Concept Drift in Recommender Systems: Building a MLOps Pipeline that takes Concept Drift into account

**Bachelorarbeit**

im Studiengang  
Wirtschaftsinformatik und digitale Medien

vorgelegt von

**Sebastian Sätzler**Matr.-Nr.: 37635

am 18. Juli 2022   
an der Hochschule der Medien Stuttgart

Erstprüfer/in: Prof. Dr. Jan Kirenz   
Zweitprüfer/in: Prof. Dr. Hendrik Meth

# Ehrenwörtliche Erklärung

„Hiermit versichere ich, Sebastian Sätzler, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit (bzw. Masterarbeit) mit dem Titel: „Concept Drift in Recommender Systems: Building a MLOps Pipeline that takes Concept Drift into account“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Aichschieß, 30.4.2022,



# Kurzfassung

Gegenstand

**Schlagwörter**: Pipeline, MLOps, Konzeptdrift, Empfehlungsdienst, Maschinelles Lernen

# Abstract

A style sheet

**Keywords:** Pipeline, MLOps, Concept Drift, Recommender System, Machine Learning

# Table of Contents

Ehrenwörtliche Erklärung 2

Kurzfassung 3

Abstract 3

Table of Contents 4

Table of Figures 6

List of Tables 6

List of Abbreviations 7

1 Introduction 9

1.1 Motivation 9

1.2 Research Methodology 10

2 Current Environment & State of Research 13

2.1 Environment 13

2.2 Recommender Systems 14

2.2.1 Overview 14

2.2.2 Data 16

2.2.3 Content-based recommender systems 17

2.2.4 Collaborative filtering 18

2.2.5 Deep & Cross Networks 22

2.2.6 State of the Art Technology 24

2.3 MLOps 25

2.3.1 Problem 25

2.3.2 What is MLOps 27

2.3.3 Maturity Levels 29

2.3.4 Concept Drift 31

2.3.5 State of the Art Technology 35

3 Goal & Specification 38

3.1 Artifact 38

3.2 Procedure 39

3.3 Challenges 41

4 Design & Development 42

4.1 Hardware & Software Environment 42

4.2 Recommender System 43

4.2.1 Data 43

4.2.2 Tools 43

4.2.3 Implementation 43

4.2.4 Challenges 43

4.3 Concept Drift Awareness 43

4.4 MLOps Pipeline 43

5 Evaluation 44

6 Conclusion 45

6.1 Design Science Research 45

6.2 Recommender Systems 45

6.2.1 Overview 45

7 Conclusion and Outlook 46

Anhang A: Beispiele für die Gliederung von Abschlussarbeiten 47

A.1 Literaturarbeiten 47

A.2 Systementwicklungen 47

Anhang B: Formatvorlagen 48

Index 49

# Table of Figures

Figure 1: structure and overview of RSs derived from Aggarwal (2016) 16

Figure 2: Visualization of a cross layer (Wang et al., 2021) 23

Figure 3: Possible DCN architectures (Wang et al., 2021) 24

Figure 4: Levels of Microsoft's (left) and Google's (right) maturity model (Google LLC, 2020; Microsoft, n.d.) 31

Figure 5: Types of concept drift (Lu et al., 2018) 33

Figure 6: Framework for handling concept drift in machine learning (Lu et al., 2018) 33

Figure 7: TFX components (Google LLC, 2019) 35

Figure 8: procedure of artifact development 40

# List of Tables

Table 1: Artifact specification table 38

# List of Abbreviations

ML Machine Learning

GPU Graphics Processing Unit

CPU Central Processing Unit

RS Recommender System

CD Concept Drift

TFX TensorFlow Extended

TFRS TensorFlow Recommenders

DCN Deep & Cross Network

DD Data Drift

STEM Academic disciplines of Science, Technology, Engineering, Mathematics

PoC Proof of Concept

IS Information System

SotA State of the Art

AI Artificial Intelligence

API Application Programming Interface

CF Collaborative Filtering

CB Content-Based Recommender System

TF-IDF Term Frequency-Inverse Document Frequency

MF Matrix Factorization

DL Deep Learning

DNN Deep Neural Networks

NN Neural Network

ANN Artificial Neural Network

SVD Singular Value Decomposition

FM Factorization Model

NLP Natural Language Processing

ReLU Rectified Linear Unit

QPS Queries per Second

CACE Changing Anything Changes Everything

CI/CD Continuous Integration Continuous Delivery

EDA Exploratory Data Analysis

# Introduction

## Motivation

Over the last two decades Machine Learning (ML) has become one of the fastest growing technical fields with an estimated growth of 21% in 2022 compared to its previous year (Rimol, 2021).

It managed to secure a position as one of the top fields in computer science for scientific research as well as enterprise adoption. ML combines concepts of linear algebra and statistics and applies them to large datasets to find patterns and generalizations in the data, which can be used to make predictions or classifications.

Leveraging these complex algorithms with the computational power of modern GPUs and CPUs, ML has seen application in a large variety of sectors ranging from medicine for diagnostics, to transportation for self-driving cars and e-commerce for shopping cart optimization (Choy et al., 2018). The latter sector employs so called Recommender Systems (RS) with the goal of suggesting products that coincide with the taste of the customer. With the advent of e-commerce, RSs have gained increasing interest from academia and especially the enterprise sector (Singh, Choudhury, Dey, & Pramanik, 2021). RSs serve a major role for large tech corporations in engaging, retaining and enticing the user-base of their platform (Jannach & Zanker, 2022). Netflix for example uses its own RS to suggest its users what movies they might be interested in. In order to incentivize research in the field of Recommender Systems, Netflix introduced the Netflix Prize in 2007: A dataset was made public with users and their movie ratings. The goal was to create a RS that would beat Netflix’ own RS at that time. For this challenge a prize pool of $1.000.000 was written out (Bennett, Lanning, & others, 2007). To this day datasets of movie ratings remain a popular way to benchmark RSs.

Despite the wide use and success of Recommender Systems and Machine Learning in general, it still is a relatively new field with a lot of research opportunity (Jordan & Mitchell, 2015). While Recommender Systems are considered integral to many online-platforms, their precision and accuracy often lack in comparison to other ML fields. This is, among other things, due to the nature of the data that Recommender Systems work with, which is often sparse (Khusro, Ali, & Ullah, 2016). Consequently Recommender Systems are especially susceptible to bad data quality and therefore could profit from comprehensive data curation and monitoring. This lends itself to take a data-centric approach when building, deploying and maintaining a RS, which is one of the subject matters that the field of MLOps sets out to tackle (Miranda, 2021).

MLOps emerged from the paradigm of DevOps and seeks to apply an automated and standardized approach to the lifecycle of ML applications, similar to what DevOps does for conventional Software. MLOps is attuned to the specific needs and problems of Machine Learning, such that its practices vary from those of DevOps, while still sharing the same goal of rapid and frequent deployment of Software (Makinen, Skogstrom, Laaksonen, & Mikkonen, 2021). The effect of data quality on the ML model presupposes that data quality management is an integral part of every MLOps system, since data quality affects all aspects of the machine learning lifecycle (Renggli et al., 2021). Detrimental data to the ML system’s performance can manifest itself in different ways.

One manifestation is concept drift (CD), which describes a changing outcome *y* to a constant input *x* over time (Lu et al., 2018). Deteriorating RS performance due to CD can directly impact the health of the online platform it is used on, as outputs of RSs are generally reciprocated back to the user experience. For instance, if a movie streaming platform stops recommending appropriate movies to a user because it failed to adapt to the change in taste, the user might stop watching movies on that platform and eventually cancel their subscription. Issues of Concept Drift need to be addressed and mitigated to ensure user-base retention for online services. Additionally, it needs to be incorporated into a MLOps system to benefit from the maintainability, consistency and automation of a unified process.

The product of this work, called an artifact, will be the implementation of a concept drift-aware MLOps pipeline for a RS. CD-awareness meaning, that it possesses the ability to account for potential CD in the data.

This paper serves as a thorough documentation of the design of the artifact, which is based on a comprehensive dissection of scientific literature touching the topics of Recommender Systems, MLOps and Concept Drift. The result will then be qualitatively evaluated and discussed.

This research follows the design science research (DSR) methodology of Alan R. Hevner (Hevner, March, Park, & Ram, 2004).

## Research Methodology

Design science is a research paradigm that emerged as a differentiation to natural science in STEM. Natural science, also referred to as behavioral science, is associated with fields like mathematics, physics, biology and chemistry. Its research methodology follows the objective of uncovering facts and theories about reality. Juxtaposed to the natural science lies the design science. Instead of uncovering rules and theories about the nature of reality, design science sets out to engineer and create artifacts with tools from scientific literature. Design science is predominantly represented in the engineering and computer science fields, where proof of concepts (PoC) and prototypes are the result of a lot of academic works. Both behavioral science and design science have distinguished approaches on how to conduct research.

Design science research contains a set of frameworks and best practices to manage academic work in the design science department. One of the more prominent methodologies is Alan R. Hevner’s “three cycles” of DSR (Hevner et al., 2004). Hevner originally designed his framework to involve the research aspect more closely to the development process of Information Systems (IS) in enterprise environments. It consists of 3 cycles which are closely related to each other and serve to build an artifact. The three cycles are what Hevner argues separates design science from other research paradigms (Hevner, 2007).

The artifact is the eventual product of the academic work using DSR. Since its first publication in 2004, DSR has found application in a wide variety of fields that surpasses conventional engineering and computer science. This means that the term “artifact” has a broad definition and is consequently hard to delimit. Generally, an artifact means anything that emerges from design science research. It could range from a theoretical model that was derived from other academic work, to a physical prototype or a production-ready software system.

The goal of DSR is to create an innovative artifact, which incorporates both theoretical-scientific, as well as the practical-environmental (e.g. business) aspects into its design. The iteration through the three cycles creates a mutual feedback-loop between the artifact and the science and business environment (Hevner et al., 2004). The result is an artifact, which is attuned to the business needs of an enterprise, while also holding scientific value and enriching the academic field with new insights and findings.

In the following, the three cycles will be elaborated in more detail.

1. **The relevance cycle**: The relevance cycle initiates the DSR process. In this cycle all requirements relevant to the artifact and the research are worked out. First, a problem is defined and opportunities and arguments are laid out supporting research to resolve the problem with an artifact. Since DSR has its roots in the enterprise sector, it is vital to map out and contextualize the environment this research takes place in, as it directly influences the design of the artifact. In order to evaluate the artifact, acceptance criteria need to be defined. This way a conclusion can be made whether the artifact succeeded in its goals to improve the environment or not (Hevner & Chatterjee, 2010). In this work the results of the relevance cycle are to be found in the introduction of this paper.
2. **The rigor cycle**: Following the relevance cycle, comes the rigor cycle. While the relevance cycle establishes the requirements for the project, the rigor cycle introduces the methods, drawn from scientific literature, used to create the artifact (Hevner & Chatterjee, 2010). This so called *knowledge base* consists of engineering methods and scientific theories and sets the foundation from which the artifact will be designed and built from. A thorough rigor cycle ensures that the artifact is grounded on state-of-the-art literature from the academic field. This establishes the connection to other scientific contributions and thus sets it apart from routine designs and routine design processes (Hevner & Chatterjee, 2010). The rigor cycle gives the artifact the scientific weight it requires to be acknowledged as an academic contribution, consequently it is vital that the design of the artifact draws sufficiently from the knowledge base of the rigor cycle. The knowledge base is referenced in the State of Research of this paper
3. **The design cycle**: “*The internal design cycle is the heart of any design science research project.*” (Hevner & Chatterjee, 2010) The design cycle is the culmination of the relevance and the rigor cycle. The information acquired from the two prior cycles will now be deployed to design and implement the artifact. Hevner points out that it is not possible to retain both maximum relevance and rigor simultaneously, thus a balance between both need to be struck (Hevner & Chatterjee, 2010). In the context of the design cycle the rigor represents the actual construction of the artifact, meaning the implementation of the knowledge base gained from the rigor cycle. In opposition to the rigor stands the relevance. The relevance represents all the requirements and evaluation criteria that were specified in the relevance cycle. The discrepancy between the relevance and the rigor gets resolved by the artifact, which is the bridge between both and thus constitutes the business and scientific contribution. The design cycle is documented in the Artifact design chapter of this paper.

DSR is a non-linear process. With progression of the research project, the relevance, rigor, and design cycle can change as new insights are garnered. Through iterative cycles Hevner’s DSR methodology accounts for the often unpredictable nature of the artifact creation process. Should either parts of the relevance, rigor or design fall out of line with the current state of the project, it needs to be updated by reiteration (Hevner et al., 2004).

Once the research is conducted and the artifact is created it itself becomes part of the knowledge base, whose insights can now be used for other research projects.

# Current Environment & State of Research

## Environment

Machine Learning serves great value to businesses. In 2017, Netflix for instance claimed an estimated saving of $1 billion through their use of RSs (Columbus, 2017). Since 2017, ML algorithms became more sophisticated and hardware more powerful to make Artificial Intelligence (AI[[1]](#footnote-2)) operations more efficient, effective and in return more profitable. Gartner calculated the revenue of the AI software market to be over $51 billion in the year 2021 with a prediction to surpass $60 billion by the end of 2022 (Rimol, 2021). Open-source software, ML cloud services and an active community make AI more accessible to a wide variety of businesses. These developments make a growing number of institutions consider optimizing, augmenting, or even reinventing their current operations with ML. McKinsey’s “The state of AI in 2021” reported that 56% of their surveyed businesses have adopted ML and AI in at least one of their business functions. An increase of 6% compared to the preceding year (Chui, Hall, Singla, & Sukharevsky, 2021). It’s apparent that ML and AI receive increasing interest in the enterprise sector.

According to a survey conducted by Refinitiv, out of 447 international institutions that use ML, only 46% have deployed AI in multiple areas and are core to its business, whereas 44% deployed ML in pockets, while the remaining 10% were still prototyping and investing in its infrastructure (Baker, 2019). This indicates that a majority of enterprises, while considering or using ML for their businesses, struggle to embed it into their existing infrastructure. This observation is also supported by Algorithmia’s 2020 report on enterprise machine learning, which uncovers that 55% of companies “*actively developing machine learning lifecycles or […] beginning their machine learning journey*” (Algorithmia) have yet to deploy a machine learning model. This report highlights that a lot of the main difficulties of ML lie in its operational aspect, such as reproducibility, versioning of models and scaling of the ML system. This leads to “*unreasonably long roads to deployment*” (Algorithmia) and impedes evolving the ML system to higher levels of maturity (Algorithmia). Another Refinitiv study also identified lacking data quality to remain the biggest challenge for ML and data science (Refinitiv, 2020).

The aforementioned McKinsey whitepaper made the observation that companies most successful with AI were employing advanced operation procedures such as MLOps, as well as putting greater effort into mitigating and reacting to “*AI-related-risks*”, such as concept drift. This applies to the operation of RSs as well. Operationalizing the ML lifecycle and ensuring high model quality through data quality assurance are essential to a RS in a productive environment. A data-centric approach to RSs combined with the application of MLOps practices should prevent or alleviate unsatisfactory RS performance while ensuring maintainability and reproducibility to an mostly automated ML process. Especially the phenomenon of concept drift needs to be accounted for as it is an inevitable occurrence for a majority of real world data. A key challenge for RSs is its often unpredictable nature and sudden appearance, which can be detrimental to business operation. Shift in the data can be subliminal as it is not tangible in most cases and therefore can go unnoticed by Data Scientists and ML engineers. Consequently the effects of CD can first become apparent through degradation of model performance. Scalable and automated ways to account for change in ML data still are in their infancy and therefore require further research and development to engineer solutions. In the long run a state-of-the-art ML infrastructure will make RS operations more profitable for businesses through labor reduction and performance improvements.

This introduction poses the baseline for the relevance cycle of Hevner’s design science research method. The collected insights from the whitepapers serve to map out the environment and establish the motivation for this research. The acceptance criteria for the later artifact will be in part derived from the findings and motivation in this chapter, which will be elaborated in the following chapter.

Having established the environment and motivation, the following research question is formulated for this work: “*What can a MLOps pipeline for a recommender system, that takes concept drift into account, look like?*”

## Recommender Systems

### Overview

Online platforms like Amazon, Netflix and Facebook sit upon massive amounts of data garnered from their user-base through interactions on their websites and applications. The internet enabled firms to leverage information about customers in a large-scale, centralized and automated fashion. In the past, in order to collect insights about customers, polls had to be conducted on sight of a physical store. The results were costly polling ventures of small sample sizes over a limited time period. Nowadays large quantities of high quality data is streamed uninterruptedly into data lakes where they are stored. This unprecedented amount of data enables data science and machine learning to thrive. While many large corporations live in abundance of user data, it still remains a main challenge for them to gain useful insight from the data (Cai & Zhu, 2015).

Recommender Systems use the given data to uncover preferences of end-users to then tailor the service to their likings. The most known examples are shopping basket recommendations on e-commerce sites like Amazon or movie and video recommendations of streaming platforms like Netflix and YouTube. As a generalization, the term *item* is used to denote any object in a set of items that can be recommended to a user, such as a product or a movie. The recipient of an item is referred to as the *user* or *query*. The user-item relationship makes up the foundation of any recommender system, as RSs work under the assumption that there exists dependencies between user- and item-centric activities.

The main task of a RS is to correctly identify these dependencies and use them to match appropriate items and queries (Aggarwal, 2016). In order to reduce information overload on websites with video and product catalogues, RSs are applied to identify items that will interest the user (Alyari & Jafari Navimipour, 2018). Aggarwal (Aggarwal, 2016) lists four aspects that define a good recommendation:

1. *Relevance*: The most fundamental objective of a RS is to recommend items that are relevant and interesting to the individual user.
2. *Novelty*: The RS should introduce items to the users that they weren’t aware of before. It has been shown that recommender systems can negatively impact the sales diversity of e-commerce, when only popular items are recommended to users (Fleder & Hosanagar, 2007).
3. *Serendipity*: Similar to the point above, RSs should have the ability to surprise users with somewhat unexpected but relevant items. While novelty focuses on the quantitative aspect of recommending items that aren’t on the radar of the user, serendipity considers the qualitative-psychological impact of a surprise in the recommendation.
4. *Increasing recommendation diversity*: This last point envisions that RSs should ensure a diverse set of items in a recommendation feed. This should be done to not fatigue the user with items of similar categories. An example would be the recommendation of movies from a variety of different genres.

Throughout the years, different approaches have been developed to optimize recommendations and overcome obstacles in this field. These recommendation systems are broadly categorized in content-based RS (CB), collaborative filtering (CF), knowledge-based RS and hybrid RS (Aggarwal, 2016). In the following, CF and CB are introduced to provide an overview of the RS landscape, as both belong to the most popular research topics for RSs, excluding hybrid systems, which are a combination of multiple stand-alone RS methods. Collaborative filtering will be elaborated in more detail, as it is a key concept for the RS implemented in the artifact. Figure 1 shows the structure and an overview of the recommenders based on Aggarwal (Aggarwal, 2016). Highlighted in red is the type of recommender system that will be used in this this research project, which is a deep and cross network. Before elaborating CF and CB, there will be a chapter dedicated to RS data.



Figure 1: structure and overview of RSs derived from Aggarwal (2016)

### Data

The data used by RSs often differ from those of other Machine Learning tasks, as it is often sparse. Sparsity refers to very few user-item interactions relative to the size of the whole dataset. Assuming one would construct a matrix of an e-commerce website, wherein represents the entirety of its user-base and represents all the items in the product catalogue. Every time a customer buys or inspects a product, an interaction between the user and the product is denoted in . Since customers only interact with a fraction of the offered products in an e-commerce web shop, there are very few entries in the 2-dimensional matrix, hence it is sparse. This is a fundamental problem that RSs face and try to overcome.

The value of an interaction between item and user can be represented in different ways depending on the context of the recommendation. Is an interaction defined as a review with a rating between 1 and 5 stars, it will be marked with an integer value in a range of 1 to 5. These rating types allow the user to quantifiably express their like or dislike of an item. These are called interval-based ratings (Aggarwal, 2016). Another popular rating type is the unary rating system. These are used on social media platforms like Instagram and Twitter. These ratings are encoded in a binary fashion, wherein an interaction (e.g. like) will be conventionally denoted with the value 1 and abstinence will be denoted by the value 0 (Aggarwal, 2016).

Both the like-feature as well as the 5-star rating system are called explicit ratings, as they are actively and consciously given by a user. While explicit ratings generally provide reliable information about an users opinion, it remains a challenge to entice users to rate. This in return makes the data more susceptible to shilling attacks, where item ratings get skewed by coordinated bombardments of overly positive or negative ratings by external sources (Khusro et al., 2016).

In opposition to explicit ratings, lie the implicit ratings. These rating systems derive a rating from the user through their behavior. An examples could be the watch time of a movie. This example works under the assumption that people that don’t close a movie, are engaged with it. The drawback of an implicit rating is its ambiguity, as online behavior needs to be interpreted first and then translated into a rating. User behavior can be interpreted wrongly and skew the data. Continuing the aforementioned movie example, a user who slept during the movie out of boredom, would be misinterpreted as a user who was engaged throughout the whole movie, since they didn’t close it. An implicit rating system has the advantage that it is not reliant on the user to give feedback, which generally results in more ratings. This is a significant factor considering sparsity is a main challenge for RSs (Su & Khoshgoftaar, 2009).

In a recommender dataset the value of a user-item interaction is referred to as a *sparse feature*. They can be augmented with so-called *dense features*. Whereas sparse features only occur through user-item interaction (e.g. a rating), dense features are specific attributes that describe the item or user at hand, comparable to features in other ML tasks.

Data understanding plays a vital part in the success of recommender systems. There are many variables ranging from its rating system over its features and sparsity that determine, which recommender system is the most suitable for the given task (Aggarwal, 2016, p. 128).

### Content-based recommender systems

Content-based methods make recommendations based on a user's past interactions with items. These users are referred to as objects and contain information from their history of item-interactions (e.g. movie watches, ratings, etc.) (Lü et al., 2012). These items have descriptive attributes, named content, which form the basis of which these recommender systems derive their recommendation decisions from (Aggarwal, 2016). These contents are used to picture the preferences of a user. In a movie recommendation setting such a content could be the genre of a movie. Depending on what contents are found in an object, certain assumptions can be made about a user's likings. Other items can then be assigned to the user based on a similarity measure between the object and other item objects. Alternatively new items can be suggested by weighing the content by their frequency in an object. Popular methods for content-based filtering are information retrieval algorithms like TF-IDF (term frequency-inverse document frequency) (Lü et al., 2012). One characteristic of CB is that both the item and the user are represented in the same feature space, in order to measure similarities between an user object and all item objects.

Item recommendations can be done independently from other users, as it is only of interest how the individual user is positioned to the items in the vector space, which is one of the advantages it has over CF. This means that CB can be applied in environments with relatively small user-bases. Since CB relies on a user history to make recommendations, it has challenges finding the right recommendations for new users, since there isn’t enough content to map out the user (Aggarwal, 2016). Conversely, making recommendations for new items is relatively easy, since the item objects only need to be provided with appropriate content. The success of content-based filtering is highly determined by the quality of attributes in these objects. These features need to be cautiously engineered. Since the predictions are calculated on the basis of the user’s and item’s attributes, it is not possible for these algorithms to uncover hidden variables in the data, like for instance some CF algorithms could. In general, the performance of CBs are very reliant on how the data has been engineered by data and ML engineers.

### Collaborative filtering

CF has been the baseline for RSs for years. In September 2009, the BellKor team was awarded with the grand Netflix prize, which it has won by applying a CF model on the competition dataset (Koren, 2009a). As the name implies, CF uses the "collaborative" power of the user-item interactions in the dataset (Aggarwal, 2016). Instead of solely relying on item and user descriptions, CF uses the composition of the dataset to determine the output of a recommendation. Colloquially, one would say that the recommendation is driven by a "joint effort" of every user-item interaction. Unlike CB, recommendations can be made without relying on hand-engineered features (Google LLC, 2021).

**Memory-based filtering.** CF can be divided into *memory-based* methods and *model-based* methods. The prior uses neighborhood based algorithms to make recommendations. These methods belong to the earliest CF practices, where ratings are predicted based on similar users or items (Aggarwal, 2016). "Memory-based" relates to the fact, that the dataset is loaded into memory, in order to make calculations. The dataset is then actively used to compute the right recommendation through nearest neighbor methods. For that reason the data in memory-based CF can be described as part of the model, as recommendations are determined by the arrangement of datapoints within the vector space.

Memory-based methods can either take a *user-based* or *item-based* approach. In user-based CF, rating predictions are provided by like-minded users. Similar user groups are identified using nearest neighbor techniques, then the unobserved ratings are filled in by the values of the peer group. This could be done by calculating a weighted average of all ratings from that group (Aggarwal, 2016). In item-based CF a set of items a user has rated most similar to an unrated item are retrieved with a neighborhood algorithm. In item-based CF a set of items, already rated by a user, are used to determine the value of an unspecified item. Via neighborhood algorithms items are determined that resemble the unrated item the closest. Like with user-based CF, the rating is derived from the neighboring entities, only that instead of using the values from similar users, the values from similar items are chosen (Aggarwal, 2016).

The advantage of a memory-based method is its interpretability, as ratings are directly taken from other users or items. Since CF methods (memory-based and model-based) obtain their values through other users and items, it is easier to achieve serendipitous recommendations, as similar users might have interacted with items the target user isn’t familiar with.

Memory-based algorithms don’t work well with sparse data, as there might not be sufficient items or users similar to each other (Aggarwal, 2016).

**Model-based filtering.** Unlike memory-based approaches, model-based methods have a training phase, whereby a model is learned. The goal is to create a function (the model) that approximates the results of an item-user interaction, which is conventionally achieved through optimization of a loss function with gradient descent or alternating least square algorithms (Koren, Bell, & Volinsky, 2009). Two of the most prominent model-based CF techniques are matrix factorization (MF) and deep learning (DL).

The former has been the standard in the field of RS and was also utilized by the winning team of the Netflix prize in 2009 (Koren, 2009a). MF is foremost a dimensionality reduction technique with which the user-item matrix gets decomposed into *low-rank[[2]](#footnote-3) latent factors*. These latent factors are the components that make up the *latent factor model* (LFM), which is a low dimensional representation of the initial matrix. LFM, broadly classified as a *factorization machine* (FM), works under the assumption that there are underlying latent variables in the data. Latent variables are variables that are not directly observable but can be inferred through mathematical computation, like for example singular value decomposition (SVD) (Loehlin & Beaujean, 2017). Koren et al. (Koren, Yehuda and Rendle, Steffen and Bell, Robert, 2022) state that the observed rating values are due to effects associated with either user or items, independently of their joint interaction. Therefore, there are large item and user biases embedded in the data. An example would be the tendency for some users to give either higher or lower ratings on average, or conversely products that receive systematically higher or lower ratings. These “hidden” factors can be retrieved during model training and incorporated into a latent factor. In the training process the latent factors are distilled that best depict these propensities in the data (Vellido, Lisboa, & Meehan, 2000). In LFM, the rating of an unobserved user-item interaction is constructed by applying the dot product of factor of user and factor of item (Aggarwal, 2016). While the baseline LFMs only use the sparse matrix of user-item interactions, more sophisticated variants, like SVD++, can also incorporate dense features and implicit ratings into its model, which allows it to factorize more information and further improve prediction quality (Koren et al., 2009).

The main advantage of MF lies in its ability to algorithmically detect latent vectors in the dataset. Overarching correlations are extracted in the dataset and used to make predictions. Unlike CB methods, the models don’t rely on hand-engineered features to make recommendations, which evades the human-error component of feature engineering and speeds up the development process. Dot products are computationally light, which makes LFM models scalable and applicable on large datasets (Blondel, Ishihata, Fujino, & Ueda, 2016). The performance advantage of FMs rapidly diminish when modeling higher-order feature combinations however, which is a tradeoff to allow for more complex embeddings in the model (Blondel, Fujino, Ueda, & Ishihata, 2016). FMs being bound by their shallow structure hinders their overall representative power (Wang, Fu, Fu, & Wang, 2017). In practice, simple FMs still are very capable at making accurate predictions (Dacrema, Boglio, Cremonesi, & Jannach, 2021).

In order to circumvent the above-mentioned limitations of FMs, *deep neural networks* (DNN) have gained increasing interest in RS research over the past years (Dacrema et al., 2021). *Neural networks* (NN) are very effective at retrieving complex embeddings from their input, which makes them state of the art in the field of image recognition and natural language processing (NLP).

Neural networks emulate a simplified model of how human or animal neurons actually work in a brain. Like the brain in biology, *artificial neural networks[[3]](#footnote-4)* (ANN) consist of multiple individual neurons that are interconnected, whose joint response to an external input creates an output. Unlike neurons in our brains, which are structured in a complex and organic way, neural networks are arranged in layers, which contain a set of artificial neurons (Charniak, 2019).

The baseline neuron in NNs can be thought of as an individual mathematical function, that consists of a weight component and a bias component, which are wrapped inside an activation function. This neural function receives an input, which can either be the external input data, or the output of neurons from the previous layer, depending on whether the neuron sits in the first layer of the network or not. In the neuron, the input value gets multiplied by the weight then the bias offsets the product by a certain value. The activation function then refactors this weighted sum, which is the output value of the neuron. Activation functions are non-linear functions that determine how “activated” a neuron is, i.e. how large the output value should be. A popular activation function is *rectified linear unit* (ReLU), which represses any activations (i.e. neuron output value of 0) up to a certain threshold (Nair & Hinton, 2010). In the output layer (the last layer of the network) activation functions like *Softmax*[[4]](#footnote-5) are preferred, as they normalize the neural computation to a probability distribution. In the case of classification, an *Argmax* function can then be applied on the output layer to retrieve the neuron with the highest Softmax output, which represent the class the NN predicts with the highest confidence (GOODFELLOW, BENGIO, & COURVILLE, 2016). The non-linearity of these activation functions in combination with the multiplicity of artificial neurons, allow NNs to retrieve highly non-linear patterns from the data, which is the reason they are also referred to as universal function approximators (Charniak, 2019; Gurney, 2014; Hanin, 2019; Heaton, 2012).

NNs are learned through *deep learning*. Intuitively speaking, the objective of the learning process is to tweak each parameter (i.e. weight and bias) in a NN in such a way that a desired output corresponding to a certain input is generated. The training is done through an iterative process called *gradient descent*. Before the training process begins, each weight in the network gets assigned an initial random value. This offsets the NN to learn distinct embeddings, which would not be possible if each weight received the same value. During the training process *test data* is fed through the network. With a loss function the desired target output is compared to the actual output of the unoptimized model . The goal with each training iteration is to decrease the distance between and . To achieve this, the gradient vector of the loss function is calculated with respect to the entire NN. This computation of the gradient in a NN is what’s called *backpropagation*. We established earlier, that each neuron (excluding the first layer) is a function of the outputs from the neurons in the previous layer. Backpropagation recursively calculates the gradient inside each neuron in the NN, starting with the output layer and propagating through the entire network to the input layer. Having calculated the gradient, the opposite sign of it is taken and multiplied by the *step size*, a coefficient which defines how aggressively the parameter values are tweaked. This process is repeated until the NN function converges to a local minimum. A small step size increases the learning time, since it takes longer to reach a minimum. Conversely, a large step-size runs the risk of exceeding the local minimum, which results in inferior performance (Gurney, 2014; Heaton, 2012; LeCun et al., 1989).

For every model training, it is crucial to separate the training data from the test data to prevent an overfit of the model. This is especially true for NNs, as they have the tendency to perfectly adapt to the data it gets trained with, which greatly impacts the generalizability of the model and impairs prediction performance on unseen data (May, Maier, & Dandy, 2010). In general, the learning of NNs is a non-trivial task, as they are very sensible to the embedding of the input data, the initialization of the weights, the choice of activation functions and hyperparameter tuning, like step size and *dropout regularization[[5]](#footnote-6)*. A lot of variables that determine the efficacy of a model make it harder to extract the full potential out of a NN. In addition, NNs can’t easily be interpreted because of their complexity, which is why they are often treated as black boxes, especially with very deep models. (Dacrema et al., 2021) discovered that simple MF methods show similar performance to other state of the art DL models, when selecting the right parameters. Compared to a linear LFM, NNs also have a lower processing speed, which is detrimental to large recommendation datasets, since passing a neural network is much more computationally expensive than a simple dot product calculation (Rendle, Krichene, Zhang, & Anderson, 2020).

This notion changes when comparing NNs to polynomial FM, which are much more computationally complex compared to their low-degree counterparts (Blondel, Fujino, et al., 2016). As universal function approximators, DNNs have the potential to extract much more complex patterns and relationships out of the dataset, especially as both dense and sparse features can be trivially incorporated into its embedding. With ongoing research, DNN approaches for RSs are further being refined and its deficiencies, like performance, investigated and improved. Currently, high profile enterprises like Google use NNs in large scale environments for recommendation tasks (Covington, Adams, & Sargin, 2016).

### Deep & Cross Networks

When using data with multiple features to make recommendations, *feature crosses* become an important concept. The notion of feature crosses is that the combination of multiple features into one feature can yield an expressive predictor variable, that would otherwise not be extractable using only single features. Individual features that have very low correlation with the label can therefore reveal themselves to be highly correlative in conjunction with other features. Uncovering these *cross features* allows to extract additional information out of the data, which in return can significantly improve model performance. The amount of features that are embedded in a cross feature is denoted by the order: A cross feature being the product of two features is called a cross feature of 2nd order. Cross features of 3rd order and up are referred to as higher-order features (Wang et al., 2017; Wang et al., 2021).

Suppose a RS that predicts the user rating of a movie. Besides the sparse features (in this case the user ID and movie ID), the data also contains the genre and length of the movie as dense features. Assuming that length as a stand-alone feature is not a strong predictor for the rating of a movie, it could be combined with the genre feature into a separate cross feature of 2nd order. Now, a feature that has both variables encoded in it can be used to better predict the ratings of a movie, the hypothetical interpretation being, that different genres set certain expectations of what the movie length should be. For instance, while it might be acceptable or even desired for fantasy movies to be 3 hours long, a 3 hour runtime for a comedy could generally be viewed unfavorably.

Feature crossings are often done as part the data preparation phase of a ML lifecycle (Studer et al., 2021). This process can be very daunting and time consuming, as it entails trial and error of which cross features are best suited for accurate predictions. Also, feature crosses aren’t always obvious and thus might go unnoticed. Instead of manually feature engineering cross interactions, it would be lucrative to directly incorporate feature crosses into the ML model and let the learning algorithm determine the weights of cross features. DNNs are capable of learning arbitrarily high order functions, provided the NN is deep enough. They however only learn feature interactions implicitly, which means that they do not reliably pick up on higher feature crosses without drastically increasing the network size (Wang et al., 2017). This brings the general efficiency of DNNs into question, especially in light of large scale recommendations that need to compute high amounts of queries per second (QPS) (Rendle et al., 2020).

In order to reduce the overall model size of NNs and invoke explicit cross feature learning, Google proposed their deep and cross network (DCN) in 2017 (Wang et al., 2017). In addition to a classic feed-forward multilayer perceptron (MLP) NN, as covered previously, a DCN is augmented by an additional cross network, consisting of cross layers. The cross network enables learning on bounded-degree cross features, whose order is explicitly defined by the layer depth of the cross network. This way, ML engineers can now directly incorporate the order of feature interactions they wish to embed, by changing the cross network structure. At the start of the network (i.e. the input layer) the feature vector doesn't have any explicit cross feature interactions yet and is therefore of order 1. With each cross layer the feature vector passes, the maximum polynomial degree of cross interactions increases by 1. This is because for each cross layer the output of the previous layer gets crossed again with the input vector to construct a new cross term . A visualization of a cross layer in Google’s DCN V2 architecture is seen in Figure 2, whereby the learned parameters and respectively denote the weight matrix and the bias of the cross layer.



Figure 2: Visualization of a cross layer (Wang et al., 2021)

A DCN is always made up of two components: A cross network and a conventional MLP neural network. While the cross network is used to learn explicit bounded-degree feature interactions, the DNN component is used to learn implicit feature interactions in the data[[6]](#footnote-7). With both components the model is therefore able to effectively leverage explicit and implicit interactions in the data, which improves prediction performance. Depending on what architecture is chosen for the DCN, the DNN can either follow the cross network, or run in parallel, as seen in Figure 3.



Figure 3: Possible DCN architectures (Wang et al., 2021)

With the introduction of cross layers, the model becomes more effective and predictable at embedding feature interactions into its network. While feature selection and feature embedding still remains a non-trivial task, it is only of concern at the level of individual features, since cross feature extraction is now delegated to the model training, instead of being a separate, manual task (Shan et al., 2016). This saves labor time and removes the human error element. Being able to extract both implicit and explicit feature interactions in the data through the combination of a cross network and DNN, makes DCN overall more competitive at predicting ratings. Besides improving the prediction performance of recommendations, the model also becomes smaller and in return faster, which makes it more viable for large-scale RS operations.

### State of the Art Technology

**TensorFlow Recommenders** (TFRS) is a python library built on the Keras API. TFRS unifies and simplifies the building of DL RS models under TensorFlow, by providing functionality and components at each step of the RS development cycle, such as methods for retrieving and ranking recommendations, activations functions, and ready-to-use RS datasets (TensorFlow, n.d.). TFRS also provides built-in RS specific network layers for model building, such as the aforementioned cross layers. In combination with the accessibility of the Keras API, it opens the possibility to intuitively realize state-of-the-art DCNs (TensorFlow, 2022a, 2022b).

**TorchRec**. In January 2022, Meta released an open source library for building recommendation systems under PyTorch, called TorchRec. This library was utilized to build the state-of-the-art RSs employed in Facebook’s Newsfeed and Instagram’s Reels (Meta, 2022). TorchRec provides optimized RS kernels for efficient matrix multiplication in the model. CUDA enabled GPU acceleration improves model training time, provided that the supported hardware is used (Zhang et al., 2022). TorchRec can automatically split up large RS models into shards for distributed prediction serving. *Sharding* is a common pain point for large scale RS operations. Modern RSs can’t be hosted on a single server, as the model size exceeds most memory specs of server racks. In order to retain the performance of large RS models, the model need to be broken up into shards. These shards are then distributed to multiple compute nodes, where they can jointly compute and serve predictions (Lui et al., 2020). Meta seeks to address this topic by providing automated sharding plans for models (Greenberg, Taylor, Ivchenko, Liu, & Sudarshan, 2022).

## MLOps

### Problem

Machine Learning has its origins in academic research (Minsky, 1961; Rosenblatt, 1961). Over the last twenty years it also found its way into the enterprise sector, where it now stands as the backbone of the most successful companies and most profitable business ventures. With the transition from academia to industry, ML engineers and data scientists are confronted with unique sets of challenges that are usually absent in a sterile research environment. While research often is dedicated to investigate one aspect of ML (i.e. the algorithm), enterprise devotes itself to the development, operation, and maintenance of entire ML systems. Software in research is written to investigate a certain topic and to accept or decline a very specific hypothesis. It has a very narrow use-case and therefore lacks generalization, which makes it often not suitable for a productive setting. In addition, development of the software usually stops with the conclusion of the research project (Garousi, Petersen, & Ozkan, 2016). On the other hand, software employed in business environments are built for longevity, with their goal being a profit driver for the company. This is composed of low operational costs and high revenue-yield (Heemstra, 1992; Myers, Kappelman, & Prybutok, 1997). As a result of these different goals, both research and industry employ different development paradigms, with focus on different aspects respectively. These differences are observed in the RS landscape of academia and industry, for example. While in academia state-of-the-art RS algorithms are assessed with their performance on static benchmark datasets (e.g. MovieLens dataset), industry also considers the integration of the algorithm into the system architecture. Instead of merely relying on benchmark performance, industry systems need to take into account the scalability of an algorithm and embed their RS into their application in a meaningful manner (Covington et al., 2016).

One of the first papers to illustrate challenges of productive ML systems is Sculley et al. (Sculley et al., 2015), which uses the framework of *technical debt*, found in software engineering, to describe problems in a productive software environment. To be more precise, the term technical debt refers to ongoing maintenance cost due to shortcuts taken during initial development of the software. The debt analogy is used to visualize upfront cost-savings due to cheaper and faster development, that however need to be paid back with interest down the line with excessive software service and maintenance (Cunningham, 1993). In classical software engineering technical debt can manifest itself in different ways, such as code refactor, deleting outdated modules, reducing dependencies and improving documentation (Fowler, 2019). The same phenomenon of technical debt is also observed in ML systems. Sculley et al. (Sculley et al., 2015) points out that a lot of the technical debt incurred in ML, is due to the fact that it directly interacts with the external world. Since ML holds a tight bond with the ever-changing data it consumes in real-world applications, maintenance turns out to be particularly difficult and expensive. So, in addition to the established technical debt of software engineering, ML also receives its own set of technical debt that needs to be considered as well (Sculley et al., 2015).

Enterprise ML are often very complex software systems that comprise various interconnected components, consequently the ML algorithm itself only makes up a small portion of the entire system. In addition to the complex array of variables in the ML model alone, all parameters from the other software components have an effect on the ML system as well, which results in a web of entanglements. It becomes hard to keep track of all dependencies within the system, which, in return, complicates the maintenance of the software, which is referred to as *configuration debt*. The changing anything changes everything (CACE) principle goes into effect and any adjustments to the system can severely compromise the model performance (Sculley et al., 2015). Related to the configuration debt, is the *reproducibility debt* in ML systems. Reproducibility is an inherently difficult undertaking in productive ML systems, as these algorithms often work with randomized parameters, non-determinism in parallel learning and continuously changing data.

Another form of technical debt occurs when ML researchers and data scientists don't adjust their habits when moving from the academic field to industry. As mentioned in the beginning, academia and industry apply different software development paradigms. Not adapting their ways of working to the needs of production software may greatly impact the maintainability of a ML system. Common culprits are so-called *glue code* and *pipeline jungles*. Glue code is understood as software code, that doesn’t directly contribute to the functionality of the software’s or system’s requirements, but acts as a compatibility layer between two or more components. Glue code is often written in presence of stand-alone ML packages or commercial-of-the-shelf software products, in order to integrate them into the ML infrastructure. Glue code generally distracts from the actual purpose of the application. It makes code less readable and degrades performance. It also makes the software less flexible to changes and updates, as glue code is tailored towards specific interactions between specific components (Sculley et al., 2015). The pipeline jungle is a special type of glue code, which evolves out of a base pipeline, through incremental expansion. Often found in the data pre-processing phase, these pipelines are continuously built upon to accommodate for new data sources and data transformation steps. In the end, the pipeline becomes error prone, hard to test, and convoluted (Sculley et al., 2015). Both glue code, and pipeline jungles are symptoms of code that isn’t designed to be reused and built upon, which is emblematic of software in research projects, as they often serve as PoCs.

Lastly, a large portion of the technical debt in ML systems is due to their nature of working with dynamic data from the external world. Not dealing with changes that occur in the dataset can lead to an amalgamation of technical debt, to the point where it might even render the whole model obsolete (Lu et al., 2018; Sculley et al., 2015). The implications of changing data will be dissected in greater detail in chapter Concept Drift.

### What is MLOps

Paying down ML related technical debt demands a re-examination of status-quo practices in a business environment. Industry has different sets of priorities compared to academia, which need to be respected. Instead of determining the viability of a model based on specific performance metrics alone, the effects on operability of the whole system need to be taken into consideration. Small performance gains at the cost of maintainability usually don't fare well in an enterprise environment. At the same time, a infrastructure needs to be set in place, that is capable of preventing and mitigating technical debt. This entails sustainable design patterns in ML code, such as modularity and abstraction, rigorous testing, as well as practices that promote team work (Sculley et al., 2015). The goal for the last few years, was to apply practices of agile software development to ML. As assessed in the prior chapter, ML development contains a few peculiarities, that sets it apart from conventional software development, which makes it incompatible with common technical methodologies, like DevOps. Out of necessity to employ methodologies that are attuned to the unique characteristics of ML, the term MLOps (was born. Despite gaining attention from high profile ML researchers and companies, MLOps still remains a vague term that is hard to pin down (Kreuzberger, Kühl, & Hirschl, 2022; Tamburri, 2020). This is due to the novelty of this field, with different sources defining MLOps differently.

A Google whitepaper about this topic defines MLOps as “*a methodology for ML engineering that unifies ML system development (the ML element) with ML system operations (the Ops element)*” (Salama, Kazmierczak, & Schut, 2021, p. 5). Similar to DevOps, MLOps seeks to employ practices and tools to shorten development cycles and enable rapid iterations of ML software (Symeonidis, Nerantzis, Kazakis, & Papakostas, 2022). Streamlined operational and governance processes are used to increase performance, reliability and security of ML systems. To sustain this agile development, focus is put on collaboration methods and tools. One overarching theme of MLOps is the unification of research, which is inherent to ML, with the efficiency of modern software development to make ML more profitable and scalable in real world applications (Salama et al., 2021; Sculley et al., 2015).

Tamburri (Tamburri, 2020) defines MLOps from a technological perspective as a set of middleware and software components, that ought to be orchestrated in the cloud to realize at minimum 5 specific functions:

* Data ingestion & transport
* Data transformation
* Continuous ML (re-)training
* Continuous ML (re-)deployment
* Output production & presentation to the end-user

These points are derived from popular cloud MLOps platforms, like Kubeflow, which enable scalable ML pipeline orchestration through containerization (Kubeflow, n.d.). These platforms provide a large portion of the infrastructure to realize state-of-the-art productive ML systems, that can be broken down into following areas of responsibility:

**Automation**. Due to changing data that ML systems consume, they need to be constantly updated and deployed. Employing a CI/CD (continuous integration/continuous delivery) approach for ML, allows development teams to quickly release updated iterations of models and other components. In CI/CD, new models can be automatically compiled into a build and be deployed instantaneously in the needed environment.

CI/CD is often realized in conjunction with automated pipelines, which are a vital part of successful ML operations. A typical ML lifecycle usually consists of repeating steps. Taking the CRISP-ML process as an example, every development cycle needs to run through a data preparation phase (Studer et al., 2021). Since these steps remains consistent throughout different lifecycles, they ought to be automated. An automated process is usually represented by a DAG (directed acyclic graph), which models a process with nodes and edges. The DAG is read by a workflow management tools, such as Apache Airflow, which executes specific tasks in a defined order (Kreuzberger et al., 2022). Pipelines remove manual labor from the ML process, which speeds up the development of new models, reduces the human error element and makes the process more consistent.

In state-of-the-art MLOps the conventional CI/CD practice is expanded by *continuous training* (CT). CT aims to automate the entire model training process. The ML system should be able to detect deteriorating model performance and initiate a model retraining pipeline (Denis Baylor et al., 2019). One form of achieving CT is deploying entire ML pipelines instead of a trained model. This way, instead of deploying a single ML model at the time, an entire pipeline can be deployed to recurrently create new and optimized models (Denis Baylor et al., 2019).

**Reproducibility**. The most straight-forward implementation of reproducibility is through versioning of the source code with a *version control system* (VCS), like Git. As an industry standard, VCSs allow to trace and revert changes in the code. This way previous states of the software code can be restored and rebuilt (Serban, van der Blom, Hoos, & Visser, 2020).

For ML applications, code versioning alone is not sufficient however. Since model training is not only dependent on the model configuration, but also the training data, the dataset needs to be versioned and tracked as well, in order to replicate a previous model (Ruf, Madan, Reich, & Ould-Abdeslam, 2021). This can be achieved through time travel functionality of modern data lakehouse architectures, such as Databricks delta lake (Yavuz & Chockalingam, 2019). Alternatively there are dedicated data management tools, like Rok from Arrikto, integrated into Kubeflow (Arrikto, n.d.).

As alluded to earlier, ML utilizes a wide array of hyperparameters and randomization-seeds for learning. These need to be tracked as well, in order to ensure reproducibility of the model. These are stored in a meta data store, alongside other information, such as training duration and training date.

**Monitoring**. Continuous monitoring is used in MLOps to assess the health of deployed ML systems. Most importantly, the prediction quality of served models are frequently measured in an automated fashion. Important metrics are given a threshold value, that are tracked live by the monitoring system. Sophisticated monitoring implementations automatically trigger pipelines to take corrective actions, upon noticing insufficient model performance (e.g. a re-training pipeline). The observations made during monitoring can feed back into the consecutive model to make it more robust. Automated monitoring can be supported by human monitoring through dashboard visualizations on applications like Kibana, Grafana, or PowerBI (Kreuzberger et al., 2022).

### Maturity Levels

Maturity levels determine the sophistication of a MLOps implementation in an productive environment. The maturity of a ML System can be defined by its technological infrastructure, employed methods of operation, and automation through pipeline orchestration. Because of the novelty of the field of MLOps, there isn't one universal maturity model agreed upon by the scientific community. Companies that are heavily invested in AI therefore design and develop their own maturity models. The most prolific proposals stem from Microsoft and Google (Symeonidis et al., 2022).

**Microsoft**. Microsoft’s maturity model is divided into 5 levels. Level 0: No MLOps, Level 1: DevOps but no MLOps, Level 2: Automated Training, Level 3: Automated Model Deployment, Level 4: Full MLOps Automated Retraining (Figure 4). In this model the following factors determine the maturity of the ML system: People, model creation, deployment, and application integration (Microsoft, n.d.).

Level 0 represents the baseline maturity, where no MLOps is employed. Systems of this maturity have little to no automation in the ML lifecycle. Builds of the ML application are created and deployed manually. Because of missing model repositories, reproducibility is limited. ML models are often treated as black boxes, that are developed in an enclosed environment by an isolated team. Consequently, the ML system isn’t embedded into the larger infrastructure of the company and it is therefore hard to retrieve feedback post deployment.

Level 1 maturity ML systems incorporate standard DevOps practices into the ML lifecycle. Opposed to level 0 maturity, builds are created and deployed automatically. There exists a baseline testing infrastructure for unit and integration tests of the application. Production models are now tracked through a basic model repository, though experimentation still is done in an ad-hoc fashion. Data engineering receives more attention and is brought together with the data science team. Data is collected continuously at this stage. The ML team is however still separated from the software engineering team. For this reason the ML model still remains a black box.

While level 1 maturity focuses on the automation of software engineering tasks relating to the application, level 2 also puts focus on automating the data science tasks. The data engineers and data scientists work together to create unified pipelines for data preprocessing and model training. Scoring scripts and experimental code are now tracked to ensure reproducibility of models. Models are still manually released to the software engineering team for application integration.

Level 3 maturity foresees unification of ML and application. Software engineers work with data engineers to automate model integration into the application. Opposed to lower maturity levels, models are now automatically released to the software engineering team through CI/CD pipelines. Every step from data collection to deployment is traceable. All ML and application components are tested automatically. Deployed models are evaluated with testing methods such as A/B testing.

Maturity level 4 envisions a mostly automated ML lifecycle. ML development and application development are fused into one process. Software engineers integrate live monitoring of various performance metrics into the application. This way model performance can continuously be checked in the production environment. Based on these live metrics a retraining pipeline can automatically be initiated, known as CT.

**Google**. Google’s maturity model is divided into 3 levels. Level 0: Manual process, Level 1: ML pipeline automation, and Level 3: CI/CD pipeline automation (Figure 4). Like in Microsoft’s model, a higher level means a more sophisticated implementation of MLOps for the ML system (Google LLC, 2020).

Similar to Microsoft’s maturity model, level 0 denotes the baseline maturity, where no MLOps practices are employed. Every step is done manually, usually executed interactively on a notebook. There is no distinction between experimental and productive code, as iterations of the model are done on the same notebook until a servable model is trained. No CI/CD is employed, which means that testing is usually done in an ad-hoc manner within the notebook. Once the model is deployed there isn’t a monitoring infrastructure to observe the model performance in production. Consequently, level 0 ML systems don’t deploy models regularly and run the risk of not noticing degrading prediction quality.

The goal of level 1 ML systems is to perform continuous training. In level 0 a trained model is deployed. In level 1 a complete ML pipeline is deployed, which triggers a model training process. The pipeline then continuously delivers up-to-date models that are then used by the application. In production, the input data is monitored live to detect potential shifts and skews, which then can trigger the pipeline to retrain a model. In order to improve reusability and accelerate the pipeline development process, the source code is broken up into modular components. There is now a strict separation between development and production environment.

In Level 2 ML systems, CI/CD is employed to the full extent. The test and deployment of pipelines are automated and the experimentation process is orchestrated. Intelligent monitoring is used on live data, which is able to discern whether a model retraining or a new experiment cycle should be initiated. Information about models, data and metadata are stored in a model registry, feature store[[7]](#footnote-8) and metadata store. This makes ML pipelines reproducible and transparent.



Figure 4: Levels of Microsoft's (left) and Google's (right) maturity model (Google LLC, 2020; Microsoft, n.d.)

### Concept Drift

“*One of the things that makes ML systems so fascinating is that they often interact directly with the external world. Experience has shown that the external world is rarely stable. This background rate of change creates ongoing maintenance cost.*” (Sculley et al., 2015)

One manifestation of ongoing change in data is called *concept drift* (CD). Specifically, CD describes a changing outcome to a constant input over time (Ng, Crowe, & Moroney, 2021). Real world examples of CD could be changing house prices due to a fluctuating house market, or people changing their taste in movies because of aging or genre trends. These changes in the underlying distribution of ML data are often unforeseeable. Therefore they are hard to detect and often go unnoticed for a long time. CD runs the risk of first being noticed through deterioration of model performance in a productive environment.

(Gama, Žliobaitė, Bifet, Pechenizkiy, & Bouchachia, 2014) identify 4 common types of CD as seen in Figure 5:

**Sudden drift**. The distribution[[8]](#footnote-9) of the data undergoes an abrupt change in a short amount of time that persists indefinitely. This phenomenon is also known as *concept shift*. One example for concept shift would be the drastic drop in demand for commercial flight to a country where war just broke out (International Air Transport Association, 2022).

**Gradual drift**. This type of concept drift has two active concepts in the data. Over time, the probability of the initial distribution decreases and gets gradually displaced by the new distribution. An example could be the genre representation in the billboard top 100 charts. As new genre trends emerge, old ones fade out and get replaced by the new ones. During the transitory period both concepts coexist in the data.

**Incremental drift**. Unlike gradual drift, the incremental drift can’t reliably be ascribed to a discrete amount of distributions. The drift occurs slowly with a lot of variables and concepts at play. Consequently, incremental drift is only noticeable when looked at over a long time period. Cultural and societal changes are often incremental in nature.

**Reoccurring concepts**. Previously seen distributions might reappear after some time, as an irregular occurrence. While seasonal changes are reoccurring, they wouldn’t necessarily be classified as concept drift, as they are predictable to an extent. An example for reoccurring concepts are rare weather phenomena like hurricanes (Knotek & Pereira, 2011).

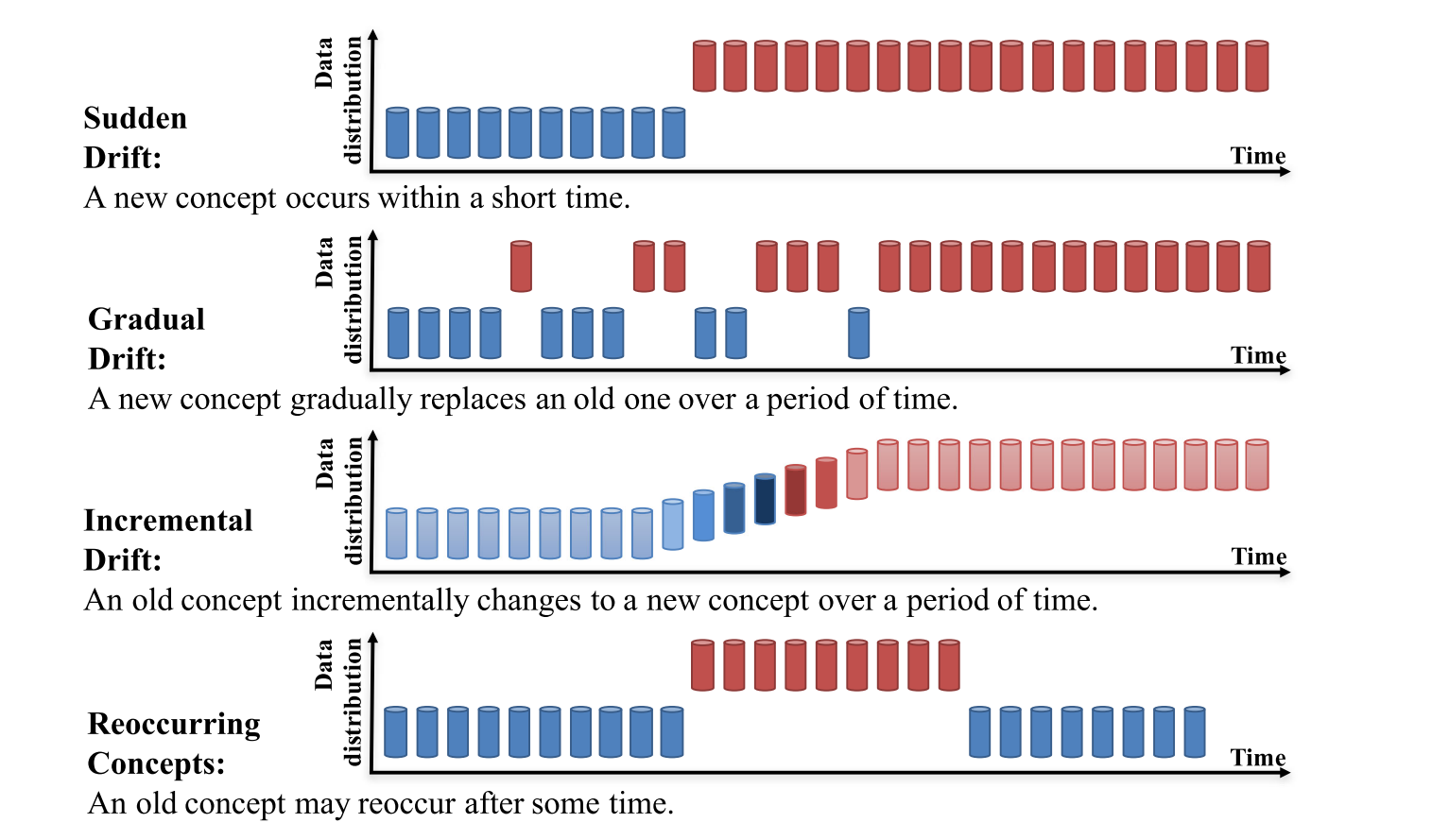


Figure 5: Types of concept drift (Lu et al., 2018)

Not all changes in data can be attributed to CD. A major challenge in CD detection is to differentiate true CD from noise and outliers. Noise and outliers are random deviations in the data, that aren’t indicative of any larger patterns and shouldn’t therefore be treated as CD (Gama et al., 2014).

Throughout the years, researchers have developed frameworks and algorithms to deal with CD. (Lu et al., 2018) proposes a framework to integrate CD handling into a production ML system (Figure 6). In this proposal CD handling consists of 3 parts: CD detection, CD understanding, and CD adaptation.

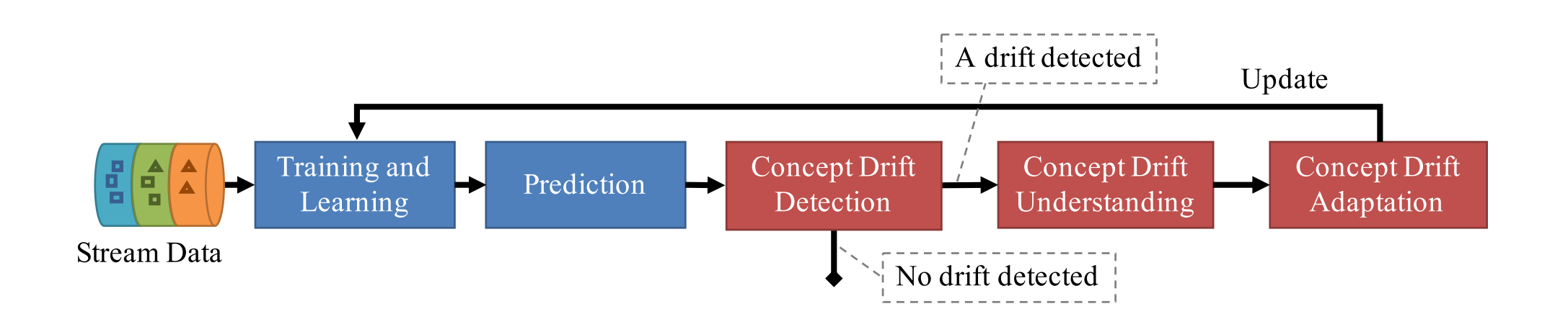


Figure 6: Framework for handling concept drift in machine learning (Lu et al., 2018)

**CD detection**. CD’s unpredictable and inconspicuous nature makes it often hard to identify. CD detection employs techniques and mechanisms to characterize and quantify CD. CD can be uncovered in different ways.

The most common way to detect CD is through observation of model performance in the productive environment. This type of CD detection is referred to as error *rate-based drift detection*. A performance decrease over time is indicative that characteristics of the data have shifted. Different algorithms can be employed to analyze CD using the model performance metrics. Drift Detection Method (DDM) was the first algorithm to assess the severity of the CD at hand, based on how the model performed over a set time window (Gama, Medas, Castillo, & Rodrigues, 2004). The window defines the timeframe of past model performance, which the current model performance is compared against. If DDM detects a significant error-rate increase, compared to the old data, it’ll declare that CD has occurred. Built upon DMM, other algorithms have been developed, such as the Early Drift Detection Method (EDDM) and the Fuzzy Windowing Drift Detection Method (FW-DDM) (Baena-Garcıa et al., 2006; Liu, Zhang, & Lu, 2017). In the context of automated CT pipelines, it becomes especially important to configure appropriate values for CD detection. The right performance metrics need to be monitored and the right performance thresholds need to be selected. Additionally, the performance needs to be observed over the right time window. Are these values not set correctly, CD detection might become too sensitive to noise. Conversely, the CD detection might pick up on CD too late.

Instead of using model performance to determine CD, the dataset can be analyzed. This category of algorithms are called *data distribution-based drift detection*. In this case, severity is quantified using distance functions to measure dissimilarities between new and old data (Lu et al., 2018). Density based algorithms and Principle Components Analysis (PCA)-based algorithms can be employed for CD detection (Feng Gu, Zhang, Jie Lu, & Chin-Teng Lin, 2016; Qahtan, Alharbi, Wang, & Zhang, 2015).

**CD understanding**. CD understanding deals with the extraction of important information about the CD at hand. CD understanding usually is the output of CD detection algorithms. Additional understanding can be acquired by employing additional methods, like visualization. According to (Lu et al., 2018) following questions should be answered in order to gain a comprehensive CD understanding:

* When did CD occur and how long does it last? (When)
* How severe is the concept drift? (How severe)
* What were the drift regions of CD? (Where)

Identifying when CD occurs is the baseline requirement for every CD detection method and therefore the most trivial question to answer. Detecting CD implicitly contains information about when the CD at hand was detected. Through continuous monitoring of the data and model, the duration of CD can be measured as well.

In order to satisfy the second question, CD needs to be quantified in a non-binary fashion. The severity of CD is measured by most dedicated CD detection algorithms, like DDM. As addressed in CD detection, the severity can be derived from diverging model performance and dissimilarities between old and new data. The steeper the performance drop, or the bigger the distance between the data old and new data, the higher the severity level of the CD.

The most difficult to answer question is *Where*. Drift regions are regions of conflict between the new and old distribution. Feature vectors located in a drift region are affected by the CD and contribute to the error-rate of the model. Identifying drift regions makes CD more tangible and allows for a more in-depth analysis of model performance. The techniques to make out drift regions are however highly dependent on the employed drift detection model. This means that there is no generalizable way to map out drift regions in the feature space (Lu et al., 2018).

Data understanding doesn’t need to be collected algorithmically alone. Insights into occurring shifts in the data can be visualized as well (Koren, 2009b). Visualizations can analyzed by data scientists to qualitatively assess CD in a dataset.

**CD adaptation**. The most conventional way of reacting to CD is to trigger a model retraining process. In the context of MLOps this is referred to as CT. After the defined performance threshold is not met, a new model is trained with new data. There are variations to simple retraining. One is called *ensemble retraining*. After CD is detected a new model gets trained, which is then added to an ensemble with the old model. Both models are then used to make predictions. Ensemble retraining is suitable for gradual CD, where two concepts are present in the data.

CD adaptation can also be incorporated into the model itself. Gradual forgetting is a technique employed during model training, where training examples are weighted based on how new they are. The result is a model that is more sensitive to the characteristics of newer data (Koychev, 2004). Another way of incorporating CD adaptation is to manually embed temporal effect into the ML model (Lo, Liao, Chang, & Lee, 2018). (Koren, 2009b) identifies temporal dynamics throughout the whole dataset and incorporates them into a CF model. This research was conducted on a Netflix dataset, where it discovered that movies receive better ratings as they age. This observation can be converted into a mathematical formula and be incorporated into the RS model.

### State of the Art Technology

In order to overcome the challenges of MLOps, a wide variety of tools have been developed in the past years. Particularly, two notable open source end-to-end ML platforms are used in industry to operationalize large scale ML applications: *TensorFlow Extended* (TFX) and *MLflow*.

**TFX**. TFX is maintained by Google and used for their own ML projects. TFX compartmentalizes common tasks of a ML lifecycle into separate components. The entire ML process from data ingestion, data evaluation, data transformation, model training, model evaluation, up to model deployment is standardized and modular. Currently there are a total of 10 built-in components that are provided by TFX. The base components can be expanded by building custom components (Figure 7).

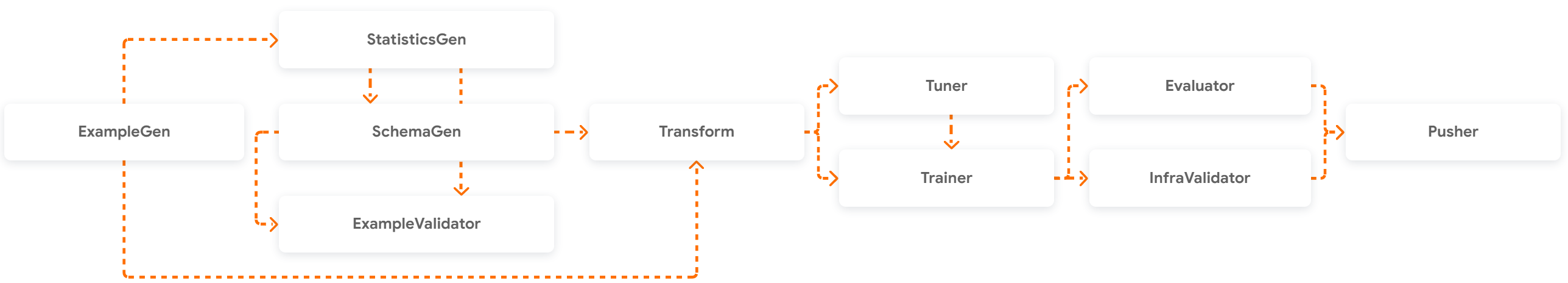


Figure 7: TFX components (Google LLC, 2019)

These components can be orchestrated and run in a pipeline with support for Airflow and Beam. In addition to the components TFX includes libraries and software components for metadata storage, model creation and model evaluation (Baylor et al., 2017). TFX’s central metadata storage is called ML metadata. This SQLite database acts as a model repository and feature store. In addition, it tracks training parameters and the order of component execution to make every pipeline run traceable. ML metadata also saves progress of model training, in order to continue the training at a later time (Crowe, 2019).

Custom components gives TFX the flexibility to embed external libraries into the pipeline, however TFX is clearly optimized to work within the TensorFlow ecosystem. This means that the TF API, TF Model Analysis (TFMA), TensorBoard and TFRS are natively supported by TFX and work without extensive configuration.

**MLflow**. MLflow is another ML platform provided by Databricks. Similar to TFX, it provides a simple API to streamline the ML lifecycle. MLflow is composed of 5 main components (Alla & Adari, 2021; Databricks, 2018; Zaharia et al., 2018).

The *MLflow Tracking* component is a logging tool to track parameters, code version and metrics during any ML process. Custom log functions can be configured and ingested into MLflow Tracking.

*MLflow project* is a format to package ML source code in a modular way. These conventions standardize the code of an ML project and make it usable in automated processes.

*MLflow Models* is a format for packaging ML models to make them compatible with external tools. This component can be used to integrate models into an application.

*MLflow Model Registry* is a centralized model repository similar to TFX. It provides additional metadata on model lineage and model version.

*MLflow Model Serving* hosts models directly from the model registry, which are callable via REST API. This feature is exclusive to the Databricks cloud service.

**KubeFlow**. KubeFlow is a solution for large scale ML operations. It provides an infrastructure for a complete MLOps environment in the cloud. It comes with essential software, like notebooks for development, TensorFlow for model creation and Apache Airflow for pipeline orchestration. In addition to built-in software packages, it also provides support for various ML tools, like TFX. KubeFlow is built on Kubernetes which handles containerization and cluster computing. This cloud infrastructure is highly scalable in nature. This means that model training can be distributed between multiple compute nodes, which accelerates the training process (Kubeflow, n.d.).

**AutoML**. Automated ML (AutoML) are services that provide end-to-end ML systems. They are provided as software-as-a-service (SaaS) on a cloud platform (e.g. GCP and AWS). The internal workings of AutoML are usually hidden to the user and the system can only be configured through pre-defined interfaces. The goal of AutoML is to provide easy-to-use and intuitive ML solutions, that don’t require extensive programming skills from the user (Karmaker et al., 2022).

# Goal & Specification

## Artifact

The goal of the DSR methodology is the design and creation of an artifact. This particular work sets out to create an automated machine learning pipeline for a RS, the main research focus being the implementation of CD-awareness into the pipeline. For evaluation purposes, a specification sheet will be generated on which the artifact will be measured against. The artifact will be compared with the features listed in the specification. Based on the specifications, the final product will be analyzed, hurdles during the design and development will be identified, and the success of this research will be determined. The specifications are structured in *base specifications* and *research specifications*. While both types of requirements are integral to the whole research project, the research requirements have a direct connection to the research question at hand. During the DSR process both requirement types receive equal prioritization as both requirement types make up the whole artifact. A table with all specifications can be seen in *table 1*.

The base requirement list consists of creating the MLOps pipeline (without explicit CD-awareness) and implementing the recommender system. MovieLens 25M will be used as the dataset, a collection of 25 million movie scores with 62.000 movies and 162.000 users (Harper & Konstan, 2016).

Table 1: Artifact specification table

|  |  |  |
| --- | --- | --- |
| **Specification type** | **Specification** | **Description** |
| Base specification | Recommender system | The artifact has deep & cross network RS made with TFRS. |
|  | Base pipeline | The artifact has a TFX pipeline for the RS. |
| Research specification | CD detection | The artifact has the ability to detect CD. |
|  | CD understanding | The artifact has the ability to extract information out of CD. |
|  | CD adaptation | The artifact has the ability to react to CD. |
|  | Integrated pipeline | The final artifact is one pipeline that can be executed in one go |
|  | Automated pipeline | The pipeline can finish without human intervention |

The research specifications have three criteria related to concept drift, which are taken from the 2018 paper “Learning under Concept Drift: A Review” (Lu et al., 2018):

*Concept drift detection* specifies the technical ability for the artifact to register concept drift in the data. An example would be the implementation of a concept drift detection algorithm.

*Concept drift understanding* specifies the ability of the artifact to retrieve additional information from the concept drift. This criteria serves to give concept drift more tangibility and helps data scientists identify the source of concept drift. The most rudimentary implementation of this specification would be a timestamp for every occurring CD. A timestamp for CD is also the minimum requirement to realize an automated pipeline, as in order to initiate concept drift adaptation the system needs to know when CD occurs.

*Concept drift adaptation* specifies an action that accounts for concept drift, like starting a retraining of the model.

Each of these components amount to what we define in this paper as CD-awareness. All three specifications will be explained in further detail in the literature review.

Besides the concept drift specifications, there are also pipeline conditions in the research specifications that envision the artifact to be one unified and automated process.

These specifications serve as an aide during the design and development of the artifact. Rather than evaluating the artifact based on how many specifications it technically fulfilled, the whole artifact will be qualitatively evaluated using the specification sheet as a guideline. In the scope of this work, one solution to CD will be implemented that is derived from the literature. It is not a comparison and evaluation of various implementations of CD-awareness. This work does not entail a quantitative evaluation of the artifact, as the focus lies on a qualitative analysis of the prototype.

## Procedure

Throughout this project various tools are used to track, organize and document this work. As part of this research, a GitHub repository was created, which contains both the thesis Word document and the software artifact.[[9]](#footnote-10) GitHub and Git are used for version control and enable work on different systems. It also serves to make the development of this research transparent and traceable. Internally, a GitHub Project Kanban board is used for project management. There, the project is broken down into individual tasks, all of which have their progress tracked. This way it is possible to gain an overview of the current status of this project and plan future development steps.

The overarching process of the artifact creation is structured in underlying phases, as seen in Figure 8. Each phase will be developed in a separate branch and then merged into the main branch, as is convention in software development.

The initial phase of this project is dedicated to the set-up of the artifact environment. This phase encompasses every necessary step to build a software environment on which the artifact will run on. The individual tasks would consist of choosing the python version, setting up a work directory and installing needed packages and other software. After setting up the environment, a state-of-the-art Recommender system will be created. In order to test the implemented RS, it will be run on the MovieLens dataset. The next phase involves building a base MLOps pipeline around the RS. The base pipeline then gets then incrementally expanded by additional components. Each pipeline component that gets added, will be tested for compatibility. Depending on what components work and don’t work, design changes to the artifact need to be made. Having implemented the RS and the pipeline, an intermediate evaluation of the current pipeline will be done. The objective in this phase is to find out the capabilities of this pipeline in relation to the research question. What components can be leveraged to fulfill the objective of this work? Accomplishing these phases should fulfill the base specifications detailed in the previous chapter. The following phase consists of incorporating CD-awareness into the pipeline. With the knowledge base from scientific papers and information about the environment from the whitepapers, a solution to CD will be engineered. This process will run iteratively and in accordance with the DSR methodology. Each iteration of this phase concludes with an evaluation, which will determine whether another iteration is initiated. Once the artifact reaches a state where no additional iterations are done, a final evaluation of the artifact will be made, which concludes this research project.

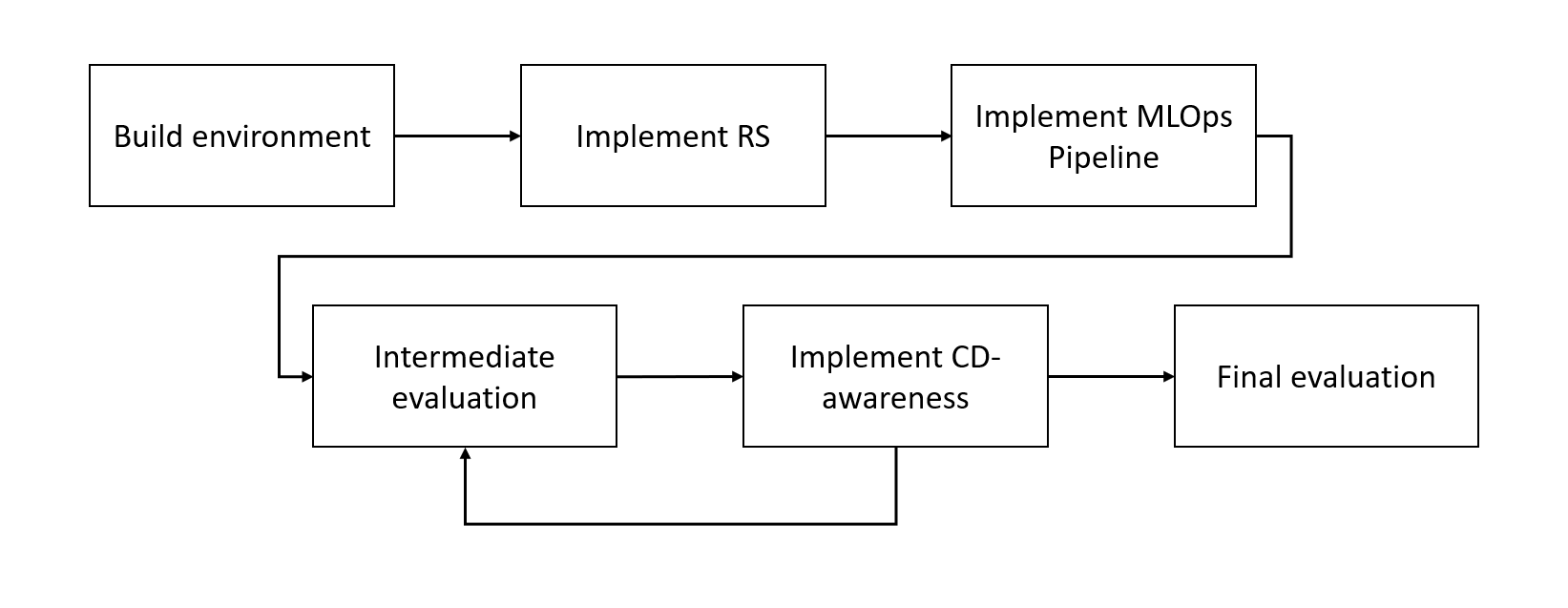


Figure 8: procedure of artifact development

## Challenges

This work is not done in association with a company, despite being a study directed towards businesses exploring a potential solution to CD in an MLOps scenario. Consequently this research can't be conducted in an existing enterprise infrastructure. This entails the challenge that, first, an environment needs to be built approximating a productive business setting. For instance, instead of using real world data from a data lake or data warehouse, this research needs to revert to publicly available datasets. Simulating a productive MLOps environment means that there is a discrepancy between this work's environment and an organization's environment, which might impact applicability of the artifact.

The open-ended nature of this work’s research question is another source of challenge that is inherent to DSR projects. In Hevner’s DSR methodology it is only specified what problem the final artifact should solve, not how it should look like, as it is part of the process to come up with an innovative design. This room for interpretation of what the artifact should look like makes it more difficult to quantify it in the end. For this reason a specification has been created, which should make the idea of the artifact more concrete. Even with a specification sheet, the challenge remains on how to meaningfully evaluate a ML pipeline. Unlike ML algorithms, there is no reference pipeline to measure the artifact against, assuming that using quantitative metrics like processing time would make sense for this work in the first place. Another aspect that hinders quantifiability of such an artifact is the phenomena of CD itself. As mentioned in the environment chapter, CD is a burden for businesses because it lacks tangibility and is therefore hard notice, let alone isolate and quantitatively measure. This project needs to deal with these limitations and find a way to evaluate the artifact in a value adding manner.

Another hurdle is a lack of scientific studies and official documentation on this specific subject, as the operationalization of ML systems only received mainstream attention in the last few years with the advent of terms like AutoML and MLOps, or tools like TFX. In the following, TFX and TFRS are chosen to illustrate the problem. This example is however applicable to other tools as well.

TFX had its first release in May 2019 (Google LLC), whereas TFRS had its initial release in fall of 2020 (Google LLC). Only in end of March 2022 has Google released its first TFX documentation using TFRS (Google LLC, 2022). As of conducting this project, more fundamental documentation is planned for both tools (Wei, 2022). It would be defensible to say that both tools still are in their infancy, going by their recent releases, their ever expanding documentation and active development. Besides the novelty of these fields, a lot of research and development in ML systems goes unnoticed by the academic community, as they are conducted behind closed doors and used as proprietary software. In 2019, Naumov, et al. stated the following about the current environment of RS research: “*Although recommendation and personalization systems still drive much practical success of deep learning within industry today, these networks continue to receive little attention in the academic community*” (Naumov et al., 2019)

# Design & Development

## Hardware & Software Environment

This research project is conducted on a deep learning compute cluster provided by the Media University Stuttgart (Theodoridis & Grießhaber, n.d.). The hardware resources are allocated by Slurm, a cluster management software. This way, multiple users can access powerful hardware to train large DL models. The DL cluster consists of multiple individual servers, which can be classified as either a CPU partition or GPU partition. While CPU compute nodes are sufficient for development and data preparation tasks, GPU acceleration can be leverages to rapidly compute tensor calculations.

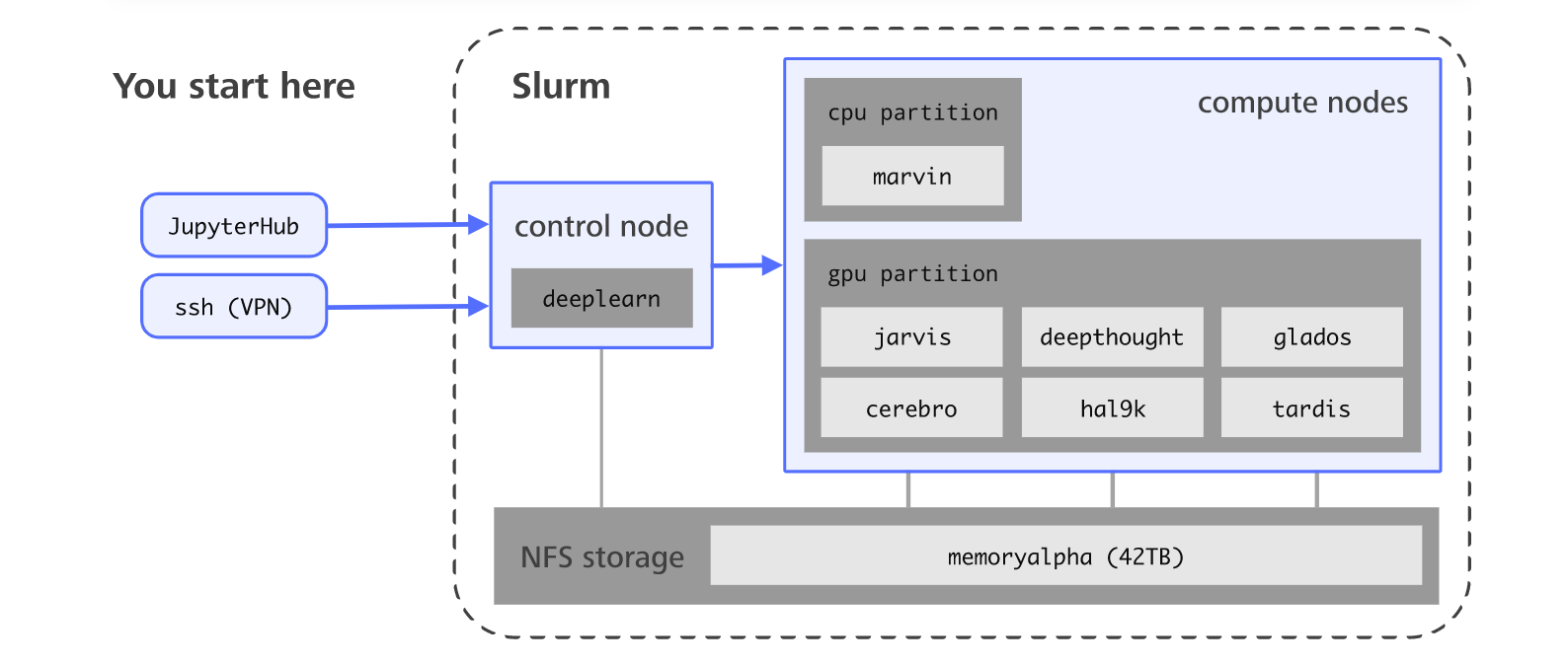


Figure 9: Infrastructure of the HdM deeplearning cluster (Theodoridis & Grießhaber, n.d.)

These servers are accessed via the control node, which acts as the entry point to the cluster. The control node can either be addressed via a web browser through JupyterHub or with secure shell (SSH), in combination with a VPN. The cluster has a shared network file system (NFS), where the files from all users are stored (Figure 9). In order to start a server, the user needs to select either a CPU or GPU instance (Figure 10). Depending on the selection, the user gets assigned a corresponding cluster node. Each time a server instance is started, an Ubuntu image gets mounted to the users work directory. This image comes with Anaconda preinstalled, in order to create and manage Python environments.

Python will be used as the main programming language of this project, as it is built-in into Anaconda and supports a wide array of data science packages (e.g. TensorFlow, NumPy, pandas) to conduct this research. Specifically, Python version 3.8 is chosen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 10: selection of instance in the HdM deeplearning cluster

During the assessment of a suitable research environment, KubeFlow was briefly considered. As mentioned in the “State of the Art Technology” chapter in “MLOps”, KubeFlow would provide an all-in-one environment for this research. KubeFlow is a viable solution for large scale MLOps in an enterprise setting, but wouldn’t be appropriate for the prototype that is developed in this research. The set up and configuration of the environment would exceed the scope of this project. KubeFlow instantiates various containers with ML software, which are intelligently distributed across a server cluster with Kubernetes. While this makes for a scalable productive system, it also has high hardware requirements.

For this artifact, a lean software stack of few selected tools are sufficient to conduct this research. In the following, the tool selection and the implementation of the individual artifact components are discussed in greater detail.

## Recommender System

### Tools

### Implementation

**Data**.

**Recommender System**.

### (Challenges)

## Concept Drift Awareness

## MLOps Pipeline

# Evaluation

# Conclusion

## Design Science Research

## Recommender Systems

### Overview

# Conclusion and Outlook

Mit der hier vorgelegten Dokumentvorlage steht ein Werkzeug zur Verfügung, das es möglich macht, auf relativ einfache Weise die Textgestaltung für Abschlussarbeiten und andere wissenschaftliche Arbeiten zu automatisieren. Dies spart Arbeitszeit, Beratungsaufwand und erhöht zugleich die Qualität des Ergebnisses hinsichtlich formaler Kriterien.

Gleichzeitig unterstützt die Dokumentvorlage die Lehre in Fächern *wie Arbeits-, Lern- und Präsentationstechniken* und *Vorbereitung auf die Bachelor- bzw. Masterarbeit*,

Die Dokumentvorlage wurde im Sommer 2000 in der vorliegenden grundlegend überarbeiteten Version zum ersten Mal hochschulweit benutzt, nachdem bereits die Vorgängerversion im Jahr 1999 für drei Abschlussarbeiten erfolgreich eingesetzt wurde.

Erweiterungen der Dokumentvorlage und der darin enthaltenen Anleitung zur Erstellung von Abschlussarbeiten mit dem Textsystem Microsoft Word sind auch für die Zukunft geplant. Das Feedback aus der Nutzung der Dokumentvorlage für Abschlussarbeiten an der HdM wird dabei fortgesetzt aufgegriffen und zur Verbesserung genutzt werden.

# Anhang A: Beispiele für die Gliederung von Abschlussarbeiten

Die nachfolgenden Gliederungen stellen lediglich Vorschläge dar, die stets am konkreten Fall überprüft und in der Regel angepasst werden müssen.

## A.1 Literaturarbeiten

1. Überblick (oder: Zusammenfassung, „Executive Summary“, alles Wichtige für den „Manager“ oder Schnellleser)
2. Fragestellung (oder: Ziele, Ausgangspunkt, Motivation)
3. Übersicht über den Stand der Wissenschaft und Technik (Beschreibung der Lösungsansätze, Beispiele etc. in einzelnen Abschnitten)
4. Bewertung der einzelnen untersuchten Ansätze, Beispiele etc., Identifikation von Defiziten
5. Synthese: Erstellung einer Gesamtschau, allgemeine Prinzipien, Beschreibung einer eigenen Sicht auf das Problem, evtl. auch eigene Vorschläge
6. Zusammenfassung (Erklärung des Nutzens), Ausblick

Anhang: eventuell recherchierte Texte, Produktbeschreibungen, etc.

## A.2 Systementwicklungen

1. Überblick (oder: Zusammenfassung, „Executive Summary“, alles Wichtige für den „Manager“ oder Schnellleser)
2. Problemstellung (oder: Ziele, Ausgangspunkt), Vorgesehener Benutzerkreis, Bedürfnisse der Benutzer
3. Stand der Technik (Wie wird das Problem bisher gelöst, wo sind die Defizite)
4. Gewählter Lösungsansatz (allgemeines Prinzip, welche Werkzeuge, z.B. Programmiersprachen werden verwendet)
5. Beschreibung der durchgeführten Arbeiten
6. Ergebnis (z.B. Screenshots mit Erläuterungen)
7. Zusammenfassung (Erklärung des Nutzens), Ausblick

Anhang: evtl. (ausgewählte) Programmbeispiele

Evtl. CD-ROM als Beilage

# Anhang B: Formatvorlagen

Nachfolgend sind die für die Benutzer der Dokumentvorlage wichtigsten Formatvorlagen aufgelistet. Vermerkt ist jeweils auch der Zweck der Formatvorlage und ob es sich um eine neue oder eine modifizierte Formatvorlage handelt.

# Index

Abbildung 25

Abbildungsverzeichnis 18

Abkürzungsverzeichnis 18

Absatz 35, 44

Absatzmarken 39

Abstand zwischen Absätzen 35

Abstract 18

Angebotsorientiertheit 12

Anhänge 19

Anleitungen 13

Ansicht 37

Arbeitserleichterung 12

Aufzählungen 30

Auto-Wiederherstellen 38

Beschriftung 25

Bildschirmabzüge 25

Bindestrich 36

Computerprogramm 29

Datei-Info 39

Dokumentvorlage 13, 44

Ebenen 31

Eidesstattliche Versicherung 18

Einrückungen 30

Fehler 37

Fließtext 24

Formatvorlage 14, 22, 44

Formatvorlagen-Katalog 33

Funktionalitäten 22

Fußnoten 25

Glossar 20

Index 20

Inhalt der Arbeit 19

Inhaltsverzeichnis 18

Keywords 18

Kopfzeile 23

Kurzfassung 18

Leerzeichen 39

Literaturverzeichnis 33

Muster 14, 16

nicht druckbare Zeichen 20, 36

Nummerierungen 30

Qualitätssicherung 12

Quellenangabe 29

Rechtschreibprüfung 25

Rechtschreibung 16, 35, 39

Schlagwörter 18

Schnellspeicheroption 38

Schriftart 24

Seiteneinrichtung 23

Serifenschrift 24

Sicherungen 37

Sichtbarkeit 36

Silbentrennung 25, 36, 39

Speichern 37

Standard (Formatvorlage) 24

Stichwortverzeichnis 20

Tabellen 27

Tabellenüberschrift 28

Tabellenverzeichnis 18

Titelblatt 17

Trennstriche 36

Überschriften 31

Untertitel 17

Vorgaben 13

Vorwort 19

Word-Dokument 14

Word-Dokumentvorlage 14, 33

Zitat 28

References

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Cham: Springer International Publishing.

Algorithmia. *2020 state of enterprise machine learning*. Retrieved from https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia\_2020\_State\_of\_Enterprise\_ML.pdf

Alla, S., & Adari, S. K. (2021). *Beginning MLOps with MLFlow*. Springer.

Alyari, F., & Jafari Navimipour, N. (2018). Recommender systems. *Kybernetes*, *47*(5), 985–1017. https://doi.org/10.1108/K-06-2017-0196

Arrikto (n.d.). Arrikto Enterprise Kubeflow Documentation. Retrieved from https://docs.arrikto.com/

Baena-Garcıa, M., Del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales-Bueno, R. (2006). Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*.

Baker, T. (2019). *Smarter Humans. Smarter Machines.* Retrieved from Refinitiv website: https://www.refinitiv.com/en/resources/special-report/refinitiv-2019-artificial-intelligence-machine-learning-global-study

Baylor, D., Breck, E., Cheng, H.‑T., Fiedel, N., Foo, C. Y., Haque, Z., . . . Zinkevich, M. (2017). TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In S. Matwin, S. Yu, & F. Farooq (Eds.), *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1387–1395). New York, NY, USA: ACM. https://doi.org/10.1145/3097983.3098021

Bennett, J., Lanning, S., & others (2007). The netflix prize. In *Proceedings of KDD cup and workshop.* Symposium conducted at the meeting of Citeseer.

Blondel, M., Fujino, A., Ueda, N., & Ishihata, M. (2016, July 25). *Higher-Order Factorization Machines*. Retrieved from http://arxiv.org/pdf/1607.07195v2

Blondel, M., Ishihata, M., Fujino, A., & Ueda, N. (2016, July 29). *Polynomial Networks and Factorization Machines: New Insights and Efficient Training Algorithms*. Retrieved from http://arxiv.org/pdf/1607.08810v1

Cai, L., & Zhu, Y. (2015). The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal*, *14*(0), 2. https://doi.org/10.5334/dsj-2015-002

Charniak, E. (2019). *Introduction to Deep Learning*. The MIT Press.

Choy, G., Khalilzadeh, O., Michalski, M., Do, S., Samir, A. E., Pianykh, O. S., . . . Dreyer, K. J. (2018). Current Applications and Future Impact of Machine Learning in Radiology. *Radiology*, *288*(2), 318–328. https://doi.org/10.1148/radiol.2018171820

Chui, M., Hall, B., Singla, A., & Sukharevsky, A. (2021, December 8). *The state of AI in 2021*. Retrieved from McKinsey website: https://www.mckinsey.com/business-functions/quantumblack/our-insights/global-survey-the-state-of-ai-in-2021

Columbus, L. (2017, July 9). McKinsey's State Of Machine Learning And AI, 2017. *Forbes*. Retrieved from https://www.forbes.com/sites/louiscolumbus/2017/07/09/mckinseys-state-of-machine-learning-and-ai-2017/?sh=63414b1b75b6

Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. In S. Sen, W. Geyer, J. Freyne, & P. Castells (Eds.), *Proceedings of the 10th ACM Conference on Recommender Systems* (pp. 191–198). New York, NY, USA: ACM. https://doi.org/10.1145/2959100.2959190

Crowe, R. (2019). Why do I need metadata? (TensorFlow Extended). Retrieved from https://www.youtube.com/watch?v=cc1-eocgm1E

Cunningham, W. (1993). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*, *4*(2), 29–30. https://doi.org/10.1145/157710.157715

Dacrema, M. F., Boglio, S., Cremonesi, P., & Jannach, D. (2021). A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research. *ACM Transactions on Information Systems*, *39*(2), 1–49. https://doi.org/10.1145/3434185

Databricks (2018). MLflow Documentation. Retrieved from https://mlflow.org/docs/latest/index.html

Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, . . . Martin Zinkevich (2019). Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)* (pp. 51–53). Santa Clara, CA: USENIX Association. Retrieved from https://www.usenix.org/conference/opml19/presentation/baylor

Feng Gu, Zhang, G., Jie Lu, & Chin-Teng Lin (2016). Concept drift detection based on equal density estimation. In *2016 International Joint Conference on Neural Networks (IJCNN)*.

Fleder, D. M., & Hosanagar, K. (2007). Recommender systems and their impact on sales diversity. In J. MacKie-Mason, D. Parkes, & P. Resnick (Eds.), *Proceedings of the 8th ACM conference on Electronic commerce - EC '07* (p. 192). New York, New York, USA: ACM Press. https://doi.org/10.1145/1250910.1250939

Fowler, M. (2019). *Refactoring: Improving the design of existing code* (Second edition). *Addison-Wesley signature series*. Boston: Addison-Wesley.

Gama, J. [João], Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with Drift Detection. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, . . . S. Labidi (Eds.), *Lecture Notes in Computer Science. Advances in Artificial Intelligence – SBIA 2004* (Vol. 3171, pp. 286–295). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-28645-5\_29

Gama, J. [João], Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, *46*(4), 1–37. https://doi.org/10.1145/2523813

Garousi, V., Petersen, K., & Ozkan, B. (2016). Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. *Information and Software Technology*, *79*, 106–127. https://doi.org/10.1016/j.infsof.2016.07.006

GOODFELLOW, I., BENGIO, Y., & COURVILLE, A. (2016). *Deep learning*. *Adaptive computation and machine learning series*. Cambridge, Massachusetts, London: MIT Press.

Google LLC. TFRS Releases. Retrieved from https://github.com/tensorflow/recommenders/releases?page=2

Google LLC. TFX Releases. Retrieved from https://github.com/tensorflow/tfx/releases?page=8

Google LLC (2019). TFX: TensorFlow Extended (TFX) is an end-to-end platform for deploying production ML pipelines. Retrieved from https://www.tensorflow.org/tfx

Google LLC (2020). MLOps: Continuous delivery and automation pipelines in machine learning.

Google LLC (2021). Collaborative Filtering. Retrieved from https://developers.google.com/machine-learning/recommendation/collaborative/basics

Google LLC (2022). Using TensorFlow Recommenders with TFX. Retrieved from https://www.tensorflow.org/recommenders/examples/ranking\_tfx

Greenberg, D., Taylor, C., Ivchenko, D., Liu, X., & Sudarshan, A. (2022). Introducing TorchRec, a library for modern production recommendation systems. Retrieved from https://pytorch.org/blog/introducing-torchrec/

Gurney, K. (2014). *An Introduction to Neural Networks*. Hoboken: CRC Press.

Hanin, B. (2019). Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations. *Mathematics*, *7*(10), 992. https://doi.org/10.3390/math7100992

Harper, F. M., & Konstan, J. A. (2016). The MovieLens Datasets. *ACM Transactions on Interactive Intelligent Systems*, *5*(4), 1–19. https://doi.org/10.1145/2827872

Heaton, J. (2012). *Introduction to the Math of Neural Networks*. Heaton Research.

Heemstra, F. J. (1992). Software cost estimation. *Information and Software Technology*, *34*(10), 627–639. https://doi.org/10.1016/0950-5849(92)90068-Z

Hevner, March, Park, & Ram (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75. https://doi.org/10.2307/25148625

Hevner, A. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, *19*.

Hevner, A., & Chatterjee, S. (Eds.) (2010). *Integrated Series in Information Systems*. *Design Research in Information Systems*. Boston, MA: Springer US. https://doi.org/10.1007/978-1-4419-5653-8

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012, July 3). *Improving neural networks by preventing co-adaptation of feature detectors*. Retrieved from http://arxiv.org/pdf/1207.0580v1

Huang, T., She, Q., Wang, Z., & Zhang, J. [Junlin] (2020, July 6). *GateNet: Gating-Enhanced Deep Network for Click-Through Rate Prediction*. Retrieved from http://arxiv.org/pdf/2007.03519v1

International Air Transport Association (2022). *The impact of the war in Ukraine on the aviation industry*. Retrieved from https://www.iata.org/en/iata-repository/publications/economic-reports/the-impact-of-the-conflict-between-russia-and-ukraine-on-aviation/

Jannach, D., & Zanker, M. (2022). Value and Impact of Recommender Systems. In F. Ricci, L. Rokach, & B. Shapira (Eds.), *Recommender Systems Handbook* (pp. 519–546). New York, NY: Springer US. https://doi.org/10.1007/978-1-0716-2197-4\_14

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science (New York, N.Y.)*, *349*(6245), 255–260. https://doi.org/10.1126/science.aaa8415

Kakantousis, T., Kouzoupis, A., Buso, F., Berthou, G., Dowling, J., & Haridi, S. (2019). Horizontally scalable ml pipelines with a feature store. In *Proceedings of the 2nd SysML Conference, Palo Alto, CA, USA*.

Karmaker, S. K., Hassan, M. M., Smith, M. J., Xu, L., Zhai, C., & Veeramachaneni, K. (2022). AutoML to Date and Beyond: Challenges and Opportunities. *ACM Computing Surveys*, *54*(8), 1–36. https://doi.org/10.1145/3470918

Khusro, S., Ali, Z., & Ullah, I. (2016). Recommender Systems: Issues, Challenges, and Research Opportunities. In K. J. Kim & N. Joukov (Eds.), *Lecture Notes in Electrical Engineering. Information Science and Applications (ICISA) 2016* (Vol. 376, pp. 1179–1189). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-10-0557-2\_112

Knotek, J., & Pereira, W. (2011). Survey on Concept Drift. Retrieved from https://is.muni.cz/el/1433/podzim2011/PA164/um/drift\_detection\_methods.pdf

Koren, Y. (2009a). The bellkor solution to the netflix grand prize. *Netflix Prize Documentation*, *81*(2009), 1–10.

Koren, Y. (2009b). Collaborative Filtering with Temporal Dynamics. In *KDD ’09, Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 447–456). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1557019.1557072

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37.

Koren, Yehuda and Rendle, Steffen and Bell, Robert (2022). Advances in Collaborative Filtering. In Ricci, Francesco and Rokach, Lior and Shapira, Bracha (Ed.), *Recommender Systems Handbook* (pp. 91–142). New York, NY: Springer US. https://doi.org/10.1007/978-1-0716-2197-4\_3

Koychev, I. (2004). Gradual Forgetting for Adaptation to Concept Drift. *ECAI*.

Kreuzberger, D., Kühl, N., & Hirschl, S. (2022, May 4). *Machine Learning Operations (MLOps): Overview, Definition, and Architecture*. Retrieved from http://arxiv.org/pdf/2205.02302v3

Kubeflow (n.d.). Kubeflow. Retrieved from https://www.kubeflow.org/

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Handwritten Digit Recognition with a Back-Propagation Network. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems* (Vol. 2). Morgan-Kaufmann. Retrieved from https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf

Lian, J., Zhou, X., Zhang, F., Chen, Z., Xie, X., & Sun, G. (2018). xDeepFM. In Y. Guo & F. Farooq (Eds.), *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 1754–1763). New York, NY, USA: ACM. https://doi.org/10.1145/3219819.3220023

Liu, A., Zhang, G., & Lu, J. (2017). Fuzzy time windowing for gradual concept drift adaptation. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE).* Symposium conducted at the meeting of IEEE.

Lo, Y.‑Y., Liao, W., Chang, C.‑S., & Lee, Y.‑C. (2018). Temporal Matrix Factorization for Tracking Concept Drift in Individual User Preferences. *IEEE Transactions on Computational Social Systems*, *5*(1), 156–168. https://doi.org/10.1109/TCSS.2017.2772295

Loehlin, J. C., & Beaujean, A. A. (2017). *Latent variable models: An introduction to factor, path, and structural equation analysis* (5. ed.). New York: Routledge.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J. [Joao], & Zhang, G. (2018). Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*, 1. https://doi.org/10.1109/TKDE.2018.2876857

Lü, L., Medo, M., Yeung, C. H., Zhang, Y.‑C., Zhang, Z.‑K., & Zhou, T. (2012). Recommender systems. *Physics Reports*, *519*(1), 1–49. https://doi.org/10.1016/j.physrep.2012.02.006

Lui, M., Yetim, Y., Özkan, Ö., Zhao, Z., Tsai, S.‑Y., Wu, C.‑J., & Hempstead, M. (2020, November 4). *Understanding Capacity-Driven Scale-Out Neural Recommendation Inference*. Retrieved from http://arxiv.org/pdf/2011.02084v2 https://doi.org/pages

Makinen, S., Skogstrom, H., Laaksonen, E., & Mikkonen, T. (2021). Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help? In *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)* (pp. 109–112). IEEE. https://doi.org/10.1109/WAIN52551.2021.00024

May, R. J., Maier, H. R., & Dandy, G. C. (2010). Data splitting for artificial neural networks using SOM-based stratified sampling. *Neural Networks : The Official Journal of the International Neural Network Society*, *23*(2), 283–294. https://doi.org/10.1016/j.neunet.2009.11.009

Meta (2022). Inside the Lab: AI breakthroughs that will power the metaverse. Retrieved from https://ai.facebook.com/blog/meta-ai-inside-the-lab/

Microsoft (n.d.). Machine Learning operations maturity model. Retrieved from https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model

Minsky, M. (1961). Steps toward Artificial Intelligence. *Proceedings of the IRE*, *49*(1), 8–30. https://doi.org/10.1109/JRPROC.1961.287775

Miranda, L. J. (2021). Towards data-centric machine learning: a short review. *Ljvmiranda921. Github. Io*.

Myers, B. L., Kappelman, L. A., & Prybutok, V. R. (1997). A Comprehensive Model for Assessing the Quality and Productivity of the Information Systems Function. *Information Resources Management Journal*, *10*(1), 6–26. https://doi.org/10.4018/irmj.1997010101

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Icml*.

Naumov, M., Mudigere, D., Shi, H.‑J. M., Huang, J., Sundaraman, N., Park, J., . . . Smelyanskiy, M. (2019, May 31). *Deep Learning Recommendation Model for Personalization and Recommendation Systems*. Retrieved from http://arxiv.org/pdf/1906.00091v1

Ng, A., Crowe, R., & Moroney, L. (2021). Introduction to Machine Learning in Production. Retrieved from https://de.coursera.org/learn/introduction-to-machine-learning-in-production?specialization=machine-learning-engineering-for-production-mlops

Qahtan, A. A., Alharbi, B., Wang, S., & Zhang, X. [Xiangliang] (2015). A PCA-Based Change Detection Framework for Multidimensional Data Streams: Change Detection in Multidimensional Data Streams. In *KDD ’15, Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 935–944). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2783258.2783359

Refinitiv (2020). *THE RISE OF THE DATA SCIENTIST:: Machine learning models for the future*. Retrieved from https://www.refinitiv.com/en/resources/special-report/refinitiv-2020-artificial-intelligence-machine-learning-global-study

Rendle, S., Krichene, W., Zhang, L., & Anderson, J. (2020). Neural Collaborative Filtering vs. Matrix Factorization Revisited. In *Fourteenth ACM Conference on Recommender Systems* (pp. 240–248). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3383313.3412488

Renggli, C., Rimanic, L., Gürel, N. M., Karlaš, B., Wu, W., & Zhang, C. (2021, February 15). *A Data Quality-Driven View of MLOps*. Retrieved from http://arxiv.org/pdf/2102.07750v1

Rimol, M. (2021, November 22). *Gartner Forecasts Worldwide Artificial Intelligence Software Market to Reach $62 Billion in 2022*. Retrieved from Gartner website: https://www.gartner.com/en/newsroom/press-releases/2021-11-22-gartner-forecasts-worldwide-artificial-intelligence-software-market-to-reach-62-billion-in-2022

Rosenblatt, F. (1961). *Principles of neurodynamics. perceptrons and the theory of brain mechanisms.*

Ruf, P., Madan, M., Reich, C., & Ould-Abdeslam, D. (2021). Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools. *Applied Sciences*, *11*(19), 8861. https://doi.org/10.3390/app11198861

Salama, K., Kazmierczak, J., & Schut, D. (2021). Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning. Retrieved from https://services.google.com/fh/files/misc/practitioners\_guide\_to\_mlops\_whitepaper.pdf

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., . . . Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, *28*(2), 2503–2511.

Serban, A., van der Blom, K., Hoos, H., & Visser, J. (2020). Adoption and Effects of Software Engineering Best Practices in Machine Learning, *1*, 1–12. https://doi.org/10.1145/3382494.3410681

Shan, Y., Hoens, T. R., Jiao, J., Wang, H., Yu, D., & Mao, J. C. (2016). Deep Crossing. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, & R. Rastogi (Eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 255–262). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939704

Singh, P. K., Choudhury, P., Dey, A. K., & Pramanik, P. K. D. (2021). Recommender systems: an overview, research trends, and future directions. *International Journal of Business and Systems Research*, *15*(1), 14. https://doi.org/10.1504/ijbsr.2021.10033303

Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Müller, K.‑R. (2021). Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. *Machine Learning and Knowledge Extraction*, *3*(2), 392–413. https://doi.org/10.3390/make3020020

Su, X., & Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, *2009*, 1–19. https://doi.org/10.1155/2009/421425

Symeonidis, G., Nerantzis, E., Kazakis, A., & Papakostas, G. A. (2022). MLOps - Definitions, Tools and Challenges. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 453–460). IEEE. https://doi.org/10.1109/CCWC54503.2022.9720902

Tamburri, D. A. (2020). Sustainable MLOps: Trends and Challenges. In *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)* (pp. 17–23). IEEE. https://doi.org/10.1109/SYNASC51798.2020.00015

TensorFlow (n.d.). TensorFlow Recommenders. Retrieved from https://www.tensorflow.org/recommenders

TensorFlow (2022a). Deep & Cross Network (DCN). Retrieved from https://www.tensorflow.org/recommenders/examples/dcn

TensorFlow (2022b). TFRS API: All symbols in TensorFlow Recommenders. Retrieved from https://www.tensorflow.org/recommenders/api\_docs/python/tfrs/all\_symbols

Theodoridis, J., & Grießhaber, D. (n.d.). Deeplearning Cluster - Docs. Retrieved from https://deeplearn.pages.mi.hdm-stuttgart.de/docs/

Vellido, A., Lisboa, P. J., & Meehan, K. (2000). Quantitative Characterization and Prediction of On-Line Purchasing Behavior: A Latent Variable Approach. *International Journal of Electronic Commerce*, *4*(4), 83–104. https://doi.org/10.1080/10864415.2000.11518380

Wang, R., Fu, B., Fu, G., & Wang, M. (2017, August 17). *Deep & Cross Network for Ad Click Predictions*. Retrieved from http://arxiv.org/pdf/1708.05123v1 https://doi.org/AdKDD

Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., & Chi, E. (2021). DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In J. Leskovec, M. Grobelnik, M. Najork, J. Tang, & L. Zia (Eds.), *Proceedings of the Web Conference 2021* (pp. 1785–1797). New York, NY, USA: ACM. https://doi.org/10.1145/3442381.3450078

Wei, W. (2022, June 9). Question: examples/best practices for integrating tfrs with tfx #380. Retrieved from https://github.com/tensorflow/recommenders/issues/380#issuecomment-1151934730

Yan, Y., & Li, L. (2020). xDeepInt: a hybrid architecture for modeling the vector-wise and bit-wise feature interactions.

Yavuz, B., & Chockalingam, P. (2019). Introducing Delta Time Travel for Large Scale Data Lakes. Retrieved from https://databricks.com/de/blog/2019/02/04/introducing-delta-time-travel-for-large-scale-data-lakes.html

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., . . . others (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*(4), 39–45.

Zhang, Y., Chen, L., Yang, S., Yuan, M., Yi, H., Zhang, J. [Jie], . . . Zheng, B. (2022, April 11). *PICASSO: Unleashing the Potential of GPU-centric Training for Wide-and-deep Recommender Systems*. Retrieved from http://arxiv.org/pdf/2204.04903v2

1. Throughout this paper ML and AI will be used interchangeably. [↑](#footnote-ref-2)
2. Rank refers to the dimensionality of a latent factor for Matrix and for Matrix , low-rank meaning that . represents the amount of linearly independent factors in a latent factor (Aggarwal (2016). [↑](#footnote-ref-3)
3. NNs and ANNs will be used interchangeably in this paper. [↑](#footnote-ref-4)
4. The Softmax function maps the neural output on range [0;1]. Highly negative inputs approach value 0, while highly positive inputs approach value 1. [↑](#footnote-ref-5)
5. In dropout regularization random neurons are skipped during the training process to combat overfit of the model (Hinton, Srivastava, Krizhevsky, Sutskever, and Salakhutdinov (2012). [↑](#footnote-ref-6)
6. Explicit feature interactions are usually retrieved by dedicated components in the model, such as the cross network in DCN. The order of feature interaction can be set (bounded-degree) and inferred by said component, which makes the feature extraction more controllable and predictable. On the contrary, implicit feature interactions can have an arbitrary order, as it is not explicitly defined in the model to which degree the interaction is bound to (i.e. DNN). Implicit feature extraction is more flexible at picking up any patterns in the data, at the cost of certainty of the degree of the feature interaction (Lian et al. (2018); Huang, She, Wang, and Zhang (2020); Yan and Li (2020). [↑](#footnote-ref-7)
7. A feature store is a data management layer that tracks and maintains features in ML data. Feature stores simplify ML pipelines and the data engineering process, by providing a centralized space where all raw and transformed data features are stored in. Previous data transformation steps are saved in the feature store, where they can be reused by other data scientists (Kakantousis et al. (2019). [↑](#footnote-ref-8)
8. Distribution and concept are used synonymously in the context of concept drift. [↑](#footnote-ref-9)
9. https://github.com/MyPetOctocat/bachelor\_2022 [↑](#footnote-ref-10)