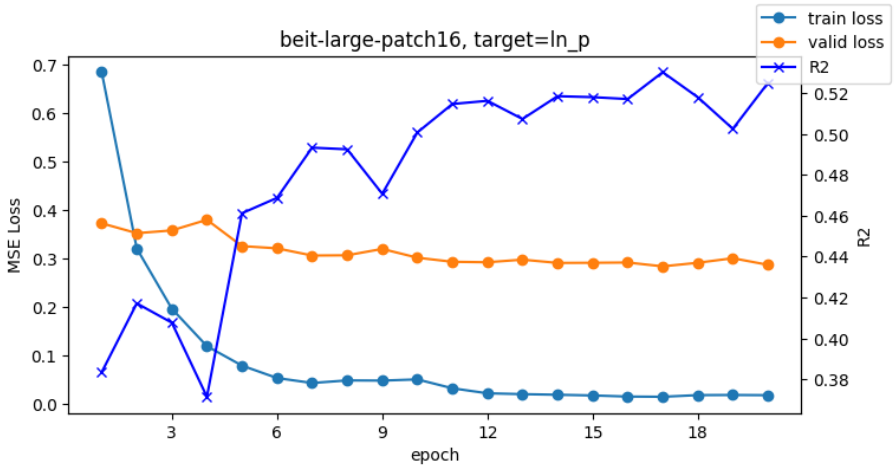
(b) distilbert-base-uncased on ln q
Stopping since valid_loss has not improved in the last 3 epochs.
CPU times: user 10min 42s, sys: 2.05 s, total: 10min 44s

Wall time: 10min 49s

Figure 62: Performance Metrics and Duration distilbert-base-uncased

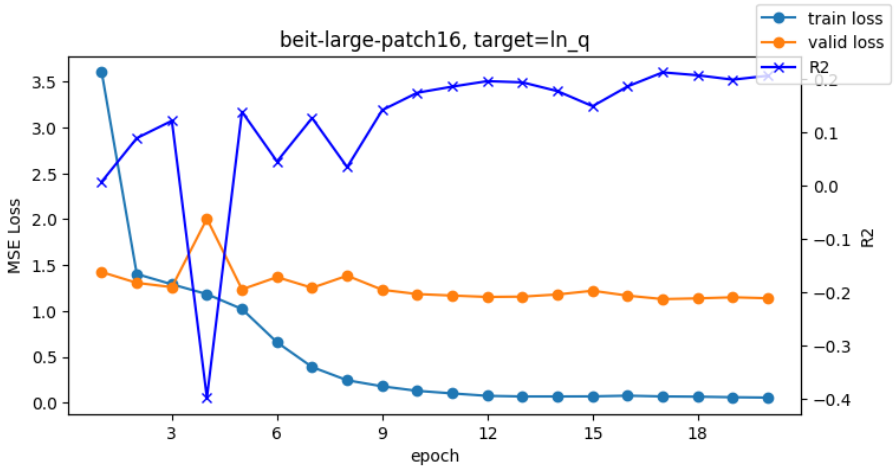
Source: own figure.

A.8 Performance Metrics and Duration beit-large



(a) beit-large-patch16 on ln p
CPU times: user 42min 23s, sys: 15min 37s, total: 58min

Wall time: 55min 51s



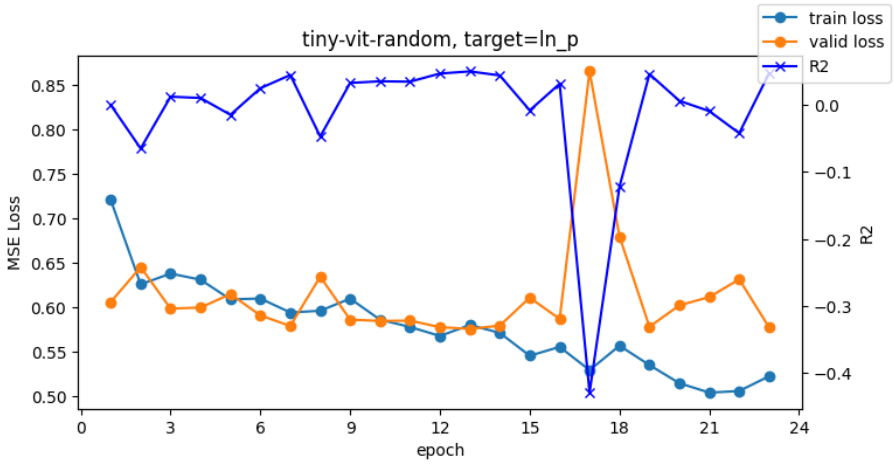
(b) beit-large-patch16 on ln q
CPU times: user 42min 49s, sys: 16min 7s, total: 58min 56s

Wall time: 55min 43s

Figure 63: Performance Metrics and Duration beit-large-patch16

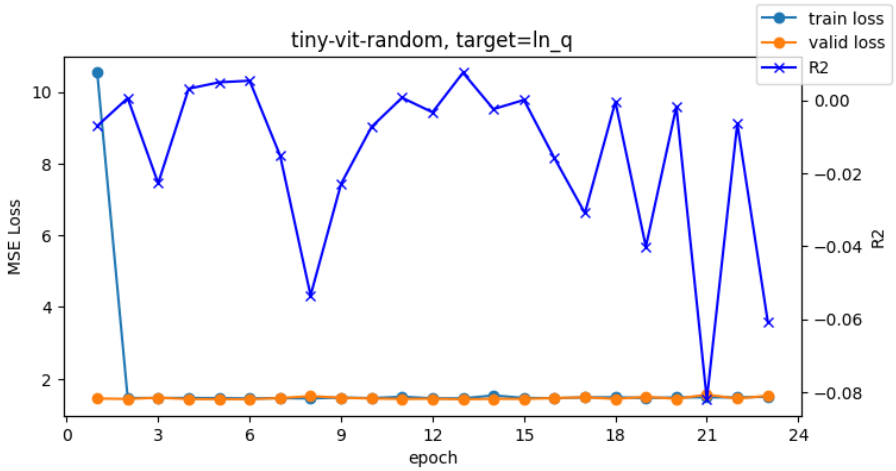
Source: own figure.

A.9 Performance Metrics and Duration tiny-vit-random



(a) tiny-vit-random on ln p
Stopping since valid_loss has not improved in the last 10 epochs.
CPU times: user 3min 29s, sys: 11.3 s, total: 3min 40s

Wall time: 1min 17s



(b) tiny-vit-random on ln q
Stopping since valid_loss has not improved in the last 10 epochs.
CPU times: user 3min 25s, sys: 11.2 s, total: 3min 36s

Wall time: 1min 15s

Figure 64: Performance Metrics and Duration tiny-vit-random

Source: own figure.

B Tables

B.1 Overview Planning Steps

Table 34: Overview of Key Steps in Planning of the Simulation Study
Source: own table.

Step	Content	Section	Design for this thesis
Objectives	Objectives to be primarily examined by the study	4.2.1	• Proof the successful implementation of the DL approaches as learners in the DML framework • Analyse use and and finite sample properties of DL networks in semiparametric causal inference [26] • Compare DL approaches to ML methods
DGP	Parametric models for data generation	4.2.2	• Look at twelve different DGPs during the exploratory phase • Select a DGP where the use of DL methods seems promising in the context of CML for detail phase
DGP - factors to vary	Factors to vary during the study	4.2.2	• n_{obs} • dim_x
DGP - factor levels	Levels in which the factors shall be varied	4.2.2	• $n_{obs} = \{500; 1000; 10000\}$ • $dim_x = \{20; 100\}$ • Base case for exploratory phase: $n_{obs} = 500$ and $dim_x = 20$

Step	Content	Section	Design for this thesis
DGP - factorial design	Design of the study with regards to factor variation	4.2.2	• Full factorial design
Estimands and targets	Estimand/ parameter of interest and targets of the study	4.2.3	• Causal parameter θ_0
Methods	Methods to include in study	4.2.4	• Causal models: – PLR models – IRM • ML methods: – Lasso – Random Forest • Network architectures: 10 network architectures with different depths and widths
Performance measures	Measures to assess performance of methods	4.2.5	• Absolute Bias • Standardized Bias • Relative Bias • RMSE • Average SE • ED • Coverage of CI
Number of simulation runs	Number of repetitions	4.2.6	• Exploratory phase: $n_{rep} = 50$ • Detail phase: $n_{rep} = 1000$

B.2 Root Mean Squared Error for Partial Linear Regression Model Scenarios in Exploratory Phase

For reasons of space the neural networks models are abbreviated as nn_0 to nn_9 and Random Forest as rf.

Table 35: Root Mean Squared Error for Partial Linear Regression Model Scenarios in Exploratory Phase

Source: own table.

	A01	A03	A05	A07	A08
lasso	0.0407	0.0519	0.3652	0.1863	0.1609
rf	0.053	0.0584	0.2554	0.1693	0.1554
nn_0	0.203	0.0734	0.9389	0.4232	0.8867
nn_1	0.1607	0.0513	0.9714	0.264	0.2558
nn_2	0.0509	0.056	1.0904	0.1788	0.2421
nn_3	0.0832	0.0498	0.8777	0.1981	0.2255
nn_4	0.0731	0.0471	0.8793	0.2318	0.2023
nn_5	0.0503	0.0525	0.3053	0.1772	0.2082
nn_6	0.0691	0.06	0.1081	0.189	0.1985
nn_7	0.0608	0.0541	0.1893	0.1742	0.2156
nn_8	0.0989	0.0522	0.8007	0.1993	0.222
nn_9	0.0544	0.0492	0.6034	0.1848	0.2328

B.3 Average Standard Error for Partial Linear Regression Model Scenarios in Exploratory Phase

Table 36: Average Standard Error for Partial Linear Regression Model Scenarios in Exploratory Phase

Source: own table.

	A01	A03	A05	A07	A08
Lasso	0.044	0.047	0.056	0.170	0.137
Random Forest	0.045	0.047	0.085	0.156	0.129
Neural Net Model 0	0.039	0.048	0.024	0.192	0.178
Neural Net Model 1	0.042	0.048	0.028	0.192	0.178
Neural Net Model 2	0.043	0.047	0.037	0.171	0.141
Neural Net Model 3	0.041	0.047	0.030	0.170	0.140
Neural Net Model 4	0.038	0.047	0.034	0.171	0.145
Neural Net Model 5	0.043	0.047	0.053	0.166	0.138
Neural Net Model 6	0.046	0.047	0.057	0.162	0.138
Neural Net Model 7	0.045	0.047	0.050	0.164	0.139
Neural Net Model 8	0.039	0.047	0.049	0.164	0.135
Neural Net Model 9	0.039	0.047	0.057	0.168	0.144

B.4 Root Mean Squared Error for Interactive Regression Model Scenarios in Exploratory Phase

For reasons of space the neural networks models are abbreviated as nn_0 to nn_9 and Random Forest as rf.

Table 37: Root Mean Squared Error for Interactive Regression Model Scenarios in Exploratory Phase

Source: own table.

	A02	A04	A06	A09	A10	A11	A12
lasso	0.1795	0.2948	0.1089	0.3164	0.2524	0.3551	0.2967
rf	0.1576	0.2612	0.1229	0.1663	0.2492	0.161	0.2619
nn_0	0.2475	0.9563	5.1837	0.7112	1.1727	0.7079	1.0628
nn_1	0.3713	1.7784	1.49	0.2238	0.5394	0.2166	0.5106
nn_2	0.337	1.6868	1.1002	0.1876	0.4473	0.1847	0.4142
nn_3	0.3675	1.984	1.3239	0.1875	0.4894	0.1927	0.4662
nn_4	0.3877	1.9884	1.03	0.1825	0.4178	0.1862	0.4056
nn_5	0.2737	0.8804	0.5174	0.1728	0.3839	0.1826	0.3424
nn_6	0.2222	0.4426	0.2211	0.1539	0.2969	0.1811	0.2446
nn_7	0.2518	0.4459	0.2425	0.1571	0.3407	0.1924	0.2894
nn_8	0.3447	0.886	0.462	0.1665	0.4083	0.1594	0.3865
nn_9	0.2976	0.7798	0.4892	0.1661	0.3637	0.1648	0.3441

B.5 Empirical Deviation for Interactive Regression Model Scenarios in Exploratory Phase

Table 38: Empirical Deviation for Interactive Regression Model Scenarios in Exploratory Phase

Source: own table.

	A02	A04	A06	A09	A10	A11	A12
Lasso	0.179	0.299	0.108	0.107	0.253	0.356	0.300
Random Forest	0.146	0.263	0.123	0.052	0.184	0.161	0.189
Neural Net Model 0	0.155	0.907	3.850	0.057	0.293	0.244	0.266
Neural Net Model 1	0.142	1.369	1.411	0.049	0.214	0.172	0.229
Neural Net Model 2	0.135	1.316	1.079	0.057	0.175	0.166	0.202
Neural Net Model 3	0.141	1.190	1.031	0.050	0.220	0.182	0.219
Neural Net Model 4	0.141	1.190	1.031	0.050	0.220	0.182	0.218
Neural Net Model 5	0.130	0.780	0.503	0.060	0.186	0.180	0.192
Neural Net Model 6	0.150	0.406	0.217	0.076	0.195	0.181	0.188
Neural Net Model 7	0.146	0.445	0.218	0.070	0.197	0.191	0.192
Neural Net Model 8	0.139	0.880	0.437	0.048	0.194	0.151	0.199
Neural Net Model 9	0.141	0.787	0.482	0.062	0.176	0.160	0.197

B.6 Average Standard Error for Interactive Regression Model Scenarios in Exploratory Phase

Table 39: Average Standard Error for Interactive Regression Model Scenarios in Exploratory Phase

Source: own table.

	A02	A04	A06	A09	A10	A11	A12
Lasso	0.149	0.260	0.112	0.237	0.249	0.333	0.237
Random Forest	0.116	0.272	0.160	0.150	0.149	0.170	0.143
Neural Net Model 0	0.128	1.430	0.718	0.236	0.208	0.213	0.200
Neural Net Model 1	0.111	1.373	0.704	0.208	0.184	0.187	0.174
Neural Net Model 2	0.105	1.250	0.586	0.187	0.161	0.176	0.153
Neural Net Model 3	0.111	1.259	0.692	0.209	0.186	0.189	0.176
Neural Net Model 4	0.109	1.100	0.504	0.194	0.171	0.181	0.163
Neural Net Model 5	0.105	0.763	0.282	0.161	0.157	0.171	0.152
Neural Net Model 6	0.108	0.413	0.156	0.164	0.158	0.177	0.152
Neural Net Model 7	0.108	0.474	0.191	0.169	0.163	0.178	0.157
Neural Net Model 8	0.107	0.768	0.260	0.167	0.164	0.172	0.160
Neural Net Model 9	0.110	0.628	0.296	0.189	0.173	0.186	0.166

C Code

C.1 Minimal Code Example for DoubleML theoretical Foundations (Section 2.3.)

The interactive Python notebook which examines the theoretical foundations of the DML framework as presented in section 2.3. is available in the digital appendix.

C.2 Simulation Study Exploratory Phase - Source Code and Notebook

The Python source code for all scenarios executed in the exploratory phase of the simulation study and the batch files are available in the digital appendix. Additionally, the notebook summarizing and presenting the resulting performance measures is included in the digital appendix.

C.3 Simulation Study Detail Phase - Source Code and Notebooks

The Python source code and shell scripts for all scenarios executed in the detail phase of the simulation study are available in the digital appendix. Additionally, the notebook summarizing and presenting the resulting performance measures is included in the digital appendix `Simulation_Summary_performance_review_Detail.ipynb`). Besides, the notebook presenting all boxplots and histograms for each scenario of the detail phase is included in the digital appendix (`Simulation_Summary_attributes_review_Detail.ipynb`).

C.4 Class TransformerEnsemble_from_pretrained

```
import torch
from torch import nn
class TransformerEnsemble_from_pretrained(nn.Module):
    def __init__(self, image_model, text_model, num_labels):
        super().__init__()
        self.image_model = image_model
        self.text_model = text_model
        self.dropout = nn.Dropout(0.1)
        self.relu = nn.ReLU()
        self.pre_classifier = nn.LazyLinear(768)
        self.feature_out = nn.LazyLinear(128)
        self.classifier = nn.LazyLinear(1)

    def forward(self, input_ids, attention_mask, features):
        with torch.no_grad():
            text_features = self.text_model(input_ids, attention_mask)
            image_features = self.image_model(features)
        output_text = text_features[1]
        output_image = image_features[1]
        output_cat = torch.cat([output_text,
                                output_image],
                                dim=1)
        output_cat = self.pre_classifier(output_cat)
        output_cat = self.relu(output_cat)
        pooled_output = self.dropout(output_cat)
        features = self.feature_out(pooled_output)
        logits = self.classifier(features)
        return logits, features
```

C.5 Text and Image Model Testing

The notebooks for testing various text an image models on the toys data are available in the digital appendix.

C.6 Case Study Toys Data - Code and Notebooks

The notebooks for the descriptive statistics of the toys dataset and the first as well as the full data analysis of the toys data are available in the digital appendix. Additionally, the Python source code and shell scripts for fine tuning the models according to the presented case study are included in the digital appendix.

C.7 Case Study Chairs Data - Code and Notebooks

The notebooks for the descriptive statistics of the chairs dataset and the first as well as the full data analysis of the chairs data are available in the digital appendix. Additionally, the Python source code and shell scripts for fine tuning the models according to the presented case study are included in the digital appendix.

C.8 SLURM output of the training progress of class CrossEmbeddings

```
module: loaded site/slurm
module: loaded site/tmpdir
module: loaded site/hummel
module: loaded env/system-gcc
module: unloaded env/system-gcc
module: loaded python/3.9.12
Currently Loaded Modulefiles:
 1) site/slurm      2) site/tmpdir     3) site/hummel
4) python/3.9.12
/work/fwwx329/CaseStudy
Mon Apr  3 14:17:21 CEST 2023
...
embedding generation for amazon toy data
Fold 1 from 3
 Fit target aux model
  epoch       r2     train_loss     valid_loss       dur
-------  ------  -------------  -------------  --------
      1  0.0680         7.4245         1.3155  675.8078
      2  0.0710         1.3580         1.3112  674.9939
      3  0.2807         1.0258         1.0152  675.6176
      4  0.2558         0.6662         1.0504  675.3550
      5  0.2443         0.4221         1.0666  675.4943
      6  0.3164         0.2936         0.9649  675.5006
      7  0.3113         0.2228         0.9721  675.2511
      8  0.3236         0.1930         0.9547  675.4126
      9  0.3099         0.1841         0.9741  675.4465
     10  0.3196         0.1690         0.9604  675.5776
Restoring best model from epoch 8.
 Fit treatment aux model
  epoch       r2     train_loss     valid_loss       dur
-------  ------  -------------  -------------  --------
      1  0.4387         0.5942         0.3591  644.9224
      2  0.4774         0.2746         0.3344  643.7037
      3  0.5069         0.1451         0.3155  644.3637
      4  0.5752         0.0829         0.2718  644.0196
      5  0.5758         0.0607         0.2714  643.9879
      6  0.5808         0.0507         0.2682  643.7724
      7  0.5778         0.0431         0.2702  644.0115
      8  0.5950         0.0375         0.2591  643.8397
Fold 2 from 3
 Fit target aux model
  epoch       r2     train_loss     valid_loss       dur
```

```
-------  ------  ------------  ------------  --------
      1  0.0281        6.6091        1.4235  680.0088
      2  0.2990        1.2719        1.0268  679.2622
      3  0.3708        0.9633        0.9216  679.3690
      4  0.4208        0.6486        0.8483  679.1532
      5  0.4009        0.4223        0.8774  679.6242
      6  0.3836        0.2980        0.9029  679.6403
Stopping since valid_loss has not improved in the last 3 epochs.
Restoring best model from epoch 4.
 Fit treatment aux model
  epoch      r2    train_loss    valid_loss       dur
-------  ------  ------------  ------------  --------
      1  0.4014        0.5958        0.3904  644.0346
      2  0.5453        0.2648        0.2966  643.2673
      3  0.5962        0.1391        0.2634  643.2570
      4  0.5989        0.0833        0.2617  643.0964
      5  0.6051        0.0591        0.2576  642.9782
      6  0.6204        0.0478        0.2476  643.1560
      7  0.6081        0.0408        0.2556  642.7866
      8  0.6127        0.0375        0.2526  643.0404
Restoring best model from epoch 6.
Fold 3 from 3
 Fit target aux model
  epoch      r2    train_loss    valid_loss       dur
-------  ------  ------------  ------------  --------
      1  0.0330        7.0875        1.3555  679.7668
      2  0.2633        1.2745        1.0326  678.9814
      3  0.3205        0.9787        0.9524  678.8707
      4  0.3345        0.6542        0.9327  678.4923
      5  0.2952        0.4040        0.9879  678.9221
      6  0.3870        0.2995        0.8592  678.7600
      7  0.3650        0.2248        0.8900  678.7552
      8  0.4105        0.2050        0.8262  678.8354
      9  0.4119        0.1692        0.8243  679.0095
     10  0.3558        0.1678        0.9030  679.0367
Restoring best model from epoch 9.
 Fit treatment aux model
  epoch      r2    train_loss    valid_loss       dur
-------  ------  ------------  ------------  --------
      1  0.3866        0.6225        0.4145  643.2958
      2  0.5567        0.2836        0.2996  642.6908
      3  0.5833        0.1509        0.2816  642.8170
      4  0.6109        0.0884        0.2629  642.6679
      5  0.5674        0.0634        0.2923  643.0559
      6  0.6384        0.0448        0.2443  642.9019
      7  0.6421        0.0432        0.2419  642.7104
      8  0.6343        0.0368        0.2472  643.0147
Restoring best model from epoch 7.
Finished
Node node334
Current TMPDIR: /scratch/fwwx329.3043469
```

C.9 DMLSim Library

The `DMLSim` Library is available in the digital appendix and in the linked GitHub repository.

D Data

D.1 Log Files Simulation Study Exploratory Phase

The log files containing all relevant information like model attributes, raw estimates, performance measures, and simulation specifications for each scenario of the exploratory phase of the simulation study are available in the digital appendix.

D.2 Log Files Simulation Study Detail Phase

The log files containing all relevant information like model attributes, raw estimates, performance measures, and simulation specifications for each scenario of the detail phase of the simulation study are available in the digital appendix.

For scenarios B03, B06, B09, and B12 the folder contains multiple log files which is due to the fact that those scenarios were executed batchwise.

D.3 Keepa Raw Data

The raw data of Amazon sales data for toys and chairs collected by the Keepa software tool is available in the digital appendix.

E Proofs

E.1 Regression Coefficient

Given a linear regression model[55]:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad \text{for} \quad i = 1, \cdots, n \tag{93}$$

where $\boldsymbol{y} = (y_1, \cdots, y_n)'$, $\boldsymbol{x} = (x_1, \cdots, x_n)'$ and $\boldsymbol{\beta} = (\beta_0, \beta_1)$ with a design matrix $\boldsymbol{X} = [\mathbf{1}_n, \boldsymbol{x}]$. So the best linear estimator $\widehat{\boldsymbol{\beta}}$ can be calculated as:

$$\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{X}'\boldsymbol{X}\right)^{-1}\boldsymbol{X}'\boldsymbol{y} \tag{94}$$

$$= \frac{1}{n\sum_{i=1}^{n} x_i^2 - n^2\bar{x}^2} \begin{pmatrix} \sum_{i=1}^{n} x_i^2 & -n\bar{x} \\ -n\bar{x} & n \end{pmatrix} \begin{pmatrix} n\bar{y} \\ \sum_{i=1}^{n} x_i y_i \end{pmatrix} \tag{95}$$

Therefore, the regression coefficient can be written as:

$$\widehat{\beta}_1 = \frac{n\sum_{i=1}^{n} x_i y_i - n^2\bar{x}\bar{y}}{n\sum_{i=1}^{n} x_i^2 - n^2\bar{x}^2} = \frac{\frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{n-1}}{\frac{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}{n-1}} = \frac{cov(x,y)}{var(x)} \tag{96}$$

Versicherung über die Selbstständigkeit - Sundermann, Moritz

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Masterarbeit, entsprechend der Kapitelzuweisung (Allocation of Chapters), ohne fremde Hilfe selbstständig verfasst habe. Ich habe keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellverzeichnis nicht benannten Internet-Quellen – benutzt. Ich habe die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht. Die schriftliche Fassung entspricht der auf dem elektronischen Speichermedium.

Hamburg, den 02.05.2023	
Ort, Datum	Unterschrift

Versicherung über die Selbstständigkeit - Teichert-Kluge, Jan

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Masterarbeit, entsprechend der Kapitelzuweisung (Allocation of Chapters), ohne fremde Hilfe selbstständig verfasst habe. Ich habe keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellverzeichnis nicht benannten Internet-Quellen – benutzt. Ich habe die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht. Die schriftliche Fassung entspricht der auf dem elektronischen Speichermedium.

Hamburg, den 02.05.2023	
Ort, Datum	Unterschrift