



**University of  
Zurich**<sup>UZH</sup>

# **Design and Implementation of an Online Marketing Prediction System**

*Severin Wullschleger  
Zürich, Switzerland  
Student ID: 13-715-081*

Supervisor: Eder J. Scheid, Dr. Thomas Bocek  
Date of Submission: September 27, 2021



# Abstract

In online marketing, everything is about the Customer Acquisition Costs (CAC), which indicate how much money has to be spent to acquire a new customer [13]. Especially in the software industry, CAC are high and increasing [2][7]. Therefore, solutions are required to reduce the CAC and keep them low as quickly as possible after a product launch.

In this thesis, it was researched what data can be exported from online marketing ad platforms (*e.g.*, Google Ads) and how it can be connected to the data collected by the promoted mobile application. With this knowledge, the goal was to find out whether and to what extent analyses and predictions regarding the performance of future online mobile app campaigns can be made by using the aggregated data and calculated Key Performance Indicators (KPI) based on the connected data from the different sources.

With the implementation of a prototype, the system operating costs were evaluated and several challenges encountered in implementing such a system were identified. The main challenge is that the export of data from mobile app campaigns is restricted in several ways, and therefore the data volume is too low to train the machine learning models in most cases. The designed prediction system component is affordable in terms of operation costs and therefore worth a try if enough data is available.

Future work could test the system on data from campaigns that promote web applications, as the data extraction capabilities are better for non-app campaigns and the low data volume might be less of an issue.

Im Online-Marketing dreht sich alles um die Kundenakquisitionskosten (Customer Acquisition Costs, CAC), die angeben, wie viel Geld ausgegeben werden muss, um einen neuen Kunden zu akquirieren [13]. Besonders in der Softwarebranche sind die CAC hoch und steigend [2][7]. Daher sind Lösungen erforderlich, um die CAC nach einer Produktlancierung so schnell wie möglich zu reduzieren und danach weiterhin niedrig zu halten.

In dieser Arbeit wurde untersucht, welche Daten aus Online-Marketing-Anzeigeplattformen (z.B. Google Ads) exportiert und wie diese mit den von der beworbenen Mobile-Applikation gesammelten Daten verbunden werden können. Mit diesem Wissen war das Ziel herauszufinden, ob und in welchem Umfang Analysen und Vorhersagen bezüglich der Performance zukünftiger Online-Mobile-App-Kampagnen anhand der aggregierten Daten und berechneten Key Performance Indicators (KPI) auf der Grundlage der verknüpften Daten aus den verschiedenen Quellen möglich sind.

Mit der Implementierung eines Prototyps wurden die Betriebskosten des Systems evaluiert und mehrere Herausforderungen, die sich bei der Implementierung eines solchen Systems ergeben, identifiziert. Die grösste Herausforderung besteht darin, dass der Export von Daten aus Mobile-App-Kampagnen in mehrfacher Hinsicht eingeschränkt ist, sodass die Datenmenge in den meisten Fällen zu gering ist, um die Machine Learning-Modelle zu trainieren. Die entworfene Prediction-Systemkomponente ist in Bezug auf die Betriebskosten erschwinglich und daher einen Versuch wert, wenn genügend Daten vorhanden sind.

Zukünftige Arbeiten könnten das System mit Daten von Kampagnen für Webanwendungen testen, da die Möglichkeiten der Datenextraktion für Nicht-App-Kampagnen besser sind und das geringe Datenvolumen weniger ein Problem darstellen könnte.

# Acknowledgments

The work during the last months has been very exciting. I learned a lot during this time and I was able, once again, to deep dive into my Master's degree major, Data Science, which was a lot of fun. I would definitely like to take this opportunity to thank a few people for their support during my master's thesis and throughout my studies at the University of Zurich.

First and foremost, I would like to thank my supervisors Eder and Tom for the great work and support during the thesis and also during the previous projects I was allowed to do with you guys. You always helped me perfectly when it was needed. Many thanks for that!

Furthermore, I would like to thank Prof. Dr. Burkhard Stiller and the whole Communication Systems Group Lab for the great cooperation during all these years and the different work and modules I did with you. I hope we can stay in touch in the future, be it through future work of interns, master projects, which I am very happy to be involved with Axelra, or just by having a coffee at the university.

Another gratitude goes to Axelra and Freya, who gave me the opportunity to write a master thesis on the Freya use-case and its marketing campaigns.

Last but not least, I would like to thank my girlfriend and my family who always supported me during the master thesis and throughout my entire studies.

*Severin Wullschleger, 13-715-081*



# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Description of Work . . . . .	2
1.2 Overall Goal and Research Questions . . . . .	3
1.3 Thesis Outline . . . . .	3
<b>2 Background and Related Work</b>	<b>5</b>
2.1 Theoretical Background . . . . .	5
2.1.1 Customer Acquisition Costs (CAC) . . . . .	5
2.2 Cloud-based Advertisement Platforms . . . . .	6
2.2.1 Google Ads . . . . .	6
2.2.2 Facebook Ads . . . . .	8
2.2.3 Display & Video 360 . . . . .	9
2.2.4 Apple Search Ads . . . . .	9
2.3 Retrievable Metrics and Fields . . . . .	10
2.3.1 Metrics Retrievable from Google Ads and Facebook Ads . . . . .	10
2.3.2 Google Analytics and its Retrievable Attributes . . . . .	16
2.4 Similar Platforms . . . . .	25
2.5 ML and AI Algorithms for Predictions . . . . .	28
2.5.1 Linear Regression . . . . .	28

2.5.2	Decision Trees . . . . .	29
2.5.3	Deep Neural Network (DNN) . . . . .	29
2.6	Related Work . . . . .	30
2.6.1	Build an Intelligent Online Marketing System: An Overview . . . .	31
2.6.2	Predictive Modeling of Campaigns to Quantify Performance in Fashion Retail Industry . . . . .	32
2.6.3	Development of Autonomous Intelligent System for Google Ads . .	34
2.6.4	Research Insights . . . . .	36
<b>3</b>	<b>Design</b>	<b>37</b>
3.1	Architecture Overview . . . . .	37
3.2	Application and Data Warehouse Architecture . . . . .	39
3.2.1	Google Cloud Data Warehouse . . . . .	39
3.2.2	Self-hosted Data Warehouse . . . . .	43
3.3	Relevant Metrics and Attributes . . . . .	45
3.3.1	Metrics and Attributes for Business Intelligence . . . . .	45
3.3.2	Feature and Target Selection for the Prediction System . . . . .	48
3.4	Prediction System Architecture . . . . .	50
3.4.1	With a Google Cloud DWH and Vertex AI for Model Training and Serving . . . . .	51
3.4.2	With a Self-hosted DWH and Self-hosted Model Training and Serving	52
3.4.3	With a Self-hosted DWH, Vertex AI Training and Self-hosted Model Serving . . . . .	52
3.4.4	(Continues) Model Learning . . . . .	53
3.5	UI Features . . . . .	55



<b>4</b>	<b>Implementation</b>	<b>57</b>
4.1	Implemented Architecture Overview . . . . .	57
4.2	Event Logging . . . . .	59
4.2.1	Data Tracking within the App . . . . .	59
4.2.2	Firebase Integration and Logging to Google Analytics . . . . .	61
4.3	Data Warehouse . . . . .	62
4.3.1	Application Data Export . . . . .	62
4.3.2	Google Analytics Data Export . . . . .	65
4.3.3	Google Ads Data Export . . . . .	67
4.3.4	Importing the Data into DWH Stage 1 . . . . .	70
4.3.5	Transforming the Data to Stage 2 . . . . .	71
4.3.6	Stage 3: Connecting the Data from Different Sources . . . . .	74
4.3.7	Exporting the Data . . . . .	77
4.4	Prediction System . . . . .	78
4.4.1	Model and Training Code Definition . . . . .	78
4.4.2	Synthetic Data / Oversampling the Data . . . . .	82
4.4.3	Model Training . . . . .	83
4.4.4	Model Serving . . . . .	85
4.5	Prediction Front-End . . . . .	87
4.6	BI Tool . . . . .	91
<b>5</b>	<b>Evaluation &amp; Discussion</b>	<b>93</b>
5.1	Results of ML . . . . .	93
5.1.1	Regression Models . . . . .	93
5.1.2	Decision Forests Models . . . . .	97
5.2	Cost Analysis . . . . .	99
5.3	Challenges . . . . .	101
5.3.1	Complexity of the Online Advertisement Universe and the Restrictions on the Data Access . . . . .	103

5.3.2	Linking Data on Ad Level for Mobile App Campaigns . . . . .	103
5.3.3	Audience Restrictions for Mobile App Campaigns . . . . .	104
5.3.4	Data Volume & Synthesized Data . . . . .	105
5.3.5	Conclusion . . . . .	105
<b>6</b>	<b>Summary, Conclusion, and Future Work</b>	<b>107</b>
	<b>Bibliography</b>	<b>109</b>
	<b>Abbreviations</b>	<b>115</b>
	<b>List of Figures</b>	<b>116</b>
	<b>List of Tables</b>	<b>118</b>
<b>A</b>	<b>Installation Guidelines</b>	<b>121</b>
A.1	Repository Structure . . . . .	121
A.2	Running the System on the Local Machine . . . . .	123
A.3	Running the Prediction Component on Vertex AI . . . . .	126
<b>B</b>	<b>Contents of the CD</b>	<b>129</b>

# Chapter 1

## Introduction

In online marketing, everything revolves around the so-called Customer Acquisition Costs (CAC). The CAC indicates how much money has to be spent to acquire a new customer [13]. It should be noted that a customer is not considered to have been acquired when they registered on a platform, but only when they also reached or transacted the defined goal of a platform. In this sense, it is irrelevant whether this was achieved by concluding a subscription or ordering a product or service. Furthermore, in almost every business, a customer is expected to achieve the platform goal more than once, *e.g.*, by repeatedly ordering a service or purchasing a monthly or yearly subscription.

With increasing digitization, fully digital companies are becoming more complex in terms of both technology and content. As a result, the processes to successfully on-board a customer become more challenging and lengthy for both the system and the customer [9]. Examples include Finance Tech (FinTech) applications that require legally binding user identification to enable the user to open a bank account or portfolio [19]. Longer and more complex on-boarding processes increase the likelihood that a potential customer will cancel the on-boarding process and not become a customer. This, in turn, increases CAC and the desire for advertising systems that make so-called re-engagement possible.

In the meantime, there are numerous communication channels and platforms that deal with the placement of online advertising and the tracking of its results. Platforms that accompany the user in the on-boarding process and assist with emails and push notifications are also widespread. Examples of such platforms are Facebook Ads [15], Google Ads [51], Display & Video 360 (DV360) [44], Adjust.com [85], Customer.io [11] or Maatoo.io [87]. For mobile applications, there is also the possibility of placing ads in the App Store [3] or Google Play Store [31]. Besides the fully digital communication channels, there are still traditional marketing channels, such as radio and TV marketing or billboard advertising.

However, these platforms and their analytics are limited to the data they can collect themselves and few are able to handle internal application data of the advertised platform. In addition, each advertising platform only evaluates the performance of the advertisements placed with it due to lack of inter-communication between platforms and their intrinsic competition. This does not benefit the user who contracted the service, as it does not have a unified view of the performance of all their campaigns. Thus, a platform that evaluates

the performance across all advertisement networks, tracks the entire process from the first impression to the defined end-target, and enables analysis of user- and platform-specific data is required to address such issues. Moreover, predictions of how a planned campaign will perform or suggestions to the user of which network campaigns should be started on are not possible with existing platforms. Thus, the research on the design of a platform that combines metrics from different platforms to provide an unified view of their performances with the combination of Machine Learning (ML) and Artificial Intelligence (AI) to provide such predictions is an interesting research topic.

## 1.1 Description of Work

The main focus of this thesis is the research of a solution that connects application-internal data (*e.g.*, a data warehouse) with cloud-based campaigns and advertising platforms to provide predictions and insights concerning their performance on customer acquisition. In this sense, it contains a research aspect, with the survey of available cloud-based campaign and advertising platforms, with the listing of metrics that can be collected from them and the selection of relevant metrics. In addition, it contains the research on ML and AI algorithms that can be applied to predict the campaign effectiveness towards acquiring customers. Besides the theoretical research aspect, the work includes a practical part with the prototypical design and implementation of such a solution, and its evaluation to assess feasibility and prediction accuracy.

To address the research aspect of this thesis, the literature regarding online advertising platforms, the economics of CAC, and ML and AI techniques were studied. Further research concerning related work on these topics and knowledge of the technical aspects were acquired. Questions such as a how to retrieve the metrics provided by the platforms, and which are the available ML libraries and frameworks to be integrated in the solution, were tackled. An overview of existing solutions that address the thesis topic was elaborated.

The practical part of this work includes the design and implementation of a solution that maps the entire process from a potential customer's first point of contact, through registration and on-boarding on the target platform, to app usage while collecting data from the process. The stored data will enable the solution to perform Business Intelligence (BI) and analytics, based on ML on the combination of application data linked to the campaigns data (*i.e.*, collected metrics). The designed prediction system should allow users to enter planned campaigns as parameters (*e.g.*, network, budget, and target groups) and should calculate predictions on how the campaign will perform (*e.g.*, number of registrations, number of final target actions). Further, the solution should allow the inverse path, providing suggestions based on desired outcomes of a campaign (*e.g.*, target groups, number of registrations, number of end target actions) as to where and to what extent campaigns should be placed. The ML and AI models are expected to continue to learn as the data increases. As there exist several advertising platforms, the solution should be modular so that campaign and advertising tracking software as well as the data model of the application(s) can be changed with little effort.

The final stage of this thesis covers a comparative evaluation and discussion by looking at the ML results, the cost of such a system and the challenges which need to be tackled as well as drawing a conclusion and propose some future work.

## 1.2 Overall Goal and Research Questions

The overall goal of this thesis is to do the required research and to acquire the knowledge to develop a design of an online marketing prediction system for mobile app campaigns and to implement a prototype of this system. This prototype should be evaluated to answer the following Research Questions (RQ):

- RQ1** To what extent can advertising campaign results of mobile application campaigns, which go beyond the normal advertising platform data horizon, be predicted when the advertising platform data is linked to in-app data?
- RQ2** To what extent are the predicted result useful and meaningful, and if not, what are the prerequisites and requirements for such a system to become useful in real-life?

## 1.3 Thesis Outline

The thesis is structured as follows: Chapter 2 provides background information and outlines related work, describing popular cloud-based advertisement platforms and identifying the metrics and fields that can be retrieved from these platforms. Chapter 2 compares similar platforms and gives a brief introduction about relevant ML algorithms for the predictions is given. Chapter 3 describes the design of the whole system. First, an architectural overview is provided before the design of each component is discussed in detail. Chapter 4 details the implementation of the system prototype. Again an architectural overview introduces the chapter before each component of the prototype is described and explained in detail. The evaluation and discussion of the prototype can be found in Chapter 5, where the ML results, the results of a performance and cost analysis and the challenges of such a system are outlined. Finally, Chapter 6 concludes the thesis and lists future work.



# Chapter 2

## Background and Related Work

This chapter starts with introducing background information about customer acquisition costs. It then discusses campaign platforms and metrics available in such platforms. It continues with a description and comparison of similar platforms and a brief introduction on ML algorithms that can be used for predictions. Finally, related work to this thesis is addressed.

### 2.1 Theoretical Background

#### 2.1.1 Customer Acquisition Costs (CAC)

In online marketing and sales, the CAC indicates how much money must be spent to acquire a new customer. The CAC is calculated by dividing the total money spend on sales and marketing by the amount of customers [13]. Especially for self-service-like online platforms, where potential customers do not have contact to an employee of the company, and where the user on-boarding is a multiple step process (*e.g.*, opening a bank account via a mobile app), registered users are likely to cancel or stop the on-boarding process in the middle of it. These users are usually not considered customers since they are not fully on-boarded and they do not use the app or have not ordered the product. Also, more money needs to be spent to bring some of them back to the platform and turn them into paying customers.

The height of the CAC depends to a large extent on the industry of the application [2], the application type and of course the product itself, but also on seasonality and other time-related components, which makes it a dynamic metric. Compared to other industries, the software industry faces high [2] and significantly increasing CAC [7].

## 2.2 Cloud-based Advertisement Platforms

This section presents the documentation on the research on available cloud-based advertisement platforms and how the data can be retrieved. The platforms Google Ads [51], Facebook Ads [15], Display & Video 360 [44] and Apple Ads Search are described. The research was conducted on the whole online advertisement ecosystem. The most popular and useful platforms for mobile app campaigning are described below.

The research of the available advertisement platforms and retrieval methods showed that it is very much focused on the two big players Google and Facebook. Of course, there are other advertising platform players out there such as Microsoft Ads [71], Twitter Ads [86], LinkedIn Ads [68] and Yelp Ads [89]. For mobile app advertisement there are also solutions such as AdMob [55] or AdColony [1] which allow the advertisers to place the ad in mobile apps. For the sake of the scope of this thesis, it was decided to limit the further research on the two big players Google and Facebook.

### 2.2.1 Google Ads

Google Ads (formerly Google AdWords and Google AdWords Express) is a solution that allows companies to advertise their solutions, services and products online. Advertisements are placed in different Google services (*e.g.*, Google Search, YouTube [65], Google Play Store where Android apps are downloaded [31]) or on private websites whose owners receive a percentage of the revenue for displaying such advertisements. The simplest Google Ads version is self-service based and everything can be done without a lot of knowledge. Users can adjust budget, targets and goals as well as start or stop a campaign within the platform [46].

Google Ads differentiates between three campaign categories [46]:

- *Search campaigns* that are displayed in Google search results;
- *Display campaigns*, which can be seen on websites and in apps; and
- *Video campaigns*, which are usually 6 to 15 seconds long and are embedded before or in YouTube videos.

Google Ads runs the following payment models [46]:

- *Cost Per Click (CPC)* or *Pay Per Click (PPC)*: The advertiser pays only if the ad was clicked.
- *Cost Per Impression (CPI)*: It is paid based on how many impressions (when the ad is shown) the ad has.
- *Cost Per Engagement (CPE)*: Only when a user engages with the ad and completes the engagement, then the advertiser needs to pay. For example, the advertiser only pays for a display video ad if the user watched it until the end.



### Data Retrieval Methods

Google Ads data can be accessed in various ways. In this section the most common methods are described.

- *Google Ads API Reports*: Google Ads provides a powerful Application Programming Interface (API) which can be used to access the Google Ads Platform programmatically [62]. It is possible to pull numerous reports via the reports section of this API. Not only read operations can be done with the API, also the campaigns and ads can be adjusted, stopped or restarted. Google provides different client libraries to access the API and it is not encouraged (only for developing purposes) to use the native HTTP-REST interface [36]. Client libraries are available in Java, .NET, PHP, Python, Ruby and Perl. The data can only be accessed with an access token, which must be requested from Google.
- *AdWords API* [61]: As already mentioned, Google Ads was known before as Google AdWords. The AdWords API is still functioning but will sunset on April 27, 2022. For the sake of completeness and because one can still find a lot of references to the AdWords API, it is listed here. The functionality of the AdWords API is included in the Google Ads API, therefore it is recommended to use and migrate to the Google Ads API.
- The *Big Query Data Transfer Service* [52] provides functionality to transfer Google Ads data to a Big Query data set. Big Query [33] is the database, data warehouse and big data tool of Google Cloud [37]. With Big Query Data Transfer it is possible to transfer all Google Ads data to a data set at once. The transfer can also be scheduled and run using the Google Cloud Console, the Google Cloud command line tool, the REST interface or the Java client library. After the data transfer, the data can be accessed querying the tables and views with normal Structured Query Language (SQL) statements and it can also be exported.
- *Google Ads Scripts* [50] is a browser-based IDE that allows users to programmatically access Google Ads data by using JavaScript. Pulling or accessing reports as well as adjust ad and campaign parameters is possible.
- The *Google Ads User Interface (UI) Console* allows users to manually export the displayed data to a CSV file (or various other file formats).
- When using the *Google Analytics (GA)* [28] platform, account managers can decide to link the Google Ads account to their Google Analytics account. By doing so, GA automatically connects some of the campaign data from Google Ads with the click data from GA. The connected data is then accessible through the GA UI, can be manually downloaded to a CSV file (or various other file formats) or, if GA is connected to Big Query too, it is automatically stored in a Big Query data set where it can be accessed, queried and exported.
- *Google Data Studio* is a BI tool from Google. It allows the user to connect to several different data resources [42]. While data imports from Google products

are supported from scratch, other data can be imported too, by using third-party platform connectors. Data Studio is focused on visualization and BI and not on data extraction and processing.

### 2.2.2 Facebook Ads

Facebook Ads is the advertisement platform of Facebook. It is part of the Facebook for Business section. Like Google, Facebook owns more than one platform to place the ads on. When using Facebook Ads, Facebook places the ads on the Facebook platform, on Instagram and in the Facebook Messenger. The Facebook Ads platform is also self-serviced and can be used with relatively few knowledge. All campaigns can be edited and stopped whenever the user wants [15].

Facebook offers the different ad placement variations listed below [18].

- *Feed-Ads*: The ad is placed in the Facebook News feed, Instagram feed, Facebook Marketplace, Facebook Video feed, in the Instagram Explore section and in the Facebook Messenger inbox.
- *Stories-Ads*: The ad is placed in Facebook, Instagram and Messenger Stories. Stories are the posts, which can only be seen for 24 hours.
- *In-Stream Ads*: The ad is shown in Facebook video streams, Instagram TV videos and in Instagram Reels.
- *Search Ads*: The ad can be found in the Facebook Search results.
- *Messaging Ads*: It is possible to sponsor messages that will appear in the Facebook Messenger for people the advertiser is already in contact with.
- *In-Articles*: The ad appears in Instant Articles in the Facebook Mobile App.
- *App-Ads*: The ad appears in third-party apps either in banners or as so called Rewarded Videos (if the ad is watched, the user gets a reward).

### Data Retrieval Methods

For accessing the data in Facebook Ads the best solution is to use the official Facebook for Business tools.

The official user interface for accessing and managing the Facebook Ads is the Facebook Ads Manager [14]. It provides all functionality to manage the campaigns and ads. To see detailed statistics and reports one has to use the Facebook Ads Report tool. There, reports can be generated, columns can be added and removed and reports can be manually exported as an Excel or CSV file or as a PNG image.

Besides the user interface, Facebook provides the Marketing API, which offers various endpoints to manage the Facebook campaigns. The reporting part of this API is called

Ads Insights [17]. The API uses the Graph API, which is the way to read and write from and to the Facebook social graph [16]. The Graph API from Facebook can also be called directly instead of going through the Ads Insights API.

In addition to the official solutions, there is also numerous campaign and ad management software available. Some are pricey and powerful and some cheaper or for free but with less features. The data from Facebook can also be imported into Google Data Studio. A third-party connector needs to be used, but then it is possible to do that in an automated way.

### 2.2.3 Display & Video 360

Display & Video 360 belongs to the Google Marketing Platform. The Google Marketing Platform has different products at hand for companies of different sizes. For small companies they provide Google Analytics, Google Tag Manager, Google Optimize, Google Surveys and Google Ads [48]. For bigger companies they offer Google Analytics 360, Google Data Studio, Google Optimize 360, Google Search Ads 360, Google Surveys 360, Google Tag Manager 360, the Google Campaign Manager 360 and, the core of it, Display & Video 360 [45].

Similar as in Google Ads, in DV360 the advertiser can set up ad campaigns for the Google Search, for YouTube, for the Google Play Store or for normal website where the ad can be seen in banners around or between the content. Compared to Google Ads, DV360 is less automated since there is more variation of the provided ad formats. Whereas Google Ads includes Image, Expanded Text and Responsive Ads with a lot of limitations, DV360 gives the advertiser more room for creativity. DV360 is also more branding focused than Google Ads [20].

A DV360 API exists, but it is only meant to manage campaigns but not reading report data [49]. For extracting reports either a user interface can be used, which only allows for manually exporting into CSV-files or automatically sending it to a Gmail address. For exporting reports from the 360 ecosystem in a programmatic way, a DV360 Service called Data Transfer v2.0 can be used. It transfers the data to CSV files and saves it in the Google Cloud Storage [43].

Another solution is also Google Data Studio. Without much effort, DV360 data imports can be scheduled to run every day, for example. As said, data processing or extraction from Data Studio is not possible in an automated way.

### 2.2.4 Apple Search Ads

Apple Search Ads provides the possibility to promote iOS apps in the App Store from Apple. There are two options to advertise the app, either by choosing the basic or the advanced option. The advanced option the advertiser can place the ads to the search tab of the app store or at the top of the search results. Customer groups can be specified

autonomously and the campaigns and the strategy can be customized. Whereas with the basic option, everything is automated, only the budget can be specified [4].

For the basic option, data retrieval works via dashboard with the possibility to manually export report data. The advanced Search Ads option includes APIs to manage campaigns, analyze the campaign and ad statistics as well as export these analyses [4].

## 2.3 Retrievable Metrics and Fields

This section gives an overview of what metrics and fields can be extracted from the Google ecosystem products as well as from Facebook. The analytics in both ecosystem are setup very similar and therefore the metrics are very well comparable. The above listed data retrieval methods are used to extract these metrics and fields.

First, the possible metrics from Google Ads and Facebook Ads are listed and compared. The second subsection describes Google Analytics and its role, and explains the metrics and attributes that can be gathered from it.

### 2.3.1 Metrics Retrievable from Google Ads and Facebook Ads

The most important and most interesting metrics retrievable from both the Google and Facebook ecosystem are listed and explained below. If there is a different term for the same metric in Google than in Facebook or a metric is only available in one of the ecosystems, the terms are marked with  $G$  or  $F$ .

**Core performance metrics:** These metrics are gathered by the platforms and are used for the calculated performance metrics:

- Impressions: The number of times an ad was shown.
- Reach ( $F$ ): The number of people who saw the ad.
- Clicks: The number of times an ad was clicked on.
- Unique Clicks ( $F$ ): The number of different people who clicked on the ad.
- Outbound Clicks ( $F$ ): The number of clicks that led to properties not owned by Facebook.
- Views ( $G$ ): The number of times a video ad was viewed. As already mentioned, at Google, a view is only counted if the video is watched until the end.
- Interactions ( $G$ ) or Results ( $F$ ): The number of times a user interacted with the ad. An interaction is a click for a text ad and a view for a video ad. Facebook generalizes it as Results and it is possible to specify in the settings what action a Result should be.

- Cost ( $G$ ) or Amount Spent ( $F$ ): The costs that caused an ad.
- Gross Impressions ( $F$ ): Number of impressions plus invalid impressions from non-human traffic.
- Invalid Clicks ( $G$ ): The number of clicks that are tagged as invalid by Google.
- Invalid Interactions ( $G$ ): The number of interactions that are tagged as invalid by Google.
- Impressions Absolute Top Percentage ( $G$ ): The percentage of impressions coming from ads displayed as the very top ad above the search result.
- Impressions Top Percentage ( $G$ ): The percentage of impressions coming from ads displayed above the search result.

**Calculated performance metrics:** The core performance metrics can be used to calculate metrics. The following metrics are common but the list is obviously not complete:

- Frequency ( $F$ ): The average number of times an ad was seen from one person (Impressions / Reach).
- Click-through Rate (CTR): The portion of impressions that led to clicks (Clicks / Impressions).
- Unique CTR ( $F$ ): The percentage of people who saw the ad and clicked on it (Unique Clicks / Reach).
- View Rate: The ratio between views and impressions (Views / Impressions).
- Interaction Rate ( $G$ ) or Results rate ( $F$ ): The ratio between interactions and impressions (Interactions / Impressions).
- Average Cost per Click (CPC): The average costs of one click (Cost / Clicks).
- Average Cost per Unique Click ( $F$ ): The average costs of a unique click (Cost / Unique Clicks).
- Average Cost ( $G$ ) or Cost per Result ( $F$ ): The average amount paid per interaction (Cost / Interactions).
- Cost per 1000 People Reached ( $F$ ): The cost for reaching 1000 people.
- Cost per 1000 Impressions (CPM) ( $F$ ): The cost for reaching 1000 impressions.
- Invalid Click Rate ( $G$ ): The ratio between invalid clicks and total amount of clicks.
- Invalid Interaction Rate ( $G$ ): The ratio between invalid interactions and total amount of interactions.

**Core conversion metrics:** These metrics are measured by the platforms and are used for the calculated conversion metrics:

- **Conversions:** The number of conversions that happened after interaction with the ad. The actions resp. events that count as a conversion can be specified in the settings. The bidding algorithms optimize towards this value.
- **Conversion Value ( $G$ ):** If a value for the conversion is set, this is the sum of the values. For example, if buying something in a web shop is set as conversion goal, the purchase amount could be the conversion value.
- **Installs:** For mobile app campaigns and advertisements it is possible to count the number of installs.
- **In-App Actions:** The number of in-app conversions that an ad led to. The conversion actions can be defined in the settings. For iOS applications, conversions cannot be tracked directly neither in Google nor in Facebook.
- **Orders, Average cart size, Average order value, Cost of goods sold:** These metrics are in beta testing at Google. The goal is that shopping orders and their order values can be tracked and optimized against these metrics.

**Calculated conversion metrics:** The core conversion metrics can be used together with the performance metrics to calculate metrics such as the following ones:

- **Cost per Conversion:** The amount the advertiser paid for one conversion on average.
- **Conversion Rate:** How often an ad interaction led to a conversion on average. (Conversions / Interactions)
- **Value per Conversion:** How much a conversion is worth on average (Conversion value / Conversions).
- **Conversion Value per Cost:** The conversion value is set in relation to the costs of the ad. (Conversion value / Cost)
- **Conversion Value per Click:** The conversion value is set in relation to the number of clicks of the ad. (Conversion value / Clicks)
- **Cost per Install:** Same as with other conversions, the cost of an app installation can be calculated (Cost / Installs).
- **Cost per In-App Action:** Also the cost per in-app actions can be calculated (Cost / In-app actions).
- **The order metrics in Google beta testing** can be used to calculate Gross Profit or Revenue metrics.
- **Average Target Cost per Action (CPA) ( $G$ ):** The target CPA is the cost per action the bid system is optimizing for. It can be set in the campaign or ad configurations. The average target CPA is calculated over the selected time period.

- Average Target Cost per Install (CPI) ( $G$ ): The target cost per install can be set too, such that the bidding algorithm is optimizing for this value. The average target CPI is calculated over the selected time period. Note that for Cost per Impression and Cost per Install often the same abbreviations are used.
- Average Target Cost per In-App Action ( $G$ ): The same can be done with the cost per in-app action. The average target cost per in-app action is also calculated over the selected time period.

The metrics can also be divided into different segments such as time, click type, conversion action or category, device types, network type and so on. In addition to the metrics, normal attribute fields are retrievable. For example, campaign attributes (*e.g.*, ID, name, campaign type, campaign start and end date) or ad attributes (*e.g.*, ID, name, type, format). For Google Ads, the following campaign parameter fields can be retrieved:

- Campaign Type
- Campaign Sub Type
- Campaign Name
- Campaign Status (*e.g.*, Paused or Running)
- Start Date
- End Date
- Bidding Strategy Type (*e.g.*, optimized towards installs or towards an action)
- Campaign Budget
- Target Cost per Install or Target Cost per Action

In addition to the normal campaign characteristics, campaign audience criterion can be defined and retrieved as well. For example at Google Ads, it is possible to restrict the campaign audience regarding Gender, Age, Income Cluster, Parental Status, Language, Location the potential customer is based in and Location the potential customer is interested in.

### Accessing and Exporting Raw Click Data

Besides figuring out which metrics can be extracted from the platform, it was also researched if and how the raw data from the advertisement platforms can be accessed and exported. The goal is to avoid reports with metrics like number of impressions, number of clicks or number of views segmented in different ways, but to have a data record of a single impression or a single click and to have all the meta data of it (*e.g.*, which ad was clicked and at what time).

It turned out that the above mentioned metrics should always be retrieved in an aggregated way. Every retrieval method is designed to export aggregated statistics on different levels. The possibilities to export statistics about your campaign, ad groups or ads with different segments or different attributes connected to it are very diverse. The data analyst has almost no limitations. For example, the Google Ads API (see the first bullet point in Section 2.2.1) provides a list of reports which can be exported [62]. The list is divided into Resources with metrics and Resources without metrics. It seems that almost every database table can be read. For example, it is possible to retrieve the click and impression data segmented by gender or to export the amount of clicks of a specified campaign on a specified day.

No possibility of exporting raw impression data was found. Probably due to data size reasons, this is not available neither at Google nor at Facebook. There is only one report available in the Google Ads API that kept alive the hope of having a single data record of a click. Unfortunately, no such endpoint was found on the Facebook side. The Click View report provides metrics aggregated at each click level, it is said in the documentation. For data size reasons, it can only be queried with a filter of one day. Also, probably for the same reason, the data can only be queried for the last 90 days [35].

Table 2.1 shows the specification of the Click View Report of the Google Ads API. One can see that the metrics Clicks is the only metric which can be retrieved, and the metric can be segmented with a few attributes. The meta data that can be gathered is seen in the *Resource field* column. The `ad_group_ad` field is a so called resource name and it references an ad group ad. The ad group ad meta data can be read from another report (the Ad Group Ad Report [26]). By connecting the two reports, it is possible to figure out which ad was clicked on. If the click belongs to a search channel ad, the keyword field is filled with the referred keyword.

However, an important attribute in this report is the `gclid`. `gclid` stands for Google Click ID and is the identifier of each click. Based on the research conducted, it was found out that the `gclid` can also be found in the event logs of Google Analytics. With these event logs, it is possible to send events and, more importantly, user properties from an application (no matter if mobile or web application) to Google Analytics. The occurrence of the Google Click ID in the GA events as well as in the Google Ads API Click View Report builds the basis of the opportunity to connect application data with the Google Ads campaign and click data. Google Analytics and its data retrieving opportunities are outlined in Section 2.3.2 as well as applied in the Design chapter (Chapter 3).

The Google Ads reports are not only retrievable by the Google Ads API but also by the Big Query Data Transfer Service that was mentioned as data retrieval method in Section 2.2.1 (third bullet point). The Big Query Data Transfer Service creates a table and a view in a data set of Big Query for each exportable report. The reports that are exported are still based on the AdWords API instead of the newer Google Ads API. Therefore, the reports are slightly different. They are listed in the AdWords API documentation [61]. The Click View Report displayed in Table 2.1 is based on the former Click Performance Report from the AdWords API. The two reports are very similar and the important Google Click ID can be found also in the older one. In summary, this means that it is



Table 2.1: Click View Report from the Google Ads API

Resource fields	Segments	Metrics
ad_group_ad	ad_network_type	clicks
area_of_interest.city	click_type	
area_of_interest.country	date	
area_of_interest.metro	device	
area_of_interest.most_specific	month_of_year	
area_of_interest.region	slot	
campaign_location_target		
gclid		
keyword		
keyword_info.match_type		
keyword_info.text		
location_of_presence.city		
location_of_presence.country		
location_of_presence.metro		
location_of_presence.most_specific		
location_of_presence.region		
page_number		
resource_name		
user_list		

possible to retrieve the click data (including the Google Click ID) not only by the Google Ads API but also with the Big Query Data Transfer Service.

### Retrievable Metrics and Fields for App Campaigns

What was not found out during the research because it is not documented anywhere, but nevertheless belongs in this chapter even though it was not found out until deep into the implementation phase, is the following: The mobile app campaigns of Google Ads differ greatly from normal search and website ad campaigns when it comes to the reporting, especially when it is about the click data being reported in the Click View or Click Performance report. With mobile app campaigns it is not possible to use one of the reports, it was replied by several Google Support employees (see Section 5.3).

Consequently, advertisers of mobile app campaigns are not able to extract single click data records including the Google Click ID and therefore it is not possible to figure out which app user clicked on which advertisement before installing and starting the app. In addition to that (also something that was discovered during prototyping), the campaign audience criterion that are retrievable for key word search campaigns differ from the ones retrievable by mobile app campaigns. Although it is possible to restrict the mobile app campaigns to that audience, the data cannot be gathered in a suitable way. For mobile app campaigns, it is only possible to pull the location and the language criterion information.

The next section describes what still can be found out with the help of Google Analytics.

### 2.3.2 Google Analytics and its Retrievable Attributes

Google Analytics (GA) is not an advertisement platform (and therefore not listed in Section 2.2), but it is useful for tracking and retrieving data. Hence, introduced in this section. Same as for the advertisement platforms, first a short GA introduction is given, then data retrieving methods are described and lastly the retrievable data is presented.

GA is a powerful tool from Google to track and analyze the data and traffic of a company. Company owners can analyze for website visitors, application events, how long visitors stay on their website or platform, and which visitors return for a second app visit [28].

For websites and web applications, GA provides a so called global site tag (the `gtag.js`), a JavaScript tagging framework that enables the sending of event data to the measurement products [27]. Mobile apps are preferably using the Firebase Software Development Kits (SDK) to measure user interactions [29][30]. Firebase is the app framework from Google and one part of it is Google Analytics. Other parts are authentication, hosting, cloud storage services, push-notification services or Crashlytics (which measures and logs app crashes). Everything from Firebase is designed for Android and iOS mobile apps [47].

A third type of possible connections to the GA measurement products is the Measurement Protocol, which allows basically every internet connected device to send event logs to GA [54]. It was tried to log back-end events from the server directly to GA through the Measurement Protocol. The tests were not successful, which proves that the message on the documentation website that *This is an alpha API* is there for a reason.

GA gathers several data automatically while it is still possible to log customized events from within the application. GA gathers enough data (*e.g.*, device ID, IP address, application version) from the Firebase SDK or the global site tag, to tell how many unique users used the app, if it was a so called *first open* event or if the user was a recurring one. The Firebase SDK or the global site tag allows to not only send logs from the app to GA, it also includes the feature to send internal user properties to GA. By sending the internal user ID to GA, it was later possible to associate and connect the data on the application server with the one from GA.

Figure 2.1 shows GA's browser-based application where data can be filtered and analyzed. From there, it is possible to download visualization as images or report data in Excel or CSV format as well as doing further reports in Google's BI tool Google Data Studio. The Figure shows the Acquisition page, which displays the number of new users and from which network and campaigns these users came.

Besides using the UI, data is also accessible through an API. The API can be used with different client libraries written in Java, Python, PHP and JavaScript [57]. In addition, as mentioned earlier, GA can be connected with Google Big Query, resulting in an automatic export of all event data from GA to a Big Query data set. The data is saved in a partitioned table, meaning for every day a new table is created with the name in the format `events_YYYYMMDD` [34].

Table 2.2 is from the GA help section and shows the columns which are exported to the Big Query data set. The fields are divided into 11 categories: App, Device, Stream and Platform, User, Campaign, Geo, Event, Ecommerce, Items, Web and Privacy info.

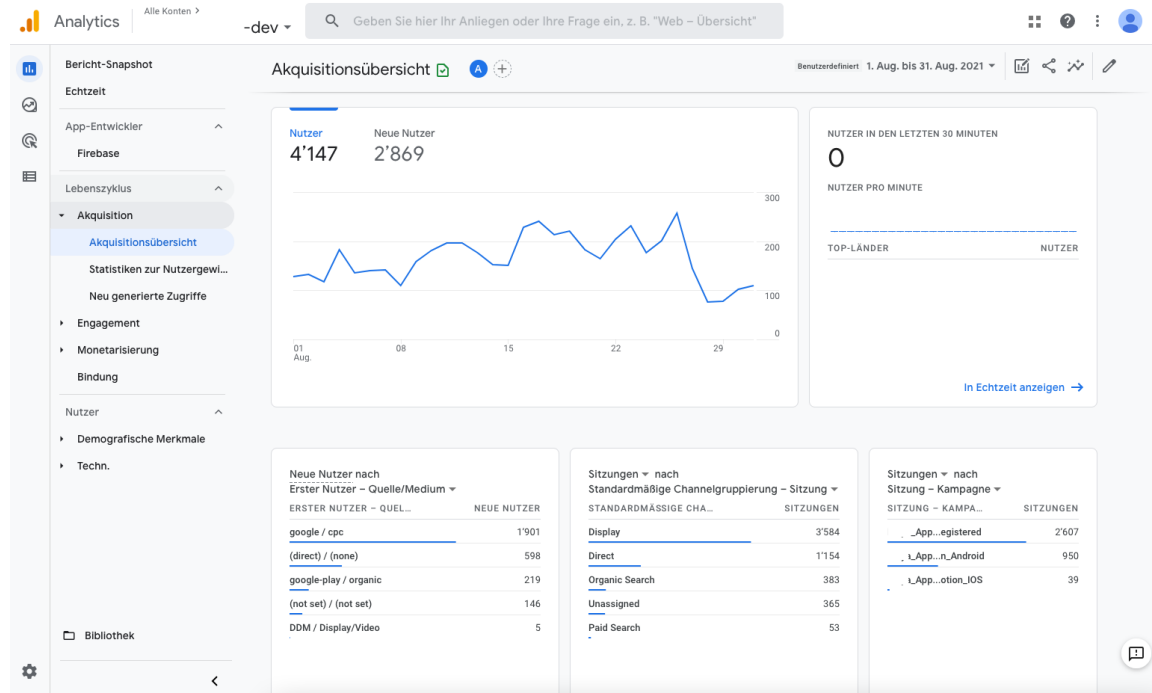


Figure 2.1: Google Analytics' User Interface

The category Device contains all device related data: For example, GA tracks which operating system was used (`device.operating_system`). As it can be seen in the category User, GA distributes a `user_pseudo_id` which is an identifier for the end user, and the `user_id` can be read which is set via the `setUserId` interface. The category Geo contains location specific data that is derived from the IP address of the user.

The category Event is the core of the data set and it is always filled: Every row of the data set contains a date, a timestamp, an event name and usually some event parameters. The `event_name` field contains `first_open` for example, when the user installed and opened the app the first time, or it can contain customized events logged from the app (e.g., `support_window_opened`), when a user was looking around in the support section. The `event_params` array-field in the Event category can contain various data about the event. Usually, there are more than one event parameter per event log. For example, the Google Click ID is also transferred in an event parameter with the `event_params.key` set to `gclid` and the `event_params.value.string_value` set to the ID value.

The last category, which is very worth mentioning, is the Campaign category. If GA is linked to the Google Ads account, the Campaign category contains information regarding the origin of the traffic to the application. If a user clicks on an advertisement of a Google Ads campaign (named *Example Campaign*) that directs the user to Apple's App Store where the app is downloaded and installed, then, when the user opens the app, the Firebase SDK logs an event which contains the marketing campaign name (*Example Campaign*) in `traffic_source.name` and the name of the network (Google Ads) in `traffic_source.source`. As stated in Table 2.2, only the first traffic source information is saved. If the user interacts with other networks or campaigns after that, the data in these fields does not change [34].

## Origin Tracking with Urchin Tracking Module (UTM) Parameters

Google products are automatically tracked by GA. For activating the automatic tracking, the advertiser only needs to activate the so called *Auto-tagging* in the Google Ads resp. DV360 settings. Auto-tagging automatically extends the ad with an additional parameter, namely the Google Click ID (GCLID) [32]. This allows GA to tell which ad the user was coming from. With the GCLID, GA can link the event to an ad and therefore the advertiser can find the campaign name in the `traffic_source.name` field.

For Facebook and other third-party advertisement platforms, on the other hand, UTM parameters need to be set. UTM parameters are used to add campaign information or references to destination URLs of the ad campaign. Google Analytics, other tracking platforms or a self-made tracking system can use the URL query parameters to read the information about the source of the traffic [40].

GA provides a Campaign URL Builder to implement the correct destination URLs with correct UTM parameters [8]. There are five standardized UTM parameters that can be added to the URL and are included by GA in the event log record [40]:

- `utm_source`: the advertiser sending traffic (*e.g.*, google or facebook)
- `utm_medium`: the marketing medium (*e.g.*, cpc, banner or newsletter)
- `utm_campaign`: the campaign name
- `utm_term`: identifies paid search keywords
- `utm_content`: can be used to distinguish different content in the same campaign or ad. (*e.g.*, two different newsletter links)

The use of UTM parameters to identify traffic source is only possible for ads that specify a destination URL. If there are campaigns and ad types where no URL can be specified, then obviously no UTM parameters can be set and therefore no traffic source can be transferred. Mobile app campaigns are this kind of campaigns where no destination URL can be set. For both Google Ads and Facebook Ads, instead of a destination URL, the advertiser sets the iOS App bundle ID for an app published in Apple's App Store or, in case of an Android app published in the Google Play Store, the app's package name. This is the reason why for mobile app promotions, it is currently not possible to obtain click event data associated to an end-user and associated to a Facebook Ads campaign at the same time.

While for Google Ads and DV360 campaigns, GA automatically tracks the source, the medium and the campaign name (see Section 2.3.2) and it is retrievable via the `traffic_source.name`, the `traffic_source.medium` and the `traffic_source.source` fields, it is, as with Facebook Ads, not possible to specify customized UTM parameters, because there is no destination URL to specify. The inability to customize the destination URL for app campaigns, combined with the fact that there is no report with raw click data for mobile app campaigns (see Section 2.3.1), leads to the finding that for mobile app

campaigns, the deepest level of information that can be linked to the end-user is that of the campaign data (*e.g.*, campaign name or campaign costs). Any deeper level, such as which ad from the campaign was clicked, cannot be associated with the end-user. This finding applies to Google Ads and DV360, while for Facebook Ads and other third-party platform not even campaign data is linkable currently. Since the environments of these advertisement platforms are very fast evolving, there is hope for the future, that customizing a destination URL will also become doable not only for website or web app campaigns but also for mobile app campaigns.

Table 2.2: Google Ads Export Data [34]

Field name	Data type	Description
<b>App</b>		
app_info	RECORD	A record of information on the app.
app_info.id	STRING	The package name or bundle ID of the app.
app_info.firebase_app_id	STRING	The Firebase App ID associated with the app
app_info.install_source	STRING	The store that installed the app.
app_info.version	STRING	The app's versionName (Android) or short bundle version.
<b>Device</b>		
device	RECORD	A record of device information.
device.category	STRING	The device category (mobile, tablet, desktop).
device.mobile_brand_name	STRING	The device brand name.
device.mobile_model_name	STRING	The device model name.
device.mobile_marketing_name	STRING	The device marketing name.
device.mobile_os_hardware_model	STRING	The device model information retrieved directly from the operating system.
device.operating_system	STRING	The operating system of the device.
device.operating_system_version	STRING	The OS version.
device.vendor_id	STRING	IDFV (present only if IDFA is not collected).
device.advertising_id	STRING	Advertising ID/IDFA.
device.language	STRING	The OS language.
device.time_zone_offset_seconds	INTEGER	The offset from GMT in seconds.
device.is_limited_ad_tracking	BOOLEAN	The device's Limit Ad Tracking setting. On iOS14+, returns false if the IDFA is non-zero.
<b>Stream and platform</b>		
stream_id	STRING	The numeric ID of the stream.
platform	STRING	The platform on which the app was built.
<b>User</b>		
user_first_touch_timestamp	INTEGER	The time (in microseconds) at which the user first opened the app or visited the site.
user_id	STRING	The user ID set via the setUserId API.
user_pseudo_id	STRING	The pseudonymous id (e.g., app instance ID) for the user.
user_properties	RECORD	A repeated record of user properties set with the setUserProperty API.
user_properties.key	STRING	The name of the user property.
user_properties.value	RECORD	A record for the user property value.
user_properties.value.string_value	STRING	The string value of the user property.

Table 2.2: Google Ads Export Data [34]

Field name	Data type	Description
user_properties.value.int_value	INTEGER	The integer value of the user property.
user_properties.value.double_value	FLOAT	The double value of the user property.
user_properties.value.float_value	FLOAT	This field is currently unused.
user_properties.value.set_timestamp_micros	INTEGER	The time (in microseconds) at which the user property was last set.
user_ltv	RECORD	A record of Lifetime Value information about the user. This field is not populated in intraday tables.
user_ltv.revenue	FLOAT	The Lifetime Value (revenue) of the user. This field is not populated in intraday tables.
user_ltv.currency	STRING	The Lifetime Value (currency) of the user. This field is not populated in intraday tables.
<b>Campaign</b>		Note: traffic_source attribution is based on cross-channel last click traffic_source values do not change if the user interacts with subsequent campaigns after installation
traffic_source	RECORD	Name of the traffic source that first acquired the user. This field is not populated in intraday tables.
traffic_source.name	STRING	Name of the marketing campaign that first acquired the user. This field is not populated in intraday tables.
traffic_source.medium	STRING	Name of the medium (paid search, organic search, email, etc.) that first acquired the user. This field is not populated in intraday tables.
traffic_source.source	STRING	Name of the network that first acquired the user. This field is not populated in intraday tables.
<b>Geo</b>		
geo	RECORD	A record of the user's geographic information.
geo.continent	STRING	The continent from which events were reported, based on IP address.
geo.sub_continent	STRING	The subcontinent from which events were reported, based on IP address.
geo.country	STRING	The country from which events were reported, based on IP address.
geo.region	STRING	The region from which events were reported, based on IP address.
geo.metro	STRING	The metro from which events were reported, based on IP address.
geo.city	STRING	The city from which events were reported, based on IP address.
<b>Event</b>		
event_date	STRING	The date on which the event was logged (YYYYMMDD format in the registered timezone of your app).
event_timestamp	INTEGER	The time (in microseconds, UTC) at which the event was logged on the client.

Table 2.2: Google Ads Export Data [34]

Field name	Data type	Description
event_previous_timestamp	INTEGER	The time (in microseconds, UTC) at which the event was previously logged on the client.
event_name	STRING	The name of the event.
event_params	RECORD	A repeated record of the parameters associated with this event.
event_params.key	STRING	The event parameter's key.
event_params.value	RECORD	A record of the event parameter's value.
event_params.value.string_value	STRING	The string value of the event parameter.
event_params.value.int_value	INTEGER	The integer value of the event parameter.
event_params.value.double_value	FLOAT	The double value of the event parameter.
event_params.value.float_value	FLOAT	The float value of the event parameter. This field is currently unused.
event_value_in_usd	FLOAT	The currency-converted value (in USD) of the event's "value" parameter.
event_bundle_sequence_id	INTEGER	The sequential ID of the bundle in which these events were uploaded.
event_server_timestamp_offset	INTEGER	Timestamp offset between collection time and upload time in micros.
<b>Ecommerce</b>		
ecommerce	RECORD	A record of information about ecommerce.
ecommerce.total_item_quantity	INTEGER	Total number of items in this event, which is the sum of items.quantity.
ecommerce.purchase_revenue_in_usd	FLOAT	Purchase revenue of this event, represented in USD with standard unit. Populated for purchase event only.
ecommerce.purchase_revenue	FLOAT	Purchase revenue of this event, represented in local currency with standard unit. Populated for purchase event only.
ecommerce.refund_value_in_usd	FLOAT	The amount of refund in this event, represented in USD with standard unit. Populated for refund event only.
ecommerce.refund_value	FLOAT	The amount of refund in this event, represented in local currency with standard unit. Populated for refund event only.
ecommerce.shipping_value_in_usd	FLOAT	The shipping cost in this event, represented in USD with standard unit.
ecommerce.shipping_value	FLOAT	The shipping cost in this event, represented in local currency.
ecommerce.tax_value_in_usd	FLOAT	The tax value in this event, represented in USD with standard unit.
ecommerce.tax_value	FLOAT	The tax value in this event, represented in local currency with standard unit.
ecommerce.transaction_id	STRING	The transaction ID of the ecommerce transaction.
ecommerce.unique_items	INTEGER	The number of unique items in this event, based on item_id, item_name, and item_brand.
<b>Items</b>		
items	RECORD	A repeated record of items included in this event.



Table 2.2: Google Ads Export Data [34]

Field name	Data type	Description
items.item_id	STRING	The ID of the item.
items.item_name	STRING	The name of the item.
items.item_brand	STRING	The brand of the item.
items.item_variant	STRING	The variant of the item.
items.item_category	STRING	The category of the item.
items.item_category2	STRING	The sub category of the item.
items.item_category3	STRING	The sub category of the item.
items.item_category4	STRING	The sub category of the item.
items.item_category5	STRING	The sub category of the item.
items.price_in_usd	FLOAT	The price of the item, in USD with standard unit.
items.price	FLOAT	The price of the item in local currency.
items.quantity	INTEGER	The quantity of the item.
items.item_revenue_in_usd	FLOAT	The revenue of this item, calculated as price_in_usd * quantity. It is populated for purchase events only, in USD with standard unit.
items.item_revenue	FLOAT	The revenue of this item, calculated as price * quantity. It is populated for purchase events only, in local currency with standard unit.
items.item_refund_in_usd	FLOAT	The refund value of this item, calculated as price_in_usd * quantity. It is populated for refund events only, in USD with standard unit.
items.item_refund	FLOAT	The refund value of this item, calculated as price * quantity. It is populated for refund events only, in local currency with standard unit.
items.coupon	STRING	Coupon code applied to this item.
items.affiliation	STRING	A product affiliation to designate a supplying company or brick and mortar store location.
items.location_id	STRING	The location associated with the item.
items.item_list_id	STRING	The ID of the list in which the item was presented to the user.
items.item_list_name	STRING	The name of the list in which the item was presented to the user.
Items.item_list_index	STRING	The position of the item in a list.
items.promotion_id	STRING	The ID of a product promotion.
items.promotion_name	STRING	The name of a product promotion.
items.creative_name	STRING	The name of a creative used in a promotional spot.
items.creative_slot	STRING	The name of a creative slot.
<b>Web</b>		
web_info	RECORD	A record of information for web data.

Table 2.2: Google Ads Export Data [34]

Field name	Data type	Description
web_info.hostname	STRING	The hostname associated with the logged event.
web_info.browser	STRING	The browser in which the user viewed content.
web_info.browser_version	STRING	The version of the browser in which the user viewed content.
<b>Privacy info</b>		
privacy_info.ads_storage	STRING	Whether ad targeting is enabled for a user. Possible values: Yes, No, Unset
privacy_info.analytics_storage	STRING	Whether Analytics storage is enabled for the user. Possible values: Yes, No, Unset
privacy_info.uses_transient_token	STRING	Whether a web user has denied Analytics storage and the developer has enabled measurement without cookies based on transient tokens in server data. Possible values: Yes, No, Unset

## 2.4 Similar Platforms

Table 2.3 presents an overview and a comparison of platforms similar to the one designed in this thesis.

There are numerous applications which focus on gathering all the information from social media applications and ad platforms, as well as from analytics platforms (*e.g.*, Google Analytics) and, in addition to that, from Customer Relation Management (CRM) platforms (*e.g.*, Salesforce [79]). These platforms (*e.g.*, Easy Insights [69], Singular [82], Measured [70]) are business intelligence tools with good integration capabilities. Usually, all the data is stored in the cloud and the platforms are focusing on data visualization and beautiful reporting.

A second category of applications (*e.g.*, Nexoya [78] or Datorama from Salesforce [12]) extend the features of the previous category by not only providing BI features but also calculating predictions about growth numbers and other results. For example, Nexoya predicts the growth of different key performance indicators (KPI), the number of followers or the number of page likes. They also offer the calculation of growth potential based on their predicted optimizations (*e.g.*, change of budget allocation), anomaly and correlation detection. The predictions are usually based on the aggregated data and KPIs and not on raw click and user data since the data is not available at that granularity level on these platforms.

Another similar platform worth mentioning is Adjust [85]. Adjust focuses on data collection and measurement and can integrate various advertising platforms (*e.g.*, Facebook, TikTok, Google Ads) and can also measure newsletter performances, for example. It makes use of its own Adjust SDK which needs to be integrated in the front-end of the application. With this tracking framework, Adjust can measure and collect a whole different level of data granularity. As Google Analytics, Adjust collects raw events and click data and makes it accessible to the platform user either by sending the data directly via callback to a data warehouse or by providing an export feature. While the tracking and collecting features are very strong, Adjust does not offer any prediction or optimization functionality.

Last but not least, also Google Analytics needs to be mentioned in this section. Although it has already been described as a tool to retrieve Google Ads or DV360 data, as well as to retrieve raw app event and click data, it is also a similar platform to the one developed in this thesis. GA is the leader in tracking, collecting and measuring usage data and also provides BI functionalities to analyze, visualize and report the data. Like Adjust, GA makes use of a tracking tool which needs to be integrated into the front-end code (global site tag or Firebase SDK) to gain access to a detailed data level and therefore, GA is able to store raw event data and make it accessible to the GA user. On the other hand, no prediction or optimization features are offered.

The last column of the Table 2.3 includes the classification of this thesis' design against the discussed characteristics of the similar platforms mentioned in the preceding columns: The design presented in this paper focuses on collecting raw click data and combining it with user sensitive application data because this is the only way to make predictions

related to user and target groups in combination with their traffic sources. Compared to the other platforms, the focus is less on normalizing and visualizing the platform data but more on using the linked data towards target group predictions. In addition, the problem is tackled that application data does not necessarily want to be kept on external servers, but should at least be hosted in a private cloud. This is because, as seen in the Data storage row of table 2.3, the platforms mentioned all use cloud storage and are therefore not considered by every customer (*e.g.*, banks).

Table 2.3: Similar Platforms Comparison

	<b>Easy Insights</b>	<b>Nexoya</b>	<b>Adjust</b>	<b>Google Analytics</b>	<b>Main thesis goal</b>
<i>Ad platform integrations</i>	Yes, <i>e.g.</i> , Twitter, Apple	Yes, <i>e.g.</i> , Facebook, Twitter, Mailchimp, Linkedin	Yes, <i>e.g.</i> , Facebook, TikTok, Newsletter, every platform which uses ad destination URLs	Yes, Google products and every platform which uses ad destination URLs	Mobile app capmpaign platforms, especially Google Ads, DV360 and Facebook
<i>Analytics platform integrations</i>	Yes, <i>e.g.</i> , Google Analytics, Firebase, Adjust	Yes, <i>e.g.</i> , Google Analytics	No	No	No
<i>CRM platform integrations</i>	Yes, <i>e.g.</i> , Salesforce, Zapier	Yes, <i>e.g.</i> , Salesforce	No	No	No
<i>Data tracking and aggregation</i>	Yes	Yes	Yes	Yes	Yes
<i>Data normalization, standardization and visualization</i>	Yes	Yes	Yes	Yes	Yes (No priority)
<i>Access to raw click and user data</i>	No	No	Yes, via instant callback and as export	Yes, accessible directly in Google BigQuery	Yes
<i>Predictions based on aggregated data</i>	No	Yes, predicts <i>e.g.</i> , growth rate, number of followers, number page likes	No	No	No
<i>Predictions based on raw click and user data</i>	No	No	No	No	Yes
<i>Optimization</i>	No	Yes, shows potential and optimization possibilities ( <i>e.g.</i> , in budget allocation)	No	No	No, only model improvement
<i>Data storage</i>	Cloud	Cloud	Cloud	Cloud	On-premise in private cloud or encrypted at Google Cloud
<i>Other features</i>	Interactive reports, Beautiful visualizations	Anomaly detection, Correlation detection	Audience Builder, Mobile app install and uninstall tracking with its own SDK, customized events	Real-time report, Traffic source detection	App user data only in private cloud

## 2.5 ML and AI Algorithms for Predictions

In this sections, ML algorithms for making predictions are outlined. Machine learning systems can be categorized in supervised, unsupervised, semi- supervised and reinforcement learning. The following list quickly explains the main idea of each category. This section is based on the book “Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems” [21].

- In *Supervised Learning*, the algorithm is fed with labeled training data and optimizes the model toward this label, so that a new data set without label can be entered and the model returns with a predicted label. Supervised learning can be categorised in classification problems, where the label is a class (categorical), and regression problems, where the label is numerical. Some of the most important supervised learning algorithms are k-Nearest Neighbors, Linear Regression, Logistic Regression, Support Vector Machines (SVM), Decision Trees (DT) and Random Forests. Neural networks can also be supervised.
- In *Unsupervised Learning* the algorithm input is data without labels. The algorithm does not learn towards a specific item but can categorize or simplify data or can find rules based on the input data. The most important unsupervised learning types are clustering, dimensionality reduction and association rule learning.
- *Semi-supervised Learning* algorithms are usually combinations of the two above. For example, some labeled data as input is used for doing predictions based on the combination with a clustering algorithm.
- *Reinforcement Learning* works differently. Many robots implement it. The learning system (called agent), observes the environment, performs an action and gets either rewarded or penalized for good resp. bad actions. The algorithm learns over time which strategy results in the most reward.

In this thesis, supervised learning algorithms are used, since the goal is to do campaign outcome predictions (new label) based on the outcome of previous campaigns (labeled input data). The outcome of a marketing campaign is measured by the number of additional users or by an amount which reflects the purchase amount, the investment amount or something similar (see Section 3.3.2). Therefore, the used ML algorithms need to be capable of solving regression problems. In the following subsections some of the ML algorithms for regression problems are described.

### 2.5.1 Linear Regression

Linear Regression is one of the simplest algorithm in supervised machine learning [21]. Linear regression models are linear functions of the input features. Basically, the model predicts a value based on a weighted sum of all the input features plus a so-called *bias term* or *intercept term* [21]. Known from basic mathematics, adding a constant to a

function does not change the shape of the function but only its position. Because the linear regression algorithm calculates a sum of features, it can only deal with numerical features. Categorical features need to be transformed into a binary feature for each category. This process is called *one-hot-encoding* [21]. Another solution to transform categorical features into numerical ones, is to give every category a number. However, this is suggested only if the categories are ordinal. Otherwise, the ML algorithm interprets, for example, the categories 1 and 2 as more similar than the categories 0 and 3.

### 2.5.2 Decision Trees

Decision Trees can perform classification as well as regression problems. DT are very powerful and it is also possible to perform multi-output predictions [21]. They work by splitting the input data set in two subsets. This is being done recursively until the defined maximum depth of the tree (defined as a hyper-parameter) is reached. The splitting of the data set is done by defining a condition with a feature and a threshold (*e.g.*, *campaign duration > four month*). The feature-threshold-pair that generates the most balanced subsets is selected for the next decision [21].

The Random Forest algorithm is one of the most popular representatives of DT algorithms and it is also an example for ensemble learning strategies. *Ensemble Learning* describes the strategy of training many models instead of one. To obtain predictions, the input features are passed to all trained models. The final prediction is then derived by aggregating the predictions from all models [21]. The simplest examples for an ensemble learning algorithm would be to have three individual models (no matter what kind of model as long as the output format is the same), each predicting the asked value. The ensemble learning algorithm then aggregates the three values (for regression) or decides for the majority of predicted classes (classification). In many cases, the decision from the ensemble learning algorithm is more accurate than the one from the best algorithm [21]. The Random Forest algorithm uses the same training algorithm on randomly selected subsets of the training set and trains multiple trees (*e.g.*, 500 trees). During the splitting of the data sets, Random Forest does not search for the very best feature among all features but for the best feature in a randomly chosen subset of features. This improves the tree diversity and results generally in a better model overall [21].

Decision Tree models such as the Random Forest algorithm can be boosted. *Boosting* describes the technique of training the models sequentially and that each model is trying to correct its previously trained predictor [21]. An example for a boosted DT model is the Gradient Boosted Random Forest algorithm.

### 2.5.3 Deep Neural Network (DNN)

The definition of a deep neural network is an Artificial Neural Networks (ANN) with more than one hidden layer [24]. A linear regression model, for example, has no hidden layer. It has two layers, one is the input layer which includes the input features and the bias term (see above), and the second is the single output layer which includes the label or the

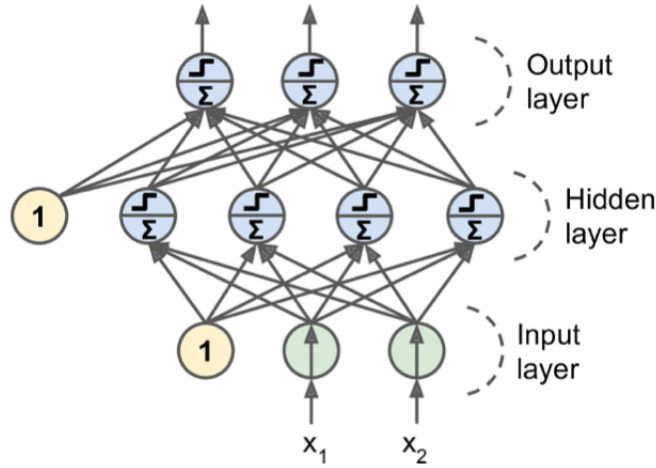


Figure 2.2: A simple ANN: The Multi-Layer Perceptron

prediction [21]. As mentioned, each input feature is weighted and the best weights are searched for the given outputs. In DNNs, on the other hand, there are additional layers that are neither the input nor the output layer. Each layer, besides the output layer, contains a number of so called perceptrons (neurons) and a constant (the bias term)[24]. Each neuron is fully connected to each neuron of its previous and next layer and each input of a neuron is weighted [24]. Figure 2.2 shows a simple ANN with one hidden layer. This ANN is called Multi-Layer Perceptron (MLP). Such a network can be trained by using the backpropagation training algorithm which functions as followed: For each training instance, the backpropagation algorithm first makes a prediction (forward pass). With the prediction the error is measured. Then, the algorithm runs through each layer in the reverse direction to measure what contribution each connection makes to the error (backpropagation). Finally, the connection weights are adjusted slightly to reduce the error (gradient descent step) [24]. This can be repeated as often as it is specified, while one of these repetitions is called *an epoch*.

Compared to the linear regression algorithm, deep neural networks can deal with nonlinear dependencies while linear regression is limited to linearities [21]. This means, if changes of the input features do not affect the output in a direct proportion, there might be a better choice than a linear regression model.

## 2.6 Related Work

The following subsections present related work in the field of online marketing systems, prediction systems and intelligent self-learning systems.

All three papers contain parts that are helpful and interesting as a foundation and/or comparison to this thesis. It should be noted that the number of related approaches is low due to novelty of the solution proposed in this master thesis.



### 2.6.1 Build an Intelligent Online Marketing System: An Overview

Cui et al. (2019) [10] designed and built an intelligent online marketing system. They see challenges when it comes to the enormous amount of data and clicks that can be collected. Collecting the data is not the problem, but analyzing it. They also see difficulties because the conversion of a user can take a long time. While the collected data (*e.g.*, ad clicks) is very big, the data from conversion events (*e.g.*, user purchases the product) can be relatively small. A third point that makes it difficult is the fact that users usually not only have one touch point with the marketing of the same product, which makes the conversion tracking even more complicated.

They claim their system to be the “first publicly available architecture of an intelligent online marketing system” which was developed and put to production. The goal of their work was the automation of ads creation, automatically updating ads and their bids and the budget allocated to each ad. In addition to that, reporting on ad performances and visualizing it were further objectives.

[10] identifies “data logging, multitouch attribution, lifetime value (LTV) modeling, bidding and budget optimization, ad and campaign management, keyword expansion, and an experimentation and reporting framework” as crucial parts of such a system. Data tracking as the base part of the system is crucial but there exists no consensus which metrics are useful. On the other hand, every possible metric should be tracked and kept to allow future analysis. Business intelligence can eventually use every possible data for supporting strategic decisions. The system performs internal data tracking with a messaging framework based on Kafka [67]. The external data is tracked with third party APIs. If possible, the system collects the same data from internal as well as external data sources and cross-validates the data from different sources and performs abnormalities detection on it.

The problem of how a campaign can be optimized towards first party user data, such as payment amounts, is also addressed by the proposal of a front-end and a back-end. The front-end approach uses Google Tag Manager (GTM) [53] although it is raised that a front-end approach may not be suitable for tracking first party user data. The back-end approach is to extract the conversion which was highly likely saved in a database and manually load it to the third party campaign tool.

Multi-touch Attribution, which describes the fact that a converting user has contact to more than one ad and crediting only the first or the last one the user saw or clicked on is not necessarily fair, is tackled by User-Defined Functions (UDF) that run rule-based and model-based methods to overcome or at least minimize that problem. Figure 2.3 shows the multi-touch attribution visualization of [10].

Moreover, [10] points out the need of a LTV of a conversion. When user convert, their conversion have varying LTVs, meaning they bring different amount of revenues to the advertiser. They built two models, one estimating the LTV assuming the conversion event already happened and the other estimating the likely hood of the conversion event really happening. The final result is the product of both models. With the resulting models, the value of each ad can be estimated and, in combination with strategic decisions, the bidding

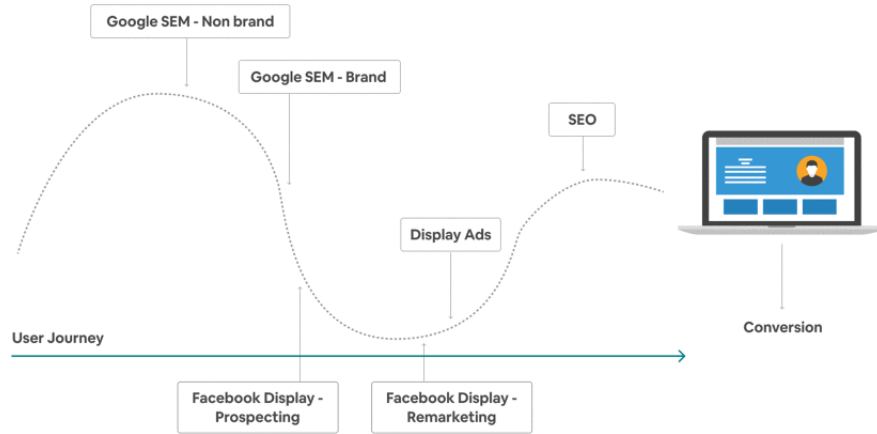


Figure 2.3: Multi-touch Attribution [10]

and budget optimization can be performed. It is also interesting that they performed the LTV calculations with a conversion time-frame of one year.

Other parts of the overview contain the automation of the ads and campaign management, automated keyword expansion, an experimentation framework to perform A/B testing automatically and the need for an nearly real-time reporting system. [10] conclude that a marketing system as outlined can “reduce operational cost, increase operational efficiency, optimize ROI [Return On Investment (ROI)], and improve customer engagement.” They also point out that their solution is not yet the best way to build a marketing system. They expect, with raising amount of click data being collected, to see experiments in leveraging deep learning for bidding and LTV modeling and Natural Language Processing (NLP) suggesting ad content and new keywords.

### 2.6.2 Predictive Modeling of Campaigns to Quantify Performance in Fashion Retail Industry

Giri et al. (2019) [22] recognized that promotional campaigns are often very expensive and the revenue from it is not very high. Difficulties in identifying the correct factors which drive customers attention are widespread, especially in the fashion retail industry where their experiment was conducted. The goal of the work was to develop data-driven predictive analytics to identify the success rate and profitability of campaigns. By modelling the behaviour of past campaigns, the goal was to identify the campaign’s key parameters. The goal was to come up with two different predictions: A regression model that calculates the average profit of the campaign and a classification model that classifies the overall performance of the campaign into success or failure.

[22] used 826 campaigns and the data from it to define a feature catalogue with 28 features, which needed to be collected and calculated for every campaign in the data preparation phase. Figure 2.4 lists the 28 campaign attributes [22] used for the models.

As shown in Figure 2.4, the attributes can be divided into six groups: campaign type attributes, discount attributes, add-on attributes, requirement attributes, the gross demand

No.	Swedish	Description
1	check	True when campaign type was check
2	firstLine	True when campaign type was first line
3	firstAndSecondLine	True when campaign type was first and second line
4	combDiscount	True when campaign type was combination
5	allOrder	True when campaign type was all order
6	ladder	True when campaign type was ladder
7	threeForTwo	True when campaign type was three for two
8	discCheck	The percent discount when payed by internal check. Can only occur together with check campaigns
9	discFirst	The percent discount on the first item. Firstline, firstandsecondline, combDiscount, allorder and ladder must have a value for DisFirst
10	discRest	The percent discount on the second item. CombDiscount, and ladder must have a value for DiscRest. Firstandsecondline can have a DisSecond.
11	marketingDiscount	The average amount discount received per order (in SEK) for the entire campaign
12	freeGift	True when offered a free gift
13	payedGift	True when offered a payed gift
14	freeShipping	True when offered free shipping
15	freeExressShipping	True when offered free express shipping
16	freeReturn	True when offered free return
17	reqSale	True when the campaign required sale items
18	reqReducedPrice	True when the campaign required items with reduced price
19	reqBrandSelection	True when the campaign required items from specific brands
20	reqValue	True when the campaign required a minimum value
21	req#Items	True when the campaign required a minimum number of items
22	reqTime	True when the campaign required to be used within a limited time
23	reqOrdinaryPrice	True when the campaign required items with ordinary prices
24	reqRedOrdPrice	True when the campaign required items with reduced or ordinary prices
25	grossDemand	The average demand created for a campaign
26	NumRecipients	The number of recipients exposed to a campaign
27	NumOrders	The number of orders resulting from a campaign
28	profit	The average profit of the order, the target value for the regression modelling

Figure 2.4: The 28 Campaign Data Attributes as Model Features [22]

attribute and the campaign performance attributes. The date for the attributes were carefully collected and prepared to fit in the 28 features. For example, the overall campaign performance was calculated and clustered into *Unsuccessful*, *Successful* and *Highly Successful* by calculating the activation value and combining it with the profit of the campaign. The activation value was defined by *Total number of orders received* divided by *Total number of recipients*. For example, a high activation plus a high profit was classified as *Highly Successful*. Figure 2.5 shows the clustering of the campaign performances.

The 25 features were used for both the prediction of the profit as well as the prediction of the success of the campaign. Regression trees and random forest were used for the profit modeling and classification trees and random forest for the success modeling. For evaluating of the models, the standard 10-fold cross-validation was used.

The profit predictions resulted “quite accurate, on average” [22]. The square Root of the Mean Squared Errors (RMSE) and the Mean Absolute Error (MAE) metrics were used to evaluate that. In addition, the R-Squared was calculated to see how well the model fits the data. General insights could be gathered from the structure of the regression tree of this task.

For the success predictions, the two models were compared in terms of Classification Accuracy (CA) and the Area Under the Curve (AUC). This measures the ability of a classifier to differentiate classes. The random forest model shows a slightly better performance than the classification trees. To display the results of the classifier predictions, confusion matrices were created, as shown in Figure 2.6, and the Receiver Operating Characteristic

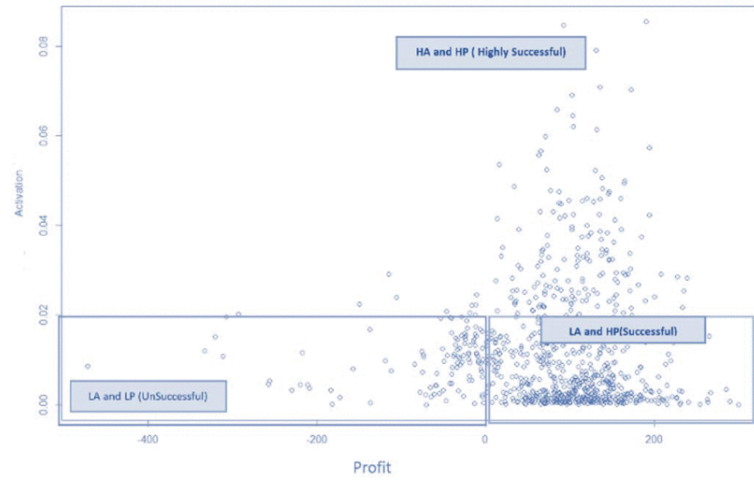


Figure 2.5: Clustering of the Campaign Performances [22]

Classification Tree					
		Predicted			$\Sigma$
		Highly Successful	Successful	Unsuccessful	
Actual	Highly Successful	74	90	1	165
	Successful	51	466	29	546
	Unsuccessful	5	29	81	115
$\Sigma$		130	585	111	826

Random Forest					
		Predicted			$\Sigma$
		Highly Successful	Successful	Unsuccessful	
Actual	Highly Successful	87	78	0	165
	Successful	49	476	21	546
	Unsuccessful	4	23	88	115
$\Sigma$		140	577	109	826

Figure 2.6: Confusion Matrices of the Classifier Predictions [22]

(ROC) curves were plotted (Figure 2.7). Additionally, insights were gathered by looking into the decision rules of the classification tree.

### 2.6.3 Development of Autonomous Intelligent System for Google Ads

Pak, Mocan, Yoldas an Baz (2018) [74] developed an “Autonomous Intelligent System for Google Ads” to reduce the cumbersome work of manually adjusting and optimizing the campaign parameters in Google Ads [51], which is not efficient and error prone due to human interaction.

A first version of the system included five sub-modules, namely automatic account detection which were the cause of errors in the system, automatic positive and negative keyword conflict detection and removal, URL error detection and budget and bidding optimization. The second version brings four additional sub-modules which extend the autonomous intelligent system: optimization of the ad text, automatic optimization of the ROI metric, optimization towards conversion and towards profit.

The automatic optimizations are performed as follows: [74] developed the sub-modules independent from each other, by developing each of them as a transfer function. In the end,

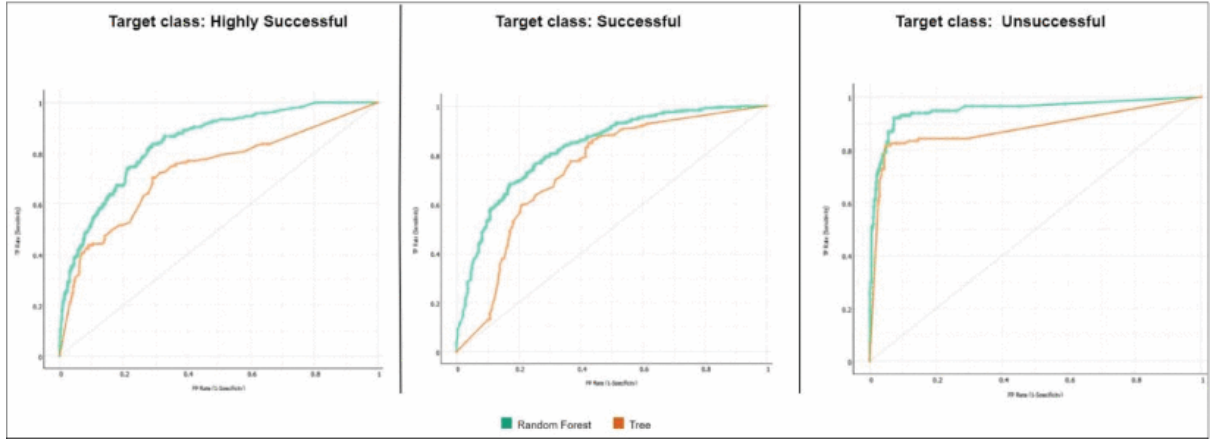


Figure 2.7: ROC Curves for the Classifier Predictions

the designed system is an optimization problem where every function tries to optimize its designated parameters. The advertisement data (such as conversion rate and other metrics as well as campaign parameters) is collected and adjusted by the Google Ads API [62] and by Google Ads Scripts [50]. Google Ads Scripts is a browser-based development environment and can be used as programming language. It facilitates the automation of Google Ads tasks. [74] used Google Ads Scripts to develop the sub-modules.

The optimization tasks are all done with algorithmic logic. The algorithms of all sub-modules are relying on some defined threshold values and if the extracted and/or calculated metric is higher, then an adjustment is made and if it is lower then the adjustment is made in the other direction. The module logic of the sub-modules can be described in pseudo code. The following code is the pseudo-code of the ROI optimization sub-module described by [74]. The other sub-modules work in a similar way.

```
{ if loss impression share (budget) >= 33%
  if Return on Investment > 0
    then increase maximum cost per click (max cpc) and report
    else decrease max cpc and report
  else (do nothing) }
```

The conclusions from the work of [74] are that the system is saving a lot of time and reduces the amount of errors since the campaigns are not managed manually anymore. More than 15,000 campaigns were managed by this system's first version already. Two modules of the second version were successfully tested at the time of writing and two modules were in the alpha test phase.

The challenges of such a system is not the automation of the data extraction or the adjustment of the Google Ads parameters. The most difficult part is to have reliable and sufficient performance data to do the measurements and calculate the metrics, they say. Interesting is also there future work section, where they state that it would be a good idea to investigate different machine learning algorithms for doing the optimizations and comparing them with the existing system.

### 2.6.4 Research Insights

When studying the related work and comparing the found papers with each other and with the goal of this thesis, some learnings and key differences were identified.

While the goal of this thesis is also, similar to [10], to combine external third-party data (Google Ads) with the internal data from the application, the plan and goal was to provide a solution to combine and store the connected data on the internal side. [10] does the whole connection on the external side. Another conclusion from this paper is, that it would have been nice to learn more about the LTV models and to know how the LTV values and the conversions are connected, since this could be integrated in a system as the one designed in this thesis.

Although the use case of [22]’s work is not in the online marketing area, valuable insights about the methodology and approach can still be drawn from it. For example, similar model evaluation and plotting methods could be used. The machine learning problem of this paper is also about marketing campaigns. The goal of the thesis at hand was to implement the predictions on a lower level, but this was not possible due to technical reasons. Besides the predictions of the numerical outcome of the campaign, [22] also generated a classification problem that was also considered for this thesis.

In the third paper from [74], Google Ads data is extracted automatically via the Google Ads API and Google Ads Script, which is also of central relevance for this thesis. The same interfaces are also used to adjust campaign parameters and change campaigns in the other direction. The automatic adjustment and optimization of the campaign parameters of [74], which happens only after the campaign has been started, could be combined with a prediction system like the one designed in this thesis to combine optimizations before and during a campaign.

# Chapter 3

## Design

This chapter describes the design of the system in detail. First, the overall architecture of the design is outlined in Section 3.1. Then, the data warehouse architecture is discussed in Section 3.2, before a short description of the BI metrics is presented in Section 3.3. In the second part of this chapter, the prediction system’s architecture is introduced in Section 3.4. Lastly, the User Interface (UI) functionality is discussed in Section 3.5.

In the following sections describing architectures, their components are highlighted in bold.

### 3.1 Architecture Overview

Figure 3.1 shows a simplified component diagram of the whole system.

There exists a **Mobile Application** (*e.g.*, a cross-platform React Native application for iOS and Android) that is to be promoted with the help of online **Advertisement Platforms** (*e.g.*, Google Ads). The app is published in app stores (*e.g.*, App Store or Google Play Store) and can be downloaded and installed from there.

The online *Marketing Specialist* sets up app campaigns on different advertisement platforms to promote the application. The advertisement platforms display ads to the potential end-user depending on the campaign settings configured by the marketing specialist.

The *End-User* (also referred as *user*) sees one or more ads and eventually clicks on one of them. The end-user is redirected to an app store from where the app is downloaded and installed. The user opens the application, registers and completes an on-boarding process (including *e.g.*, identity verification) before finally using the app.

The mobile app as front-end communicates to an **Application Back-End** where user and application data is stored. This includes logging and storing of events for actions performed on the app. In addition to the internal logging, the front-end also sends events to the **Analytics Platform**. With this event logging, the ad platforms can measure conversions (*e.g.*, number of installs or number of registrations).

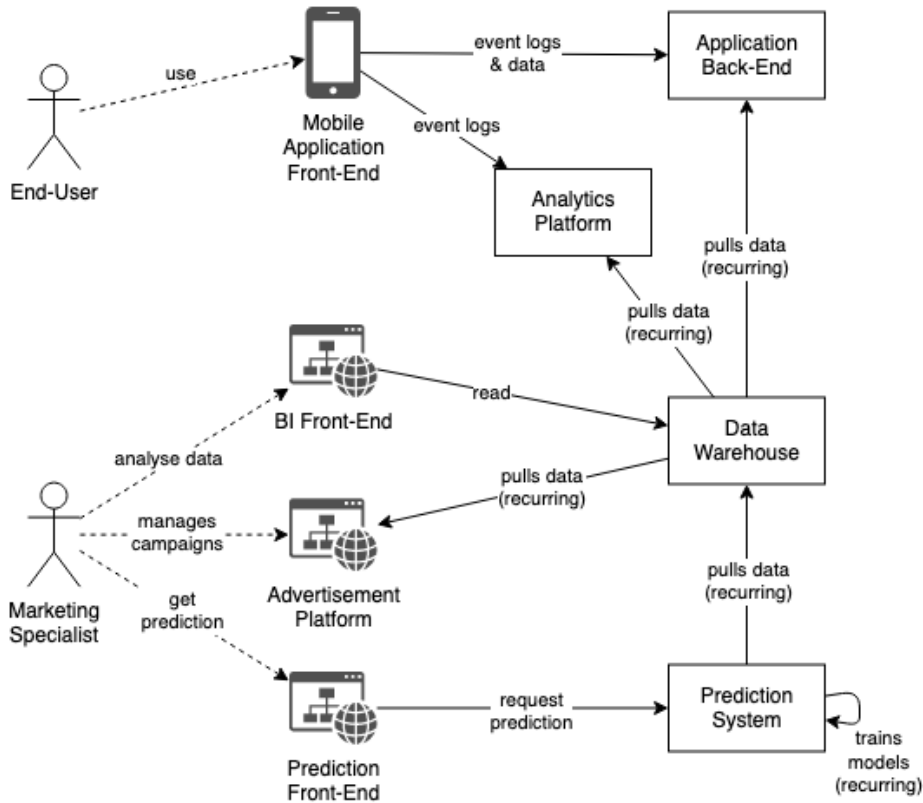


Figure 3.1: Designed Architecture Overview

The application data from the back-end is copied to a **Data Warehouse** instance in a frequent manner (*e.g.*, every night). The same data warehouse collects the data from all the advertisement platforms and the analytics platform.

Within the data warehouse, all the data is processed so that it can be used by the **Prediction System** component. The prediction instance uses the preprocessed data to train the prediction models. In that way, it is assured that the modelling can be repeated in a frequent manner too. The data warehouse does not only process the data for the prediction system, it also prepares different views for the usage in the **BI Front-End** component.

The marketing specialist or other analysts can access the data via the BI component. It is possible to see visualization and reports and to download the raw data for further analytics. In addition, the **Prediction Front-End** component allows the marketing specialist to enter parameters about future campaigns and it will return predictions about the campaign outcome based on the passed parameters and the historical application and campaign data used to train the model. The prediction system also allows the other direction: The marketing specialist enters the desired campaign results and gets suggested campaign parameters.



## 3.2 Application and Data Warehouse Architecture

The first component of the architecture is the data warehouse. It was designed to be used independently from the prediction system part. The only restriction of this component is that the data needs to be exportable so that it can be used as input to the prediction system. By doing the designs independent from each other it was also ensured that the goal of a modular overall system was pursued and achieved.

The system's design considers, in general, two situations. Either the application owner is able to send all the app data and all user information to big cloud providers (*e.g.*, Google Cloud, Amazon Web Services (AWS), Azure), or they cannot allow the data being saved on external servers. The problem with the big cloud providers is that while various data center regions exist (also data centers in Switzerland), most providers cannot guarantee that the data will always and without exception be held in Switzerland. Therefore, institutions that are obliged to store data within Swiss borders (*e.g.*, banks) are often unable to use any of the large cloud providers. If not allowed, usually a private or trusted cloud provider is chosen where the system is hosted, or the whole system is hosted on the company's own servers.

### 3.2.1 Google Cloud Data Warehouse

Figure 3.2 depicts the architecture with a Data WareHouse (DWH) completely hosted in the cloud. The other parts of the system (application back-end the application DataBase (DB)) are preferably hosted in the same cloud, but can basically be hosted everywhere. Google Cloud is the ideal cloud provider for the DWH, because with Google Ads, DV360 and Google Analytics, central advertising and analytics components of the system are already hosted by Google, in the Google Cloud, or at least enable smooth integration. The next paragraphs describe the DWH architecture design in detail.

The React Native **Mobile Application** communicates via HTTPS with two REST APIs hosted in Kubernetes clusters running on Google Cloud's Kubernetes Engine. One API is the one of the Identity Access Management (IAM) server (**IAM Cluster**), which handles the authentication and authorization of the user. So there is the registration endpoint, all the authentication endpoints that work according to OAuth 2.0 specification, as well as identity change endpoints, such as password or email changes. The second API is the Application Server (**Backend Cluster**) API and contains the remaining endpoints that are required for the application usage.

The IAM and the application server are Java Spring Boot applications. To persist data, they have a Java Database Connectivity (JDBC) connection to the **Production DB** in a Cockroach DB cluster (running on Google Cloud's Kubernetes Engine too). As shown in Figure 3.2, the IAM and the back-end are connected to the same database. A schema or table level split was deemed sufficient for this use case. If a third-party IAM would be used or if the data should be clearly separated from each other, a dedicated database would be preferred.



three stages:

- **Stage 1:** Extracted data from different applications (raw and unmodified).
- **Stage 2:** Processed data with application specific calculated fields (KPIs) and attributes.
- **Stage 3:** Data from Stage 2 is connected with each other and prepared such that BI analysts and the ML component can access it.

The data from the Production DB is transferred nightly to the **Big Query** DWH using the following Extract Transform and Load (ETL) process:

1. First, the Production DB data tables are saved via CSV dump to a Google **Cloud Storage** (GCS) bucket. This step is triggered by a Kubernetes *cron job* in the Production DB cockroach cluster.
2. Then the data is loaded into Stage 1 of the DWH. This is done by Big Query's Data Transfer Service, which can access the GCS bucket, reads the CSV files and fills the tables in the DWH data set. The Data Transfer task is configured to run every night. The imported data corresponds to a duplicate of the data in the Production DB.
3. In a further step, the data is transformed with the help of views (*e.g.*, attributes and KPIs are calculated). This forms Stage 2 of the DWH.

The result of this ETL process is that the application data of the Production DB is in tables of the Big Query DWH. The application data was cleaned and KPIs and other attributes were calculated and persisted as well.

The lower part of Figure 3.2 outlines the path of advertisement platform data and other analytics data into the DWH. The design is limited to the assumption that ads are served on **Facebook for Business** (Facebook Ads; including Facebook and Instagram) and on the Google platforms **Google Ads** and **Display & Video 360**. However, if ad platforms are added, they can also be loaded into the data warehouse. The connection is different depending on the platform, but usually it is done by pulling the data via API, converting it into a CSV file and saving the CSV file in a GCS bucket. As long as the data can be queried, any application can be connected to the solution in a modular way.

The ad platforms display ads to the end user. The end user clicks or does not click on the advertisement and thus triggers impressions and click events that can be tracked autonomously by the ad platforms. However, once the app is downloaded and installed, the ad platforms are no longer able to collect further data independently, because the actions take place within the app and outside the ad platform coverage area. For this, two SDKs are integrated in the mobile app. These enable conversion tracking for the advertisement platforms. To measure Facebook Ads conversions, the Facebook SDK is integrated in the mobile app and for Google Ads and DV360 the **Firebase** SDK needs to be integrated. The Firebase SDK collects additional data and sends it to Google Analytics.

The two SDKs send events to Facebook and **Google Analytics** independently, and additionally the sending of customized events can be triggered within the front-end code. As indicated in Figure 3.2, all events from the two SDKs are sent as HTTPS requests. In addition to the customized events, it is important that after a successful registration, the front-end sends the Account ID received from the application server as a user property to Google Analytics and Facebook (via the SDKs). This way, the connection between the GA event data, the Ad platform data and the application data from the Production DB can be established in the DWH. Sending the user properties to GA is triggered in the front-end code.

As explained in Section 2.3.2, the GA data is automatically stored in a Big Query data set by allowing the link between Google Analytics and Big Query. This is required by this design. Also GA and Google Ads as well as GA and DV360 are linked, so that the conversion data is transferred from GA to Google Ads and DV360 and that GA receives campaign information from the ad platforms.

An ETL process pulls the data from each of the ad platforms to the DWH:

- To extract the Facebook Ads data, a scheduled Google Cloud Function (GCF) is set up to run nightly. The GCF calls the Facebook Ads Insights API via HTTPS and converts the response into a CSV file which is stored in a GCS bucket. From the GCS bucket, it can in turn be read by the DWH using the Big Query Data Transfer service and populate the tables in the DWH. The Data Transfer task is configured for a nightly import from the GCS bucket. The scheduling of the GCF is done by the Google Cloud Scheduler (a *cron job* service in the Google Cloud) [39], which sends a message to a so-called subscription in a Google Cloud Pub/Sub (a messaging service in the Google Cloud) topic [38]. The GCF, in turn, is configured that it fires when a message is received on the said subscription.
- The Google Ads data is extracted via Big Query Data Transfer Service and loaded into Big Query. The service creates its own data set that contains various partitioned tables and views with all accessible Google Ads data. The scheduling (nightly) of the transfer is defined in the task configuration of the data transfer.
- The DV360 data is loaded into a GCS bucket by the DV360 Data Transfer service. This happens every hour automatically. From the GCS bucket, they are imported nightly into a Big Query data set, again via a GCS CSV import task with the Data Transfer Service.

In a second step, the Ad Platform data in the first stage of the DWH is processed with Big Query views so that it can be well analyzed and linked to other data (DWH Stage 2). After this step, all collected data is in Stage 2 of the DWH. For Stage 3 they are connected with each other. Connecting the data is also done with views. Finally, the connected data is aggregated, standardized and normalized. This results in cross-application KPIs that are valuable for BI and the predictions. This forms the last step of the ETL process.

Unfortunately, views present bad performance, because each call to the view also generates underlying queries on the tables; thus, data needs to be calculated from start, which hinder

performance. However, in traditional DWHs, the problem can be solved by regularly extracting tables with indices from the views. This could also be implemented in Big Query if view performance were an issue.

The last part of the data warehouse architecture is how the data is consumed, while the consumers have different options to access the DWH data. The goal is to serve different DWH access options for different levels of technical skills and for different requirements of data insight depth. First, a **BI Dashboard** is integrated into the existing admin dashboard, which displays the most important and most frequently used BI reports and KPIs within the browser. The web application receives the required data via the back-end, which connects to the DWH using JDBC. However, the customizability and available features in this dashboard are limited. Visualization and table report can be downloaded as images or CSV files. For customizable reports, drill-down and drill-through functionalities, a power user tool which connects to the DWH directly is available, as a second option (*e.g.*, Google Data Studio or Tableau [25]). Analysts can perform their own data analysis and create their own interactive reports. A third option is the possibility to access the DWH tables and views directly and use them in custom SQL queries. In this way, even the technical and SQL-savvy data analyst is served and can draw his insights from the data. Querying the DWH with SQL queries can be done with the Big Query UI Console, the Google Cloud Command Line Interface (CLI) or with one of the offered client libraries.

### 3.2.2 Self-hosted Data Warehouse

In this section, a solution that is suitable for application owners who do not want to send their data to the Google Cloud (*e.g.*, banks) is presented. Different parts of the architecture are identical and will not be discussed in the same level of detail. Mainly, the differences to the above described Google Cloud DWH architecture are highlighted. Figure 3.3 depicts the described architecture.

In this design, the application is hosted in a trusted private cloud or on dedicated servers. The **IAM** and the **Backend Cluster**, the **Admin Dashboard** and the **Production DB** Cockroach cluster run in an in-house Kubernetes environment or in that of a private cloud. The goal is to host the data warehouse in the same environment.

The data warehouse is designed as a separate Cockroach DB cluster: the **DWH / Analytics DB**. It is deployed in the same Kubernetes environment as the Production DB Cluster. Consequently, the ETL process from the Production DB to the DWH is not identical to the cloud solution: Every night, a Kubernetes *cron job* in the Production DB Cockroach cluster triggers a CSV dump that converts the data tables of the Production DB into CSV files and stores them in a directory or the private cloud storage. A second *cron job* in the DWH Cockroach cluster controls the import of the saved CSV files and the loading of the data into Stage 1 of the DWH. Then, as in Big Query, the data is processed using views and the DWH Stage 2 is created.

Extracting the data from the ad platforms works identically to the cloud data warehouse solution. However, as shown in Figure 3.3, after extracting the data, it is not loaded into Big Query but the goal of each extracting process is to have a CSV file on the Google

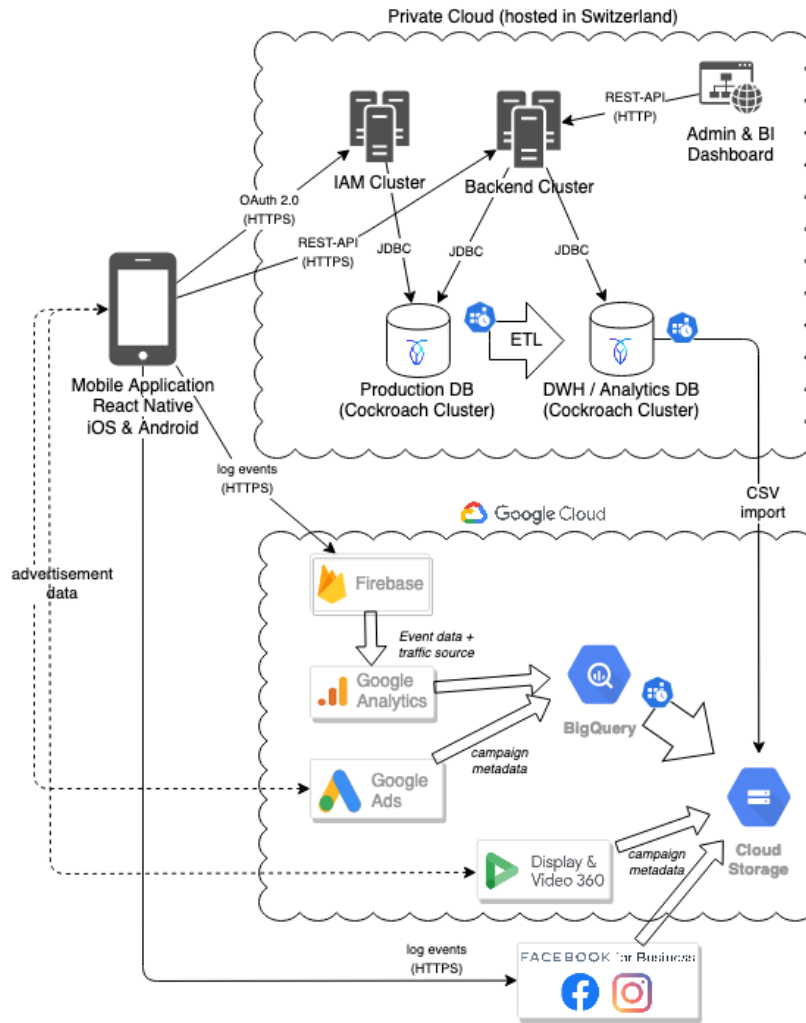


Figure 3.3: Self-hosted Data Warehouse Design

**Cloud Storage** bucket. The **Facebook Ads** API is called by a Google Cloud Function and the response is converted into a CSV file and stored in the GCS bucket and the **DV360** Data Transfer loads the data into a GCS bucket anyway. So the processes only differ from the cloud DWH solution in the sense that the data is not transferred from GCS to Big Query. The **Google Ads** data that is extracted with the Big Query Transfer Service as well as the **Google Analytics** data that is automatically stored in Big Query must be exported from **Big Query** as a CSV file and transferred to Google Cloud Storage. A Google Cloud Function is used for this purpose, as this is the only way to implement a recurring export from Big Query. The GCF is once again set up with the Google Cloud Scheduler and Google PubSub so that an export to GCS is triggered every night.

Another Kubernetes *cron job* in the DWH Cockroach Cluster triggers the imports of all mentioned CSV files from the GCS bucket and the loading of the contained data into the corresponding Stage 1 tables. The processing of the data to Stage 2 and the linking of the data to cross-platform KPIs and attributes (resulting in Stage 3) is done identically to the cloud data warehouse design.

The **BI Dashboard** in the web front-end receives the required data via the back-end, which connects to the DWH Cockroach DB using JDBC. The Power User BI tools connect directly to the database. Which protocol is used, depends on the BI tool.

### 3.3 Relevant Metrics and Attributes

In this section the relevant metrics for the business intelligence as well as the prediction sub system are identified and described.

#### 3.3.1 Metrics and Attributes for Business Intelligence

The following basic metrics were evaluated as relevant for the business intelligence system. Various other interesting metrics are calculated from these basic metrics. Some examples of these calculated metrics can be found below.

- No. of app installs
- No. of registrations
- No. of users stuck in the identification process
- No. of fully on-boarded users
- No. of paying users
- Total amount paid
- Costs of the campaign
- Costs of the ad group
- Costs of the ad

The relevant metrics depend on the on-boarding process and the functionalities of the app. In this design, it is assumed that the mobile app is a finance tech (Fin Tech) app, which contains some common app building blocks: The app contains a registration process that requires users to identify themselves (*e.g.*, to open a bank account) before they are considered as fully on-boarded. In addition, there is an app functionality related to a payment, the value of which is stored and can therefore be read. This payment can be of different size and can occur several times (*e.g.*, different subscriptions).

With the basic metrics listed above, various other values can be calculated. For example, the campaign costs per install, per registration, per fully-onboarded user or the customer acquisition costs (Costs of the campaign divided by No. of paying users). In order to improve the campaigning as much as possible, the campaign costs per paid amount should be optimized (Costs of the campaign divided by Amount paid). This metric can

be equated with the capital turnover, which sets the revenue in relation to the invested money. Depending on what service or product that amount was paid for, that metric may be called different: *e.g.*, cost per subscription amount, cost per purchase amount or cost per asset under management.

Furthermore, in business intelligence, it should be possible to segment the KPIs by some attributes. The following were identified as interesting and relevant segmentation attributes. They also depend on the app use case and the data being stored.

- Gender,
- Age group (*e.g.*, 25-34),
- Living area (*e.g.*, postal code grouped by first digit),
- Device operating system,
- Nationality, and
- Language.

In addition, there are segmentation attributes that are very app-specific. For example, an investment app can be segmented according to a selected risk profile, the selected investment horizon or the selected investment focus.

In the case of ad platforms, there are also options for splitting the data further. Once several campaigns, ad groups or advertisements have been set up and executed, their parameters and results can be compared with each other. For app campaigns, the settings that can be set when creating an app campaign are very limited compared to other campaign types. Below is a list of app campaign properties that are considered relevant for BI.

- Campaign type
- Campaign sub type
- App platform (*e.g.*, Apple App Store, Google Play Store)
- Start date
- End date
- Daily budget
- Target locations: One can set the area, where an ad campaign shall appear.
- Targeted location type: It is possible to either target people in the specified locations or people interested in the specified locations.
- Excluded location type: The exclusion of locations does also work either with people in or interested in a specified location.



- Target languages
- Bidding focus: The bidding algorithm can optimize toward the number of app installs or towards the number of an action event (*e.g.*, registration).
- Target cost value: The target value for cost per install or cost per action event (depending what bidding focus was chosen) can be defined.
- List of ad headlines
- No. of ad headlines
- Avg. length of headline
- List of ad descriptions
- No. of ad descriptions
- Avg. length of ad description
- No. of image ads
- No. of video ads
- No. of html5 ads
- List of audience groups

For non-app campaigns, more campaign and campaign criterion parameters can be retrieved. Especially the audience of the campaign can be examined better. For the sake of completeness, the relevant campaign properties that are only available for non-app campaigns are listed below.

- Audience gender female
- Audience gender male
- Audience gender unknown
- Audience age 18-24
- Audience age 25-34
- Audience age 35-44
- Audience age 45-54
- Audience age 55-64
- Audience age 65+
- Audience age unknown
- Audience household income top 10%

- Audience household income 11-20%
- Audience household income 21-30%
- Audience household income 31-40%
- Audience household income 41-50%
- Audience household income lower 50%

The data in Stage 2 of the DWH (see Section 3.2) is linked and processed in such a way that the DWH Stage 3 contains the KPIs and attributes listed above so that they can be read by the BI dashboard, the BI Power user tool and by self-written SQL statements.

### 3.3.2 Feature and Target Selection for the Prediction System

For the prediction system, the campaign attributes shown in Table 3.1 were defined as machine learning features. As mentioned, for app campaigns, the last 15 features cannot be set. Nevertheless, they were defined as desired features because they are usable for other campaign types.

Table 3.1: Selected features for the ML models

Feature Name	Feature type
campaign type	categorical
campaign sub type	categorical
app platform	categorical
start month	numerical
end month	numerical
campaign duration (no. of month)	numerical
daily budget	numerical
target language en	binary
target language de	binary
target language fr	binary
target language it	binary
bidding focus	categorical
target cost value	numerical
no. of ad headlines	numerical
avg. length of headline	numerical
no. of ad descriptions	numerical
avg. length of ad description	numerical
no. of image ads	numerical
no. of video ads	numerical
no. of html5 ads	numerical
audience gender female	binary
audience gender male	binary

audience gender unknown	binary
audience age 18-24	binary
audience age 25-34	binary
audience age 35-44	binary
audience age 45-54	binary
audience age 55-64	binary
audience age 65+	binary
audience age unknown	binary
audience household income top 10%	binary
audience household income 11-20%	binary
audience household income 21-30%	binary
audience household income 31-40%	binary
audience household income 41-50%	binary
audience household income lower 50%	binary

The features listed in Table 3.1 are used to predict campaign results. The KPIs in Table 3.2 were selected as labels for the machine learning. The labels focus on the on-boarding states and the most desired states (*fully on-boarded user* and *paying user*) should be predictable segmented by gender, age and paid amount (e.g., investment amount). The two KPIs *Customer acquisition costs* and *Cost per Assets Under Management (AUM)* are not defined as labels, as they are calculated from the campaign costs divided by the predicted number of paying users resp. from the campaign costs divided by the predicted AUM.

Table 3.2: Selected KPIs (labels) for the ML models

Label name	Label type
app installs	numerical
registrations	numerical
fully on-boarded users	numerical
paying users	numerical
total amount paid	numerical
fully on-boarded users female	numerical
fully on-boarded users male	numerical
fully on-boarded users age -14	numerical
fully on-boarded users age 15-24	numerical
fully on-boarded users age 25-34	numerical
fully on-boarded users age 35-44	numerical
fully on-boarded users age 45-54	numerical
fully on-boarded users age 55-64	numerical
fully on-boarded users age 65+	numerical
fully on-boarded users ios	numerical
fully on-boarded users android	numerical
fully on-boarded users age -14 male	numerical
fully on-boarded users age 15-24 male	numerical

fully on-boarded users age 25-34 male	numerical
fully on-boarded users age 35-44 male	numerical
fully on-boarded users age 45-54s male	numerical
fully on-boarded users age 55-64s male	numerical
fully on-boarded users age 65+ male	numerical
fully on-boarded users age -14 female	numerical
fully on-boarded users age 15-24 female	numerical
fully on-boarded users age 25-34 female	numerical
fully on-boarded users age 35-44 female	numerical
fully on-boarded users age 45-54 female	numerical
fully on-boarded users age 55-64 female	numerical
fully on-boarded users age 65+ female	numerical
paying users female	numerical
paying users male	numerical
paying users age -14	numerical
paying users age 15-24	numerical
paying users age 25-34	numerical
paying users age 35-44	numerical
paying users age 45-54	numerical
paying users age 55-64	numerical
paying users age 65+	numerical
paying users amount -3k	numerical
paying users amount 3k-7k	numerical
paying users amount 7k-15k	numerical
paying users amount 15k-35k	numerical
paying users amount 35k-75k	numerical
paying users amount 75k-150k	numerical
paying users amount 150k-300k	numerical
paying users amount 300k+	numerical
AUM	numerical
AUM per user	numerical

For the suggestions in the other direction, the same variables are used as basis. The user of the suggestion system can select one of the labels listed in Table 3.2 and define it as the desired outcome. The system will then search for campaigns with similar outcomes and list these parameters. The user can then modify the variables to meet his requirements. Central to the suggestions of the system is that the total budget is as low as possible. The total budget is calculated with  $budget_{total} = budget_{daily} \times duration_{campaign} \times 30$ .

### 3.4 Prediction System Architecture

In this section, based on the DWH design and the identified features and labels for the ML, a prediction system architecture is proposed to achieve the goal of successfully predicted

campaign outcomes. The prediction system architecture depends on the selected data warehouse architecture and the data storage needs of the application and data owner. For each of the proposed DWH designs, a suitable prediction system design is outlined.

### 3.4.1 With a Google Cloud DWH and Vertex AI for Model Training and Serving

Figure 3.4 shows the prediction system architecture if, in addition to the ad platform data, the app data can also be sent to the Google Cloud and therefore a Google Cloud Big Query data warehouse has been chosen. In this case, the Google Cloud Machine Learning service Vertex AI [64] can be used in its full potential.

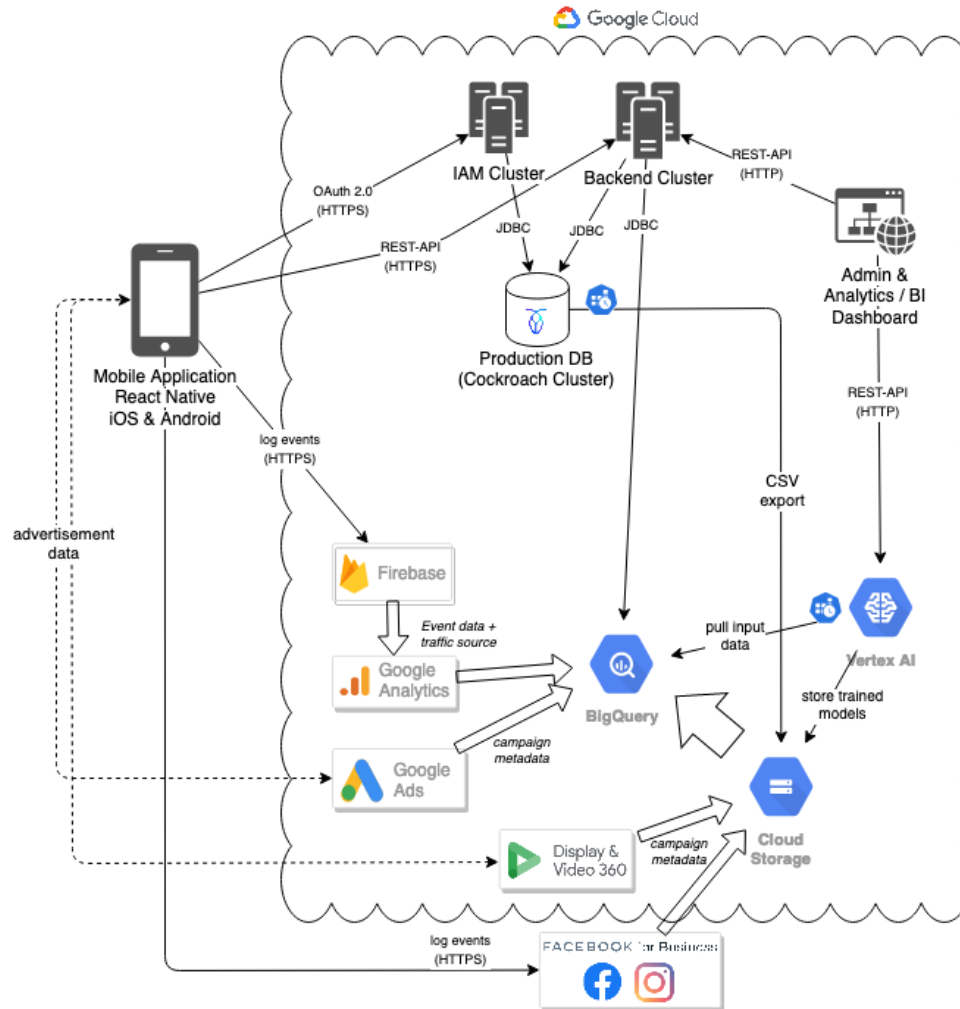


Figure 3.4: Overall Design with a Cloud DWH and a Vertex AI Prediction Component

In Stage 3 of the Big Query DWH, a view is prepared to match the machine learning requirements of **Vertex AI**: The defined features as well as the labels in one column each. In the Vertex AI ML engine, the DWH view is defined as the data set import source.

Vertex AI directly connects to the **Big Query DWH**. Vertex AI works well with various machine learning frameworks (*e.g.*, TensorFlow [83], scikit-learn [81] or PyTorch [76]). The Machine Learning code is containerized using Docker and is loaded into the Google Container Registry (GCR). The Vertex AI training job then pulls the Docker container directly from the GCR. During the training process, for each label specified in Table 3.2, a ML model is trained and stored in the specified **GCS bucket**.

The models are also deployed to cloud endpoints using Vertex AI. For this, the endpoint service of Vertex AI is connected to the GCS bucket where the trained models were previously stored. Predictions can be obtained by sending requests to the deployed endpoints or by using the Google Cloud libraries (*e.g.*, Python), which sends the same HTTPS request.

The requirement of the continuing learning system is fulfilled by triggering the Vertex AI training every week. This is done by scheduling a GCF with a Cloud Scheduler job, which starts the Vertex training. In addition, the GCF backups the old model and the training job then overwrites it with a new one. By doing this, the endpoints are automatically using the freshly trained models. In the future, the Vertex service *Pipelines* should handle the scheduled training but this feature is not available yet.

### 3.4.2 With a Self-hosted DWH and Self-hosted Model Training and Serving

If a self-hosted DWH is used, there is a higher likelihood that the data will also not want to be stored in a GCS bucket for model training with Vertex AI. A design for a fully self-hosted prediction system is described below and visualized in Figure 3.5.

The DWH view in Stage 3 is exported to a CSV file and stored in the internal **Document Storage**. The dockerized machine learning code (running on the Kubernetes **ML Cluster**) reads the CSV file in the internal document storage and processes the data directly. The trained models are stored back into the document storage. The new models are then hosted in the additional Kubernetes **Prediction Cluster** and exposed as prediction endpoints.

Exporting the CSV files, training the models and re-deploying the models is done as a bash script pipeline and kicked off with a Kubernetes *cron job* on a weekly basis. The predictions can then be retrieved by sending HTTPS request to the prediction endpoints.

### 3.4.3 With a Self-hosted DWH, Vertex AI Training and Self-hosted Model Serving

If the application data does not want to be stored in a GCS bucket, but the own server infrastructure or that of the chosen private cloud is too weak or not suitable to train the machine learning models, then there is a hybrid design, visualized in Figure 3.6, that can be chosen. The required DWH view is exported to the private **Document Storage** in

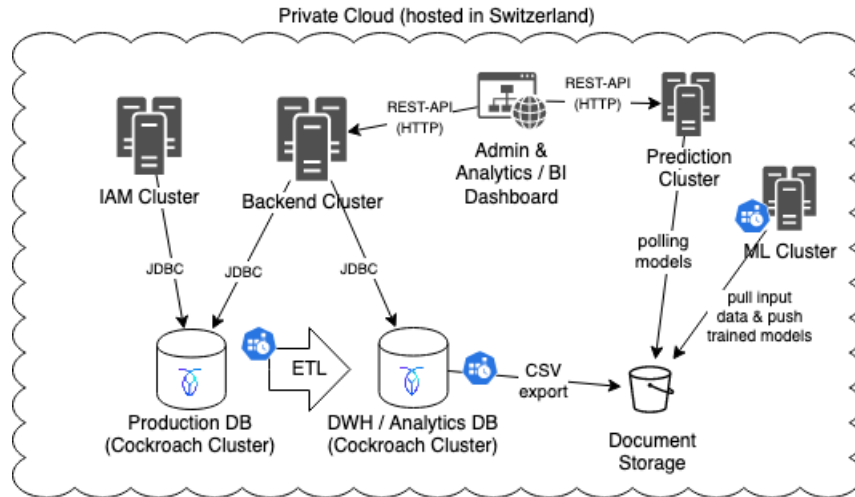


Figure 3.5: Self-hosted Model Training and Serving Design

the same way as in the fully self-hosted system (see Section 3.4.2), also scheduled with a Kubernetes *cron job*.

The dockerized machine learning code runs in **Vertex AI**, but pulls the required data (the CSV file) from the document storage in the private environment. The ML framework trains the models and stores them directly back into the private environment. Since the machine learning code is written and dockerized in-house, it can be ensured that the data is only stored in the memory of the machine learning code and cannot be retrieved from the outside. Vertex AI only executes the code in the Dockerfile and even if the container is not deleted after execution, no data from the DWH can be found within the container.

The models can then be deployed, again in the same way as in the fully self-hosted system architecture; in the Kubernetes **Prediction Cluster** and accessed from the outside via HTTPS requests.

Access to the document storage in the private environment can also be restricted so that only the used CSV file can be loaded from the Google Cloud IPs and also only models from the Google Cloud IPs can be stored in the document storage (path and IP restriction).

### 3.4.4 (Continues) Model Learning

Various sub processes across the data warehouse are scheduled with *cron jobs* and Google Cloud schedulers so that they export and/or transform data nightly. Business intelligence processes and analytics are automatically updated to the data from the previous day. With the prediction system, this would technically also be possible, but would not make financial sense, since training the models is computationally intensive and the input data for the machine learning only changes when a new campaign is completed. This design foresees a weekly update of the models. The pipelines are triggered by *cron jobs* or Google Cloud scheduling is used, depending on the design being implemented.

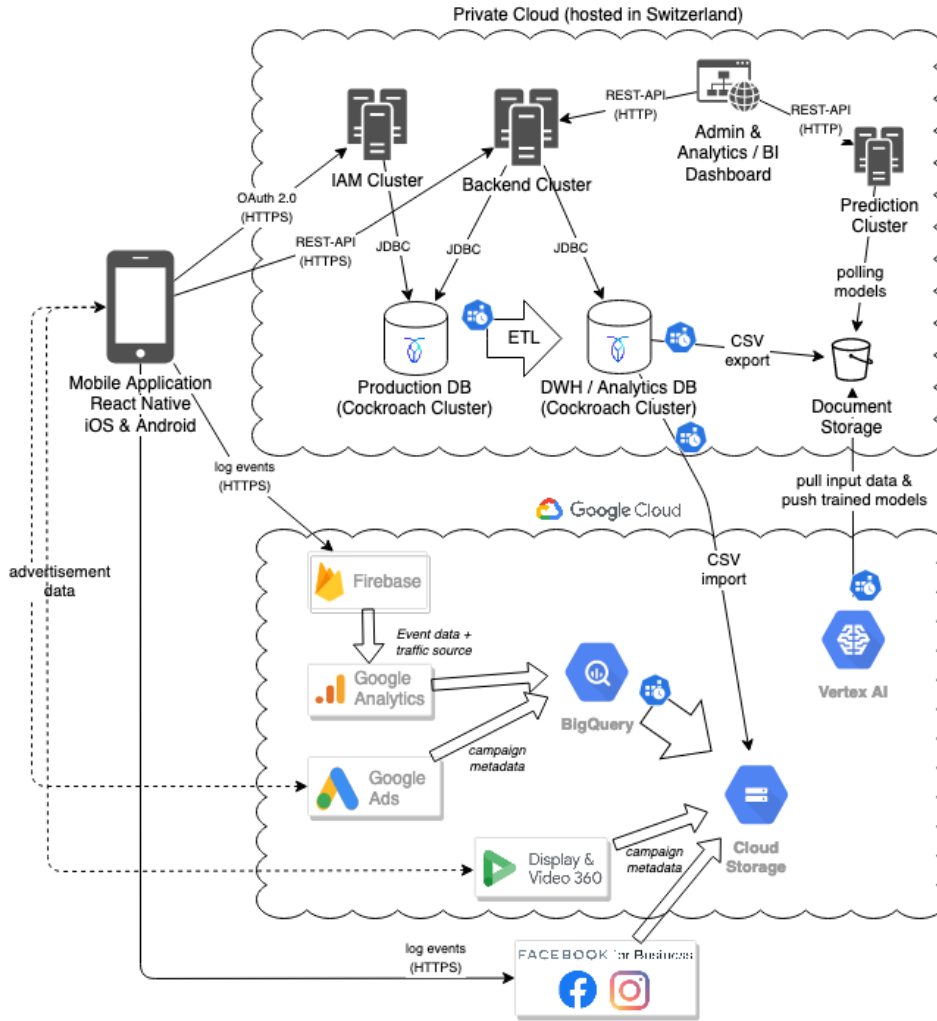


Figure 3.6: Design with a self-hosted DWH, Vertex AI Training and self-hosted Model Serving

Supervised machine learning algorithms are used for training the data. The prediction of the defined labels (see Table 3.2) form a regression problem, since all results can be any values in the positive value range. It was decided to not use an additional classification problem that predicts if the campaign overall is a success or not (see Section 2.6.2). The reason for this is that a human interaction would be needed to classify the past campaigns as success or fail. And if no human interaction is necessary, meaning that the success or fail can be calculated, then the overall success for the predicted campaign can also be derived from the other predicted values.

To train and build the models, the ML frameworks TensorFlow, scikit-learn and PyTorch can be used, as they are compatible with Vertex AI. This design does not define which ML algorithms should be used. Several should be tried and the one that performs best with the test data should be used. If there is a large change in the data or the amount of data, the selected ML algorithm should be reevaluated to ensure an optimal design. Supervised machine learning algorithms for regression problems that can be used are, for example, Linear Regression, Decision Tree, Random Forest, Support Vector Regression



or a Deep Neural Network Regression.

The machine learning code is containerized in a Docker image. This way it can be used on any machine and in any environment.

## 3.5 UI Features

The web application's UI, shown in Figure 3.2 and Figure 3.3 in the top right corner in each case, includes three functionalities.

- **App content management:** Translations of the app texts or adjustable app values (*e.g.*, the price of a subscription) can be managed without a new front-end release.
- **BI dashboard:** Contains the most important and most used reports and KPIs for fast and simple reporting.
- **Prediction interface:** The marketing analyst can enter the marketing campaign parameters with this interface, and KPIs and results of such a campaign are predicted.
- **Suggestion interface:** The marketing analyst enters a desired campaign outcome, and the system displays campaigns that performed almost as good as or better than the desired outcome. The result list is sorted by the costs of the campaign.



# Chapter 4

## Implementation

To evaluate the design described in Chapter 3, a prototype of the designed system was implemented. It was focused on the automated processing of data from the ad platforms and the mobile app resp. the app back-end. The goal was not to implement one part of the system in a detailed and clean way, but to implement the whole process from exporting data from the different systems, aggregating and connecting this data, machine learning on the connected data and providing the trained models as endpoints that can be called by the admin dashboard to get predictions, to see what hurdles and difficulties such a design brings.

### 4.1 Implemented Architecture Overview

While describing the implemented architecture, its components are highlighted in bold.

Figure 4.1 shows a diagram of the prototype architecture. The **Mobile Application**, the **IAM** and the app **Backend** in Kubernetes Cluster and the Cockroach Cluster with the Cockroach **Application DB** have been implemented already. Also existing was an **Admin Dashboard** front-end hosted on the back-end server that could be used to customize the app content. The existing system could be deployed to the local machine with little effort using Docker Compose [73], so only a Docker engine is needed rather than a Kubernetes engine. This made it easier to extend these components and add the new components to the system.

The mobile app and the app back-end have been extended so that app events are logged and persisted in an event table in the Production DB and the Firebase SDK has been integrated in the mobile app so that events can be logged directly to **Firebase**. Firebase was connected to **Google Analytics** so that events are automatically forwarded to GA. In addition, **Google Ads** was set up and Firebase was connected with it so that the events can be interpreted as a conversion and the Google Ads algorithm can learn with it. Campaigns were set up on Google Ads. The integration was limited to Google Ads, as the additional integration of Facebook Ads and DV360 would have massively increased the complexity of the prototype and thus the effort of this implementation.

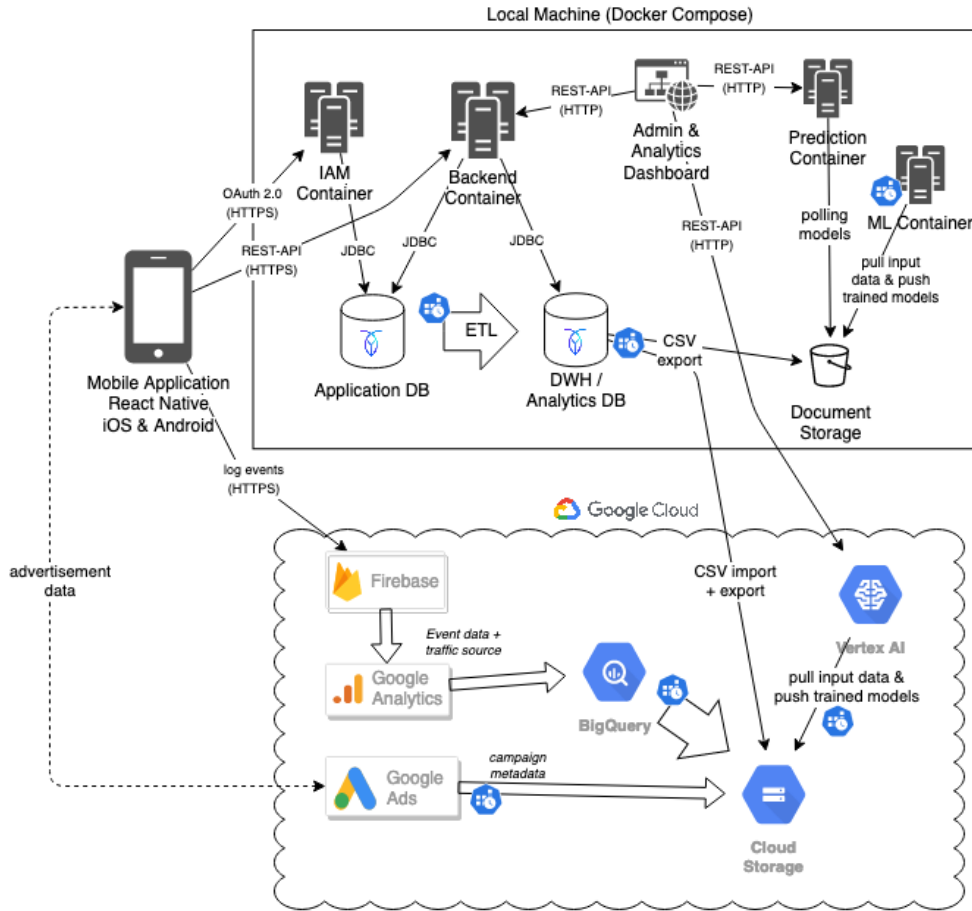


Figure 4.1: Architecture of the Prototype Implementation

The data warehouse implementation is a simplified version of the self-hosted DWH design described in Section 3.2.2. A Cockroach **DWH / Analytics DB** Docker container was set up and the events data stored by the app as well as the data of the created accounts are exported and imported into the data warehouse. Google Ads data is exported to a CSV file via a Google Cloud Function and Google Ads API and imported via Cockroach Import statement. Google Analytics was connected to **Big Query** so that the GA event logs are exported directly to Big Query. From there, the data is converted to a CSV file via GCF and stored in a Google **Cloud Storage** bucket. Another Cockroach Import statement then pulls the GA data into the DWH. In the DWH, the data is processed, enriched and connected with the help of views.

The prediction model training was on the one hand implemented self-hosted (on the local computer) with the **ML Container** and on the other hand the model training was additionally outsourced to **Vertex AI**, in order to be able to evaluate the two implementation types in terms of performance and costs. The view in Stage 3 of the DWH is converted to a CSV file with a Cockroach Export statement and stored on the computer. For machine learning, TensorFlow was chosen. The TensorFlow ML code imports the CSV file, trains the models and stores them again, on the computer or in the GCS bucket. In the self-hosted implementation, the models are then deployed as prediction endpoints using the **Prediction Container** with TensorFlow Serving. For the Vertex AI implementation

variant, the models are provided as endpoints by Vertex AI.

The admin dashboard has been extended with a simple prediction functionality to an **Admin & Analytics Dashboard**. Parameters of new campaigns can be entered via front-end input fields and the predicted campaign performance will be displayed. The predicted CAC are returned too.

## 4.2 Event Logging

This section describes how events are tracked by the app back-end and how the implemented Firebase SDK is used.

### 4.2.1 Data Tracking within the App

Whenever possible, events are logged from the back-end. That is, when an API endpoint is called from the mobile app (e.g. create bank account), the front-end does not send an additional request to save an event, but the back-end calls a `saveEvent` function at the end of the called method, which saves a new row in the events table. Only if no request is sent from the front-end to the back-end, but an event should be saved anyway, a request is sent to the implemented event endpoint. An example of this is when a user opens a support screen, which normally does not trigger a back-end request.

There are two types of events. The pre-registered events and the normal (or post-registered) events. Pre-registered events are events that happen before registration and therefore no account ID is known yet. However, it is useful to store these events, for example, to find out which screens are interacted with before registration and how often.

Listing 4.1 shows the implemented POST endpoint `apiv1events{event_type}` for normal (post-registered) events in the `EventController` and the method from `EventService` that is called. The `EventService`'s `storeEvent` method is the method that is also called by other endpoint functions to log an event. The event timestamp, the event type, the advertisement IDs of Apple and Google (`idfa` and `gpsAdid`) and the account ID that triggered the event are stored. For pre-registered events, no account ID can be stored. The app back-end is implemented in Java Spring Boot.

```

1 // EventsController
2 @SecurityRequirement(name = "OAuth2")
3 @RequestMapping(path = "/api/v1/events/{eventType}", method =
    RequestMethod.POST)
4 @ResponseStatus(HttpStatus.OK)
5 public void triggerEventFromUI(@Valid @PathVariable("eventType")
    EventType eventType) {
6     Account account = securityService.getLoggedInAccount();
7     eventService.storeEvent(account, eventType);
8 }
9
10 // EventsService

```

```

11 public void storeEvent(Account account, EventType eventType) {
12     Event event = new Event();
13     event.setCreatedAt(new Date());
14     event.setType(eventType);
15     event.setAccount(account);
16     event.setIdfa(account.getIdfa());
17     event.setGpsAdid(account.getGpsAdid());
18     eventRepository.save(event);
19 }

```

Listing 4.1: The Event endpoint and the EventService methods to store the logged events.

To be able to imagine which event types are stored, in Listing 4.2 the EventTypes Enum is shown in an abbreviated form. There are pre-registered events (*e.g.*, a failed registration attempt), on-boarding events (*e.g.*, the successful identification of the user) or app usage events (*e.g.*, the user changed the language).

```

1 export enum EventTypes {
2     // Pre-register events
3     APP_OPENED = 'APP_OPENED',
4     REGISTRATION_ATTEMPT = 'REGISTRATION_ATTEMPT',
5     [...]
6
7     // onboarding
8     REGISTERED = 'REGISTERED',
9     PHONE_NUMBER_ADDED = 'PHONE_NUMBER_ADDED',
10    PHONE_NUMBER_VERIFIED = 'PHONE_NUMBER_VERIFIED',
11    [...]
12    IDENTITY_VERIFICATION_ACCEPTED = 'IDENTITY_VERIFICATION_ACCEPTED',
13    [...]
14
15    // app usage
16    LOGGED_IN = 'LOGGED_IN',
17    SIMULATION_WINDOW_OPENED = 'SIMULATION_WINDOW_OPENED',
18    HELP_VIEWED = 'HELP_VIEWED',
19    LANGUAGE_CHANGED = 'LANGUAGE_CHANGED',
20    [...]
21 }

```

Listing 4.2: Some of the event types that are logged and saved in the Events DB table.

In addition to the events, information about the user and their account is stored on the account entity. The following is a list of what is stored.

- User information
  - First name
  - Last name
  - Living address
  - Nationality
  - Date of birth
  - Gender
  - Language

- Device meta-data
  - Apple Ad ID
  - Google Ad ID
  - Used app version
  - Used phone operating system (OS)
  - Used OS version
  - If the user has a jailbroken iPhone
  - The Firebase cloud messaging token
  - If the user's phone has WhatsApp installed to reach the user for support
- Account and on-boarding related data
  - The on-boarding step the account is currently in (*e.g.*, IDENTIFICATION\_SUBMITTED or ON\_BOARDED)
  - The chosen investment risk profile
  - The chosen investment horizon
  - The chosen investment topics the user wants to invest in
  - and other app related properties

### 4.2.2 Firebase Integration and Logging to Google Analytics

The mobile application is implemented in React Native [77]. On one hand, it was extended with the event endpoint requests where it was needed, and on the other hand, the Firebase SDK was integrated.

The Firebase SDK logs most Google Analytics events automatically, but custom event logs must be implemented in the front-end code. Listing 4.3 shows that after the successful registration of a new user not only the REGISTERED event is sent to GA (line 7), but also the received user resp. account ID (line 6). This is an important step, which allows to connect the data from the application database with the GA and Google Ads data.

Only the events that are relevant for the Google Ads algorithm and can be interpreted as a conversion are sent to Google Analytics. All other events are tracked only via the application back-end.

```

1 import analytics, {firebase} from '@react-native-firebase/analytics';
2
3 const onContinue = useCallback(async () => {
4   const tokenPair = await createUserCall({email, password});
5   dispatch(setTokens(tokenPair.accessToken, tokenPair.refreshToken));
6   const newUser = await getUserCall();
7
8   setAnalyticsUserId(newUser);
9   logEventToFirebase(EventTypes.REGISTERED);
10  [...]
11 }
```

```

12
13 const setAnalyticsUserId = (user: UserType) => {
14   firebase.analytics().setUserId(user.id);
15 };
16
17 const logEventToFirebase = (eventType: EventTypes) => {
18   firebase.analytics().logEvent(eventType.toString());
19 };

```

Listing 4.3: The front-end code which sends the user ID and the REGISTERED event to Google Analytics after a successful registration.

## 4.3 Data Warehouse

This section describes the implementation of the data warehouse. Firstly, it describes how the data is exported from the various sources and secondly, how it is then loaded into Stage 1 of the DWH and processed in Stages 2 and 3 so that it can be used by the BI tools and the prediction system.

One exporter each was implemented for the Application DB data, the Google Ads data, and the Google Analytics data. All three work slightly in a different way and are interesting to look at in more detail, because if further systems should be integrated, it is very likely that one of these export approaches can be used.

### 4.3.1 Application Data Export

The Account entity and all related entities are exported. This results in an export of the data of the database tables *Account*, *Event* and an additional auxiliary table.

A SQL select statement was developed that combines all data into one view. This select statement can be seen in Listing 4.4. First, the account data is anonymized: First name, last name and home address are removed, only the zip code is kept. Email and phone number are not included in the *Account* table anyway, since they are stored in the user entity of the IAM. Additionally, auxiliary tables are joined and converted to columns (lines 8-26). The event data is split into pre-registered (lines 29-37) and post-registered (lines 39-46), because they need to be associated differently with the account data: The post-registered events can be joined using the account ID (line 51) while the pre-registered events uses the Firebase Cloud messaging token (*fcm\_token*) and the two advertisement IDs (*idfa* and *gps\_adid*) as connection variables (lines 63-64).

```

1 WITH
2   /* Accounts without sensitive attributes */
3   anonym_accounts AS (
4     SELECT account_id,
5            gender,
6            onboarding_state,
7            [...]
8            MAX(sinn_profil) AS sinn_profil_2,

```



```

9          CASE
10             WHEN MIN(sinn_profil) IS NOT NULL THEN
11                 concat(MIN(sinn_profil), ' & ', MAX(sinn_profil))
12             ELSE
13                 NULL
14             END AS sinn_profil_pair
15 FROM (
16     SELECT a.id AS account_id,
17            a.gender,
18            a.onboarding_state,
19            [...]
20            s.sinn_profil
21     FROM account AS a
22          LEFT OUTER JOIN account_sinn_profil AS s ON a.id = s.
23                  account_id
24     ORDER BY s.sinn_profil)
25 GROUP BY account_id,
26          gender,
27          [...]
28 ),
29 /* Pre-registered events only (only fcm_token is available) */
30 pre_registering_events AS (
31     SELECT id                AS event_id,
32            fcm_token         AS event_fcm_token,
33            [...]
34     FROM events
35     WHERE account_id IS NULL
36           AND fcm_token <> ''
37           AND fcm_token IS NOT NULL
38 ),
39 /* All events triggered for users having an account_id */
40 post_registering_events AS (
41     SELECT id                AS event_id,
42            account_id        AS event_account_id,
43            fcm_token         AS event_fcm_token,
44            [...]
45     FROM events
46     WHERE account_id IS NOT NULL
47 )
48 /* Add account attributes to all (pre- and post-registering) events */
49 SELECT *
50 FROM post_registering_events
51      LEFT OUTER JOIN anonym_accounts ON event_account_id = account_id
52 UNION
53 SELECT DISTINCT ON (event_id,
54                    event_account_id,
55                    event_created_at,
56                    event_fcm_token,
57                    event_gps_adid,
58                    event_idfa,
59                    event_type,
60                    event_attribute_key,
61                    event_attribute_value) *
62 FROM pre_registering_events
63      LEFT OUTER JOIN anonym_accounts

```

```

64      ON event_fcm_token = fcm_token AND event_idfa = idfa AND
        event_gps_adid = gps_adid

```

Listing 4.4: The SQL select statement for exporting the data from the application's production database.

To export a SQL select statement from a cockroach database to a CSV file, a bash script with the cockroach command in Listing 4.5 is needed. This allows the CSV file to be placed directly on the host machine, in a GCS bucket, or in any other document storage. Lines 4 or 6 specifies the destination, line 7 tells the process to save NULL-fields as "NULL" in the CSV file and the SQL statement is pasted at line 9. For exporting to a GCS bucket, a service account is needed and the credentials need to be integrated in the bucket Uniform Resource Identifier (URI). Line 4 contains the command to Base64 encode the service account credentials and add it to the URI. The whole command can be scheduled on the Production DB with a Kubernetes *cron job* and on the local environment with a *cron job* in a Docker container.

Listing 4.6 shows an example Kubernetes *cron job* which was implemented for triggering the export of the data from the Production DB. It has not been deployed to production. The *cron job* would run the export script every night at 2 AM (CEST).

```

1  cockroach sql --insecure -e "
2  EXPORT INTO CSV
3    -- to GCS bucket:
4    --'gs://${BUCKET_NAME}?AUTH=specified&CREDENTIALS=$(cat /root/
      gcloud_service_account_creds.json | base64 --wrap=0)'
5    -- to the local file storage:
6    'nodelocal://self/data_transfer/v_events_anonymized_accounts.csv'
7    WITH nullas = 'NULL'
8    FROM
9    <<THE SQL STATEMENT GOES HERE>>
10   ;"

```

Listing 4.5: The cockroach export command, which is used to export data from a cockroach DB to a CSV file and save it either to the local file system, a GCS bucket or another document storage.

```

1  apiVersion: batch/v1beta1
2  kind: CronJob
3  metadata:
4    name: export-cron
5  spec:
6    schedule: "0 0 * * *"
7    concurrencyPolicy: Forbid
8    jobTemplate:
9      spec:
10       template:
11         spec:
12           containers:
13             - name: production-db-postgres
14               image: cockroachdb/cockroach
15               command: ["/bin/bash", "/csv-export-app-account-and-event-
                        data.sh"]

```

```
16 restartPolicy: OnFailure
```

Listing 4.6: The Kubernetes *cron job* YAML file which could be used to export the application data from the production DB every night at 2 AM (CEST).

### 4.3.2 Google Analytics Data Export

As mentioned earlier, the Google Analytics data is automatically saved to a Big Query table when GA is linked to Big Query. The linking could be done via the settings in the Google Analytics UI.

The data in Big Query is structured by Firebase Project IDs. The GA data can be found under the path `<FIREBASE_PROJECT_ID>:analytics_<ANALYTICS_ID>.events_*`. The data is stored in a partitioned table. For each day a new partition is created automatically (e.g., partition *events\_20210601*). In addition, the data in the tables are nested. This means that table columns can contain not only the known primitive database types but also objects, maps and arrays, comparable to a JSON file, which makes querying the data more difficult. It also complicates the export as CSV file, because the data has to be flattened. Another difficulty is that the new partitions are created with a table expiration time of 60 days and this cannot be changed. This means, for an implementation of a Cloud DWH in Big Query, that the data of the newly created partitions must be copied nightly to another table where no table expiration is set. However, for a self-hosted DWH, the following implementation is sufficient.

As a first step, a Big Query View was created, which combines the data in the partitioned *events* table into a single table, and additionally selects the columns relevant for the DWH and flattens them in such a way that a normal table structure is created. A snippet of the `CREATE VIEW` statement that can be executed in the CLI using the Google Cloud SDK can be seen in Listing 4.7. Lines 9 and 10-13 show how a nested object can be flattened. How mappings and arrays are unnested can be seen in lines 20-23. Line 15 then adds the unnested Google Click ID to the query output.

```
1 CREATE OR REPLACE VIEW '<FIREBASE_PROJECT_ID>.analytics_270412022.
  v_events' AS
2 SELECT
3   event_date ,
4   event_timestamp ,
5   event_name ,
6   [...]
7   user_id ,
8   user_pseudo_id ,
9   device.operating_system           device_operating_system ,
10  [...]
11  traffic_source.name               traffic_source_name ,
12  traffic_source.medium             traffic_source_medium ,
13  traffic_source.source             traffic_source_source ,
14  [...]
15  ep_gclid.value.string_value       ep_gclid ,
16  [...]
17 FROM
```

```

18  '<FIREBASE_PROJECT_ID>.analytics_270412022.events_*'
19  [...]
20  LEFT OUTER JOIN
21  UNNEST(event_params) AS ep_gclid
22  ON
23  ep_gclid.key = "gclid"
24  [...]

```

Listing 4.7: The Big Query CREATE VIEW statement to union all partitioned event tables and flatten the nested objects.

Based on the created view, the automated nightly export process can be started. Manually exporting the data from Big Query to a CSV file is easy to implement. Using the CLI command in Listing 4.8, the data in the same Google Cloud account can be transferred to the bucket even without credentials. However, scheduling the same task is a bit more tedious. The Google Cloud SDK can be installed on a Docker container and a *cron job* could nightly invoke the command in Listing 4.8.

```

1  bq extract \
2  --destination_format CSV \
3  "<FIREBASE_PROJECT_ID>:analytics_270412022.v_events" \
4  gs://<BUCKET_NAME>/bigquery-ga-export.csv

```

Listing 4.8: The Google Cloud CLI command to export a Big Query View to a GCS bucket.

To avoid having to control operations within the Google Cloud from a Docker container outside the cloud, the export was implemented and scheduled with Google Cloud services instead: A Google Cloud Function reads the data from the Big Query View, converts it to a CSV file and stores it in the GCS bucket. The GCF is implemented that it can be triggered with a Google Pub/Sub message. Listing 4.9 shows a snippet of the mentioned Google Cloud Function written in Java. The Java class overrides the *accept* method which requires a *PubSubMessage* input parameter. The message can be read as it is done in line 11. The parameters defined in lines 3-7 are combined into a Big Query Job object (lines 15-18) and the job is started in lines 19-22.

```

1  public class CSVExtractor implements BackgroundFunction<PubSubMessage> {
2      Logger logger = Logger.getLogger(CSVExtractor.class.getName());
3      String projectId = "<FIREBASE_PROJECT_ID>";
4      String datasetName = "analytics_270412022";
5      String tableName = "v_events";
6      String destinationUri = "gs://" + <BUCKET_NAME> + "/ga-events.csv";
7      String dataFormat = "CSV";
8
9      @Override
10     public void accept(PubSubMessage pubSubMessage, Context context) {
11         logger.info("Received message with id " + pubSubMessage.getMessageId());
12         try {
13             BigQuery bigquery = BigQueryOptions.getDefaultInstance().
14                 getService();
15
16             TableId tableId = TableId.of(projectId, datasetName, tableName);
17             Table table = bigquery.getTable(tableId);

```

```

18     Job job = table.extract(dataFormat, destinationUri);
19     Job completedJob =
20         job.waitFor(
21             RetryOption.initialRetryDelay(Duration.ofSeconds
22                 (1)),
23             RetryOption.totalTimeout(Duration.ofMinutes(3)));
24     if (completedJob.getStatus().getError() != null) {
25         [...]

```

Listing 4.9: The Google Cloud Function which extracts the data from the Big Query View, transforms it to a CSV file and loads it to the GCS bucket.

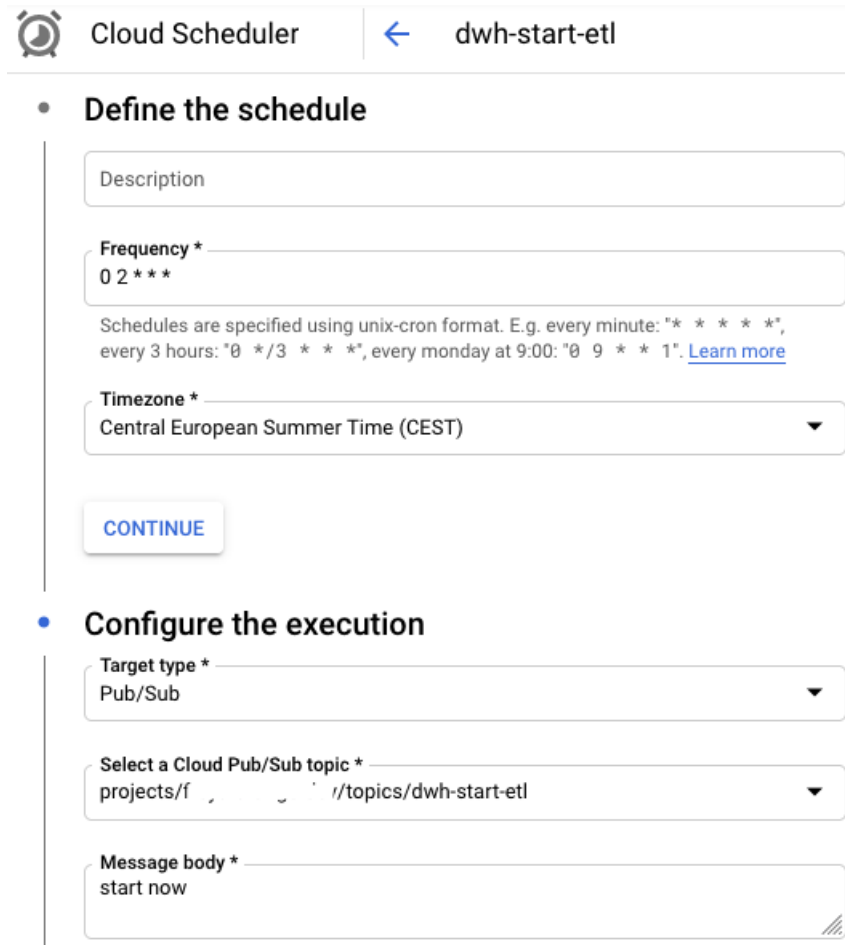
The GCF was manually deployed to the *europe-west6* region of the Google Cloud. The function was allocated with 512 MB of memory and the timeout was set to 60 seconds. A Cloud Pub/Sub topic *dwh-start-etl* was created for triggering the GCF. This means that every time a message is written to this topic, the GCF is executed. Finally, a Google Cloud Scheduler job was set up to send a message to the *dwh-start-etl* topic at 2 AM to start the GCF. Figure 4.2 shows the configuration of the Cloud Scheduler job. The job is defined by a cron statement. In addition, a development runner was implemented that allows the cloud function to work on the local machine triggered by HTTP requests instead of Pub/Sub messages.

### 4.3.3 Google Ads Data Export

The export of the Google Ads data was also implemented with a Google Cloud Function. The GCF is implemented in Java and sends requests to the Google Ads API. In order to connect to the Google Ads API, a registration process for a developer token must be completed. An application with a completed questionnaire and an architectural sketch of the system is required to receive a token. If the application is accepted, the token can be found in the Google Ads UI console. The whole process requires admin rights on the so-called My Client Center (MCC) account of Google Ads. Also, a Google Cloud service account is needed [56], which has system wide domain delegation rights [41][63].

The GCF for the Google Ads data export contains six requests to the Google Ads API. The requests are sent to the API one by one, the response is converted to a CSV file and stored on the local machine. The GCF can be scheduled with Google Cloud Scheduler just like the Google Analytics export function (see Section 4.3.2). With the implemented function, the following Google Ads reports are queried.

- *Campaign Performance Report*: All existing campaigns and their meta data and parameters.
- *Campaign Criterion Report*: The criterion of a campaign. Can be connected with the belonging campaign.
- *Ad Group Report*: The ad groups of a campaign. Can be connected with the belonging campaign.



The screenshot shows the Google Cloud Scheduler configuration interface. At the top, there's a header with a clock icon, the text 'Cloud Scheduler', a back arrow, and the job name 'dwh-start-etl'. Below this, there are two main sections: 'Define the schedule' and 'Configure the execution'.

**Define the schedule**

- Description:** A text input field.
- Frequency \*:** A text input field containing '0 2 \* \* \*'. Below it, a note explains the unix-cron format: 'Schedules are specified using unix-cron format. E.g. every minute: "\* \* \* \* \*", every 3 hours: "0 \*/3 \* \* \*", every monday at 9:00: "0 9 \* \* 1"'. A link 'Learn more' is provided.
- Timezone \*:** A dropdown menu showing 'Central European Summer Time (CEST)'.
- CONTINUE:** A blue button.

**Configure the execution**

- Target type \*:** A dropdown menu showing 'Pub/Sub'.
- Select a Cloud Pub/Sub topic \*:** A dropdown menu showing 'projects/f /topics/dwh-start-etl'.
- Message body \*:** A text input field containing 'start now'.

Figure 4.2: Configuration for the Google Cloud Scheduler job which triggers the Google Cloud Functions

- *Ad Group Ad Report*: The ads that belong to an ad group. Can be connected to the ad group and should be connectable to the Click View Report (see below).
- *Ad Asset Report*: The assets that belong to an ad. Can be connected to the belonging ad.
- *Click View Report* (see Section 2.3.1): Contains click data with the Google Click ID and an Ad Group Ad reference. Therefore, should be connectable to the Ad Group Ad Report.

It is important to mention that the Click View Report can only be retrieved for the last 90 days. In addition, the API does not support historization. Consequently, changes (*e.g.*, a campaign parameter) and their historization must be handled by the DWH. Historization was not implemented for this prototype. If a campaign is changed, the old one is overwritten. Therefore, for this prototype it would be suggested to stop a running campaign and start a new one instead of changing the running campaign.

The code snippet in Listing 4.10 illustrates how the Google Ads API is used in the Java GCF. The rest of the GCF is similar in structure to that of the GA Export (see Listing 4.9).

Line 1 initializes the Google Ads Client. The `ads.properties` file which is needed for the initialization contains the developer token, the user email (it must be one of the Google Workspace domain) and the service account credentials path. The search query is defined in lines 5-12 and then passed into the `SearchGoogleAdsStreamRequest` object in lines 14-18. Instead of a data stream it is also possible to use a pagination endpoint. The API is called in line 21 and the response read, parsed and added to the `csvRows` list in lines 33-48. Before that, the CSV header row is added in lines 24-31. As seen in line 41, the different data types need to be all parsed to String. The `CSVConverter` in line 49 then finally converts and saves the CSV file to a specified path.

```

1  GoogleAdsClient googleAdsClient = GoogleAdsClient.newBuilder().
    fromPropertiesFile(new File("./ads.properties")).build();
2  GoogleAdsServiceClient googleAdsServiceClient =
3      googleAdsClient.getLatestVersion().createGoogleAdsServiceClient();
4
5  String searchQuery =
6      "SELECT "
7          [...]
8          + "campaign.name,"
9          + "campaign.start_date,"
10         + "campaign.status,"
11         [...]
12         + " FROM campaign";
13
14  SearchGoogleAdsStreamRequest request =
15      SearchGoogleAdsStreamRequest.newBuilder()
16          .setCustomerId(Long.toString(CUSTOMER_ID))
17          .setQuery(searchQuery)
18          .build();
19
20  ServerStream<SearchGoogleAdsStreamResponse> stream =
21      googleAdsServiceClient.searchStreamCallable().call(request);
22
23  List<String[]> csvRows = new ArrayList<>();
24  csvRows.add(
25      new String[] {
26          [...]
27          "campaign_name",
28          "campaign_start_date",
29          "campaign_status",
30          [...]
31      });
32
33  for (SearchGoogleAdsStreamResponse response : stream) {
34      for (GoogleAdsRow googleAdsRow : response.getResultsList()) {
35          csvRows.add(
36              new String[] {
37                  [...]
38                  googleAdsRow.getCampaign().getStartDate(),
39                  googleAdsRow.getCampaign().getStatus().name(),
40                  googleAdsRow.getCampaign().getAppCampaignSetting().getAppStore
                      ().name(),
41                  Long.toString(googleAdsRow.getCampaignBudget().getAmountMicros
                      ()),
42                  [...]

```

```

43         googleAdsRow.getCampaign().getUrlCustomParametersList().stream
44             ()
45             .map(o -> o.getKey() + ":" + o.getValue())
46             .collect(Collectors.joining(",")),
47     });
48 }
49 CSVConverter.toCSV("./data_transfer/campaignReport.csv", csvRows);

```

Listing 4.10: The code snippet which calls the Google Ads API and transforms the response to a CSV file.

### 4.3.4 Importing the Data into DWH Stage 1

At this point in the DWH ETL process, all data is exported to CSV files and resides on the local machine or in the GCS bucket. This means, the next step is to load this data automatically and every day into Stage 1 of the DWH. Using Docker Compose and a Cockroach DB Docker image, a new Docker container was spun up for the DWH Cockroach DB. Listing 4.11 shows what the service definition looks like. When the container is started, first an initialize script is executed and then a Cockroach DB single node is booted (line 6). In addition to the data storage volume and the initialize script, a data transfer folder is mounted into the container (line 10) to access the CSV files already on the host machine. This would not be necessary in a production deployment, since all files are pulled from document storages. The standard ports to access the Cockroach DB and the Cockroach Web UI had to be changed, because the Production DB already uses the standard Cockroach DB port 26257 and the application back-end already uses port 8080 (line 12-13).

```

1  version: "3"
2  services:
3    dwh-db:
4      image: cockroachdb/cockroach
5      container_name: dwh-db
6      command: shell -c 'chmod a+x /etc/cockroach/conf/init-dwh-db.sh; /
7                  etc/cockroach/conf/init-dwh-db.sh & /cockroach/cockroach start-
8                  single-node --insecure'
9
10     volumes:
11       - ./data/node_1:/cockroach/cockroach-data
12       - ./init-dwh-db.sh:/etc/cockroach/conf/init-dwh-db.sh
13       - ./data_transfer:/cockroach/cockroach-data/extern/data_transfer
14     ports:
15       - "26258:26257" # 26257 occupied by production db
16       - "8081:8080"   # 8080 occupied by backend
17     environment:
18       - COCKROACH_USER=${DWH_COCKROACH_USER}
19       - COCKROACH_DATABASE=${DWH_COCKROACH_DATABASE}

```

Listing 4.11: The Docker Compose YAML file to start the DWH DB as a Cockroach DB.

To import the data in the CSV files into the Cockroach DB, a Cockroach import command as in Listing 4.12 is used for each import. As with the export, the storage URI for the



import can be either a GCS bucket URI, a URI to the local machine (as in line 5) or to another document storage. Before the commands are executed for the first time, the table schemas must be created with normal SQL `CREATE TABLE` commands. For data sources that always provide all data, the table can be truncated before the import. For data sources that provide only a part of the data, the old data must not be deleted, of course. Lines 8-10 define the import options. In the case of this example, a comma separated CSV is expected, the string `NULL` is interpreted as NULL and the header line is ignored (line 10).

The commands are scheduled with a *cron job* in a separate Docker container. The data is imported each day at 3 AM. The architecture of the *cron job* container is discussed in Section 4.3.7, as the same *cron job* container is also used to export data from the DWH.

```

1 cockroach sql --insecure -e "
2   TRUNCATE TABLE t_googleads_campaigns;
3   IMPORT INTO t_googleads_campaigns
4   CSV DATA (
5     'nodelocal://self/data_transfer/campaignReport.csv'
6   )
7   WITH
8     delimiter = ',',
9     nullif = 'NULL',
10    skip = '1'
11  ;"
```

Listing 4.12: The `cockroach` command to import a CSV file from the local machine to the Cockroach DB table.

### 4.3.5 Transforming the Data to Stage 2

The next step, Stage 2, is the transformation of the data. The data from different data sources still remain separate from each other. In the next subsections the transformations in this stage are described.

#### Application Data Transformations

The account data of the application DB is enriched, for example by grouping and aggregating the event data by account ID. In this way, additional KPIs can be obtained at the account level. The following transformations were implemented.

- The date of birth is used to calculate the age and create an age group property. The age groups in Google Ads for example are 18-24, 25-34, 35-44, and so on, instead of 10-20, 20-30 and 30-40. In this prototype the age group is calculated the same way. It can be calculated by subtracting 5 years from the age, rounding to the tens and then adding the 5 years again (*e.g.*, the date of birth 1988-05-06 is the age 33 and becomes the age group 25-34). Listing 4.13 shows how the age group is calculated in SQL.

```

1 FLOOR(
2     (EXTRACT(days FROM
3         (CAST(CURRENT_DATE AS TIMESTAMP) - CAST(dob AS TIMESTAMP)))
4 / 365 - 5) / 10) * 10 + 5 as age_group,

```

Listing 4.13: The SQL statement snippet which performs the age group calculation.

- The postal code is rounded to the thousands, in that way a postal code region is created which reflects, in Switzerland at least, also geographic regions (*e.g.*, 8006 becomes 8000, which reflects the canton of Zurich).
- The gender column and the age group property is combined to a gender age group property (*e.g.*, *f\_25-34*).
- All the events triggered from an account are counted and the resulting number is saved to a new column (*e.g.*, how many times the app was opened by counting the APP\_OPENED event triggered by an account).
- For the on-boarding events the `event_created_at` timestamps were extracted to generate additional properties on the account level (*e.g.*, the on-boarding duration or the duration from registration to the first payment).
- Since it was not possible to get payment amount information from the bank, some additional mocked payment properties were added to the account level: A property was mocked to reflect the amount of money an account invested.
- Because the bank did not allow to pull real investment and payment data into this prototype, additional properties had to be mocked: A field was mocked to show the total invested amount. The mocked amount was calculated from the first three digits of the postal code and the age group.
- From the total invested amount (= total assets under management), the AUM cluster, which an account is part of, is calculated. The AUM clusters were defined as 0-15k, 15k-30k, 30k-45k and so on. Listing 4.14 displays the implementation of the AUM mock and the one of the AUM cluster.

```

1 CAST(LEFT(plz, 2) AS FLOAT) * age_group as aum_mock,
2 FLOOR( CAST(LEFT(plz, 2) AS FLOAT) * age_group / 15000 ) * 15000 as
    aum_group_mock

```

Listing 4.14: The SQL implementation of the AUM mock.

## Google Analytics Data Transformations

Listing 4.15 includes a shortened version of the view that performs the GA data transformations. The Google Analytics data is grouped and aggregated by Pseudo User ID (line 24) to get one row of GA KPIs per user identified by GA. The Pseudo User ID is the ID GA assigns to a user, and it remains the same for events coming from the same user. Since there are usually multiple events per Pseudo User ID, each GA column needs to be aggregated. The following aggregations and additions were implemented.

- String properties are concatenated with a semicolon if there are several different values. These include values that can change during app usage (*e.g.*, device category, device OS, device language) and those that should actually be unique per Pseudo User ID. These include the properties User ID (line 7), which is important for connecting the data and corresponds to the Account ID of the application data, the traffic source name (line 9), which corresponds to the ad campaign name, or the Google Click ID, which can be used to connect the Click View Report.
- To aggregate the event date, event timestamp or other time or integer columns, the SQL MIN-function is used (line 5).
- A APP\_REMOVE\_cnt property is generated which counts how many times the APP\_REMOVE event has been logged, and thus tells if the app has already been uninstalled by the user (lines 11-14).
- Additionally, the timestamps of the FIRST\_OPEN and the APP\_REMOVE events are extracted, so that the app usage duration can be returned (lines 3 and 15-22).

```

1 SELECT event_date,
2        [...]
3        APP_REMOVE_et - FIRST_OPEN_et app_usage_duration
4 FROM (
5         SELECT MIN(event_date) event_date,
6                [...]
7                STRING_AGG(distinct user_id, ';') user_id,
8                user_pseudo_id,
9                STRING_AGG(distinct traffic_source_name, ';')
10               traffic_source_name,
11               [...]
12               (SELECT count(*)
13                FROM t_ga_events
14                WHERE user_pseudo_id = tgae.user_pseudo_id
15                  AND event_name = 'app_remove') APP_REMOVE_cnt,
16               (SELECT MIN(event_timestamp)
17                FROM t_ga_events
18                WHERE user_pseudo_id = tgae.user_pseudo_id
19                  AND event_name = 'first_open') FIRST_OPEN_et,
20               (SELECT MAX(event_timestamp)
21                FROM t_ga_events
22                WHERE user_pseudo_id = tgae.user_pseudo_id
23                  AND event_name = 'app_remove') APP_REMOVE_et
24         FROM t_ga_events tgae
25         GROUP BY user_pseudo_id
26 )
27 WHERE -- only take environments NULL or 'prod'
28        (up_environment IS NULL
29         OR up_environment = 'prod')
30 -- not more than one user_id per user_pseudo_id -> this only happens
31    during development
32 AND user_id NOT LIKE '%%;'

```

Listing 4.15: The SQL statement snippet which performs the GA data transformations.

## Google Ads Data Transformation

The Google Ads data transformation mainly involves connecting the exported reports with each other. The Ad Asset Report contains a reference to the Ad Group Ad. The Ad Group Ad Report references the Ad Group and the Ad Group Report references the Campaign. Additionally, the Campaign Criterion Report references the campaign. The Click View Report contains a reference to both the Campaign and the Ad Group Ad entity (see also the challenges described in Section 5.3). The views that connect the data with each other are simple joins of to database tables. In addition to joining the reports data, the following transformations are performed.

- In order to recognize possible seasonality, the start month and the end month as well as the start quarter and the end quarter of the campaign start and end date are extracted from the Campaign Performance Report.
- The campaign duration is calculated from start and end date and added as a property. Listing 4.16 is the part of the SELECT-clause which adds the campaign duration column.

```
1 CASE
2     WHEN campaign_end_date != '2037-12-30' THEN
3         (campaign_end_date - campaign_start_date) / 30
4     END AS duration_in_month,
```

Listing 4.16: The SQL statement snippet which calculates the campaign duration.

- The Campaign Criterion Report is grouped and aggregated by the associated campaign. The categorical data from multiple rows are transformed into binary columns to represent the different criteria in one row per campaign. An example code snippet is found in Listing 4.17. It shows the generation of the binary `gender_type_female` column.

```
1 COUNT( CASE WHEN campaign_criterion_gender_type = 'FEMALE' AND
2             campaign_criterion_negative = FALSE THEN 1 END )
3 gender_type_female,
```

Listing 4.17: The SQL statement which generates the binary `gender_type_female` column.

- The Ad Group Ads Report is grouped by campaign and the amount of different image ads, video ads, ad descriptions and ad headlines are counted. In addition, for the ad headlines and ad descriptions the average text length was calculated.

### 4.3.6 Stage 3: Connecting the Data from Different Sources

With the data from the different sources being transformed, the data is ready to be connected with each other. The application data is linked to the Google Analytics data using the account ID. In the case of Google Analytics, this corresponds to the user ID that is sent from the front end to Google Analytics after the user has registered. The Google

Analytics data, in turn, is connected to the Google Ads data using the Google Click ID. During the implementation it was noticed that there are very few Google Analytics events with a Google Click ID value, but comparatively many with a `traffic_source_name` field pointing to the campaign name of a Google Ads campaign. This issue is described in more detail in Section 5.3. Due to this inconsistency it was decided to additionally link the GA data directly to the campaign names from the Google Ads data using the `traffic_source_name` field. In this way, all accounts from the Application DB that GA were able to associate with a Google Ads campaign, can still be linked to the Google Ads campaigns and the campaign criterion data.

The view `v_overall_kpis_by_campaigns` in Listing 4.18 is used to connect the GA data with the application data. For the usage by the BI tools only the JOIN statement in lines 33-34 is used, since the BI tools are able to group, aggregate and filter the data on their own. For the prediction system the data is grouped by campaign name (the `traffic_source_name` field) (see line 35) and the application data KPIs defined in Table 3.2, which are used as ML labels later, are generated. For each campaign name the fully onboarded users (lines 3-21) and the users in the on-boarding process (lines 22-24) are counted, segmented by gender (lines 4-6), age group (lines 7-9), device (lines 10-12), postal code thousands (lines 13-15), gender-age-group-pair (lines 16-18) and AUM cluster (lines 19-21). Also the users stuck in the on-boarding process (lines 22-23), the users stuck right before or in the identity verification step (lines 25-26), the paying users (lines 27-28) and the amount of payments (lines 29-30) are counted. The total assets under management generated by each campaign are returned with lines 31-32.

```

1 SELECT
2   traffic_source_name ,
3   -- amount fully onboarded users
4   -- by gender
5   COUNT( CASE WHEN   onboarding_state = 'ON_BOARDED' AND gender ILIKE '
6     female' THEN 1 ELSE NULL END ) as fully_onb_users_f ,
7   [...]
8   -- by age group
9   COUNT( CASE WHEN   onboarding_state = 'ON_BOARDED' AND age_group = 10
10     THEN 1 ELSE NULL END ) as fully_onb_users_age_10s ,
11   [...]
12   -- by device
13   COUNT( CASE WHEN   onboarding_state = 'ON_BOARDED' AND phoneos ILIKE '
14     ios' THEN 1 ELSE NULL END ) as fully_onb_users_ios ,
15   [...]
16   -- by plz 1000s
17   COUNT( CASE WHEN   onboarding_state = 'ON_BOARDED' AND plz_region = '
18     1000' THEN 1 ELSE NULL END ) as fully_onb_users_plz_1000 ,
19   [...]
20   -- by (gender, age_group)-pair
21   COUNT( CASE WHEN   onboarding_state = 'ON_BOARDED' AND gender_age_group
22     ILIKE 'male_10' THEN 1 ELSE NULL END ) as
23     fully_onb_users_age_10s_m ,
24   [...]
25   -- by asset under management cluster (mocked)
26   COUNT( CASE WHEN   onboarding_state = 'ON_BOARDED' AND aum_group_mock <
27     3000 THEN 1 ELSE NULL END ) as fully_onb_users_aum_less_3k ,
28   [...]
29   -- amount users in onboarding process (same categories as for fully

```

```

        onboarded)
23  COUNT( CASE WHEN onboarding_state != 'ON_BOARDED' AND gender ILIKE '
        female' THEN 1 ELSE NULL END ) as registered_users_f,
24  [...]
25  -- amount users before identification
26  COUNT( CASE WHEN onboarding_state = 'IDENTITY_VERIFICATION' THEN 1
        ELSE NULL END ) as users_in_identification,
27  -- amount paying users (both single payments and standing orders)
28  COUNT( CASE WHEN EINZELAUFTRAG_PAID_cnt > 0 OR
        REPEATED_TRANSACTION_cnt > 0 THEN 1 ELSE NULL END ) as paying_users
        ,
29  -- amount payments
30  CAST(SUM( EINZELAUFTRAG_PAID_cnt ) + SUM( REPEATED_TRANSACTION_cnt )
        AS INT) as payments_cnt,
31  -- amount of asset under management
32  CAST(ROUND( SUM( aum_mock ) / 25000 ) * 25000 AS INT) as AUM
33  FROM v_accounts_with_event_kpis ea
34  RIGHT OUTER JOIN v_ga_events gae ON ea.account_id = gae.user_id;
35  GROUP BY traffic_source_name;

```

Listing 4.18: The view `v_overall_kpis_by_campaigns` which connects application data with GA data.

The last and most important view for predicting the campaign outcomes is the `v_parameters_and_kpis_by_campaign` view in Listing 4.19. The view contains the majority of the features that have been identified as relevant (see Table 3.1) and can be extracted from the available data, as well as the labels or KPIs that could be retrieved from the combination of all data (see Table 3.2). The `v_overall_kpis_by_campaigns` view is joined with the view `v_googleads_overall_parameters_by_campaign` (lines 15-16) in which all the information from Google Ads are summarized by campaign. Since after this step, the marketing cost and the marketing revenue are joined in one view, finally the important overall KPIs *customer acquisition costs* (line 13) and the *cost per AUM* (line 14) can be retrieved. This view is the basis for the prediction system. How it is exported and used in the prediction system is described in the next sections.

```

1  SELECT
2  [...]
3  traffic_source_name,
4  fully_onb_users_f,
5  [...]
6  users_in_identification,
7  users_before_identification,
8  paying_users,
9  payments_cnt,
10 AUM,
11 CAST(CASE WHEN paying_users > 0 THEN AUM / paying_users END AS INT) AS
    aum_per_user,
12 CAST(daily_campaign_budget_amount_micros * duration_in_month * 30 AS
    INT) AS campaign_costs,
13 CASE WHEN paying_users > 0 THEN CAST(
    daily_campaign_budget_amount_micros * duration_in_month * 30 AS INT
    ) / paying_users END AS cac_micros,
14 CAST(CASE WHEN AUM > 0 THEN CAST(daily_campaign_budget_amount_micros *
    duration_in_month * 30 AS INT) / AUM END AS INT) AS cost_per_aum
15 FROM v_googleads_overall_parameters_by_campaign param

```

```
16 LEFT OUTER JOIN v_overall_kpis_by_campaigns kpis ON param.
    campaign_name = kpis.traffic_source_name
```

Listing 4.19: The final (before using the data for predictions) view `v_parameters_and_kpis_by_campaign`, which connects application, GA and Google Ads data and calculates important KPIs per campaign.

### 4.3.7 Exporting the Data

The prediction system needs the `v_parameters_and_kpis_by_campaign` view described in the previous section to train the models. To keep the overall system modular, it does not make sense to let the ML code access the DWH view in a cumbersome way, but it is best, as designed, to work with a CSV export and import again. Thus, the requirement was that the view is exported every night to a CSV file. Other required exports can be implemented identically.

An additional Docker Compose service `dwh-cron-job` was set up, whose Dockerfile is based on the latest Ubuntu image. Cron is installed on it. In addition, the Cockroach DB image is installed that `cockroach` commands can be executed. All commands that should be executed regularly are added to a bash script each and made executable. The `dwh-crontab` file, where all *cron jobs* are defined is added to the cron service. The entrypoint of the Dockerfile starts the cron daemon: `ENTRYPOINT ["cron", "-f"]`. Listing 4.20 shows the implemented crontab file. It schedules the export of the application data from the Production DB (lines 1-2), the imports of the CSV files into the DWH (lines 3-7) and the CSV export of the `v_parameters_and_kpis_by_campaign` view from the DWH (line 8). The DWH view is exported every night at 4 AM. *Cron jobs* are defined in UTC time.

```
1 0 0 * * * /csv-export-anonymized-account-data.sh >> /var/log/export-
    anonymized-account-data.log 2>&1
2 0 0 * * * /csv-export-events-data.sh >> /var/log/export-events-data.log
    2>&1
3 0 1 * * * /import_into_t_ga_events.sh >> /var/log/
    import_into_t_ga_events.log 2>&1
4 0 1 * * * /import_into_t_googleads_campaigns.sh >> /var/log/
    import_into_t_googleads_campaigns.log 2>&1
5 0 1 * * * /import_into_t_googleads_campaign_criteria.sh >> /var/log/
    import_into_t_googleads_campaign_criteria.log 2>&1
6 0 1 * * * /import_into_t_googleads_adgroupads.sh >> /var/log/
    import_into_t_googleads_adgroupads.log 2>&1
7 0 1 * * * /import_into_t_googleads_clickview.sh >> /var/log/
    import_into_t_googleads_clickview.log 2>&1
8 0 2 * * * /csv-export-overall-view.sh >> /var/log/export-overall-view.
    log 2>&1
```

Listing 4.20: The crontab file which schedules the CSV file exports and imports.

## 4.4 Prediction System

This section describes the prediction system prototype that was implemented. First, it explains how the model training works, how the continues learning is achieved and how the models are deployed as prediction API endpoints.

### 4.4.1 Model and Training Code Definition

The machine learning code is written using TensorFlow in Python. Using the classic Keras [66] of TensorFlow, a regression problem was implemented so that the models can be trained with a linear regression algorithm. Additionally, the linear model was extended with two hidden non-linear layers to a deep neural network model. Besides that, both models are identical. With the TensorFlow Decision Forests sub-framework, also using Keras [66], code for training two additional models was implemented. Random Forest and Gradient Boosted Trees models can be trained with this code.

#### Regression Models

In Listing 4.21, a section of the regression model training code is shown in a simplified and abbreviated form. The program code goes through the following steps.

1. The data is read from the CSV file in line 1.
2. Line 2 gets the model version number from the environment variable. This is important if the models should not be overwritten when a new training is done.
3. The labels for which models should be trained are defined in lines 3-6. For this prototype, only seven of the labels defined in the design chapter were used for the model training. For every label defined in this array, the same code starting at line 8 is executed.
4. The data set is copied (line 8) and with one-hot encoding the categorical columns are converted into binary ones (line 10).
5. In lines 12-25, the data set is split into a train and test data set (lines 12-13) and the current label is selected and removed from the train feature and the test feature data set (lines 18-19). The labels trained in the other iterations need to be removed as well (lines 21-25).
6. A normalization of the data ranges is prepared in lines 27-28. It is generally advised to do normalization of the data ranges [6].
7. The normalizer layer is added to the model definition in lines 29-32. For regression models Keras' Sequential model is used.



8. The linear model is compiled (lines 34-37): An optimizer is set and the loss function is defined. An optional metrics array is passed.
9. The model training is executed with `model.fit` at line 39 and the training history is saved to the `linear_history` variable to plot the training process later.
10. The test results are saved in memory to later display a comparison with the DNN model (lines 45-46) and the trained model for this label is saved to the file system (line 49).
11. The training history is plotted in lines 51-60 and the diagram is saved to the file system too (line 59).

For the DNN model, the same train and test feature data sets are used. The model definition and compilation as well as the training, saving and plotting process works identically. Lines 62-67 shows the only difference between the two models: While still using the Sequential model from Keras and the same normalization layer, two additional hidden, Dense layers using the non-linear activation function `relu` (Rectified Linear Unit) are used instead of a single linear layer.

The code obviously cannot be used for a productive system, because with the for-loop the same model training parameters are used for each label. The code is intended to show that automated training with the same DB view is possible and that the models and the model training are all set up the same way. The for-loop must of course be broken down into individual sequential pieces of code (or even individual Python modules) so that the parameters can be tweaked for each model individually.

```

1 raw_dataset = pd.read_csv("/data/v_parameters_and_kpis_by_campaign.csv")
2 MODEL_VERSION = os.getenv('MODEL_VERSION', '1')
3 LABELS = ['fully_onb_users_f', 'fully_onb_users_m', 'paying_users', 'aum',
4           ', 'aum_per_user', 'cac_micros', 'cost_per_aum',
5           #'fully_onb_users_age_10s',
6           [...]]
7 for label in LABELS:
8     dataset = raw_dataset.copy()
9
10    dataset = pd.get_dummies(dataset, columns=['
11        campaign_advertising_channel_type', ...], prefix='', prefix_sep='')
12
13    train_dataset = dataset.sample(frac=0.8, random_state=0)
14    test_dataset = dataset.drop(train_dataset.index)
15
16    train_features = train_dataset.copy()
17    test_features = test_dataset.copy()
18
19    train_labels = train_features.pop(label)
20    test_labels = test_features.pop(label)
21
22    nonFeatures = LABELS.copy()
23    nonFeatures.remove(label)
24    for nonFeature in nonFeatures:
25        train_features.pop(nonFeature)

```

```

25     test_features.pop(nonFeature)
26
27     normalizer = preprocessing.Normalization(axis=-1)
28     normalizer.adapt(np.array(train_features))
29     linear_model = tf.keras.Sequential([
30         normalizer,
31         layers.Dense(units=1)
32     ])
33
34     linear_model.compile(
35         optimizer=tf.optimizers.Adam(learning_rate=0.1),
36         loss='mean_absolute_error',
37         metrics=["mse"])
38
39     linear_history = linear_model.fit(
40         train_features, train_labels,
41         epochs=100,
42         verbose=0,
43         validation_split = 0.2)
44
45     test_results = {}
46     test_results['linear_model'] = linear_model.evaluate(
47         test_features, test_labels, verbose=0)
48
49     linear_model.save('./models/linear_regression/' + label + '/' +
50                     MODEL_VERSION)
51
52     fig = plt.figure(figsize=(12,8))
53     plt.plot(linear_history.history['mse'])
54     plt.plot(linear_history.history['val_mse'])
55     plt.title('model loss')
56     plt.ylabel('MSE')
57     plt.xlabel('Epoch')
58     plt.legend(['mse', 'val_mse'], loc='upper left')
59     plt.show()
60     fig.savefig('./models/linear_regression/linear_reg_mse_' + label + '_' +
61               + MODEL_VERSION + '.jpg')
62     plt.close(fig)
63
64     dnn_model = keras.Sequential([
65         normalizer,
66         layers.Dense(64, activation='relu'),
67         layers.Dense(64, activation='relu'),
68         layers.Dense(1)
69     ])
70     [...]
71     print("Results:")
72     print(pd.DataFrame(test_results, index=['MAE', 'MSE']).T)

```

Listing 4.21: The code which trains the linear regression and the DNN-Regression model.

## Decision Forest Models

In Listing 4.22 the abbreviated code for the decision forest models is shown. For the Random Forest and the Gradient Boosted Trees Model the TensorFlow Decision forest

package must be imported (line 1). In this package, Keras is also included, but the implementation of Keras is slightly different. For the Decision Forest models, similar program steps to the ones from the regression models are executed.

1. The data set is imported.
2. The model version and the model labels to be trained are defined as environment variable and in the LABELS array (lines 2-3).
3. For this framework, the features must be defined individually if not all columns are needed as features (lines 4-7).
4. For each label the data set is copied and split into a train and test data frame and converted from a data frame to a data set (lines 15-17). The label and the task type are being defined at lines 16 and 17.
5. At line 19, the model is configured as a regression task with the defined features and compiled with an optional metrics array (line 20).
6. The model training is executed at line 21.
7. Using similar functions as in the regression code, the models are evaluated, the results are cached and the model is stored to the file system.
8. With a training logs inspector the logs can be retrieved and a diagram can be drawn.

The code for the Gradient Boosted Trees models is identical with the exception that the model at Line 5 is configured with `tfdf.keras.GradientBoostedTreesModel`. Also plotting training logs is not possible for this model.

```

1 import tensorflow_decision_forests as tfdf
2 [...]
3 LABELS = ['fully_onb_users_f', ...]
4 all_features = [
5     tfdf.keras.FeatureUsage(name="campaign_advertising_channel_type"),
6     tfdf.keras.FeatureUsage(name="campaign_advertising_channel_sub_type"),
7     [...]]
8
9 def split_dataset(dataset, test_ratio=0.30):
10     test_indices = np.random.rand(len(dataset)) < test_ratio
11     return dataset[~test_indices], dataset[test_indices]
12
13 for label in LABELS:
14     dataset_df = raw_dataset.copy()
15     train_ds_pd, test_ds_pd = split_dataset(dataset_df, 0.30)
16     train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_ds_pd, label=
17         label, task=tfdf.keras.Task.REGRESSION)
18     test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test_ds_pd, label=
19         label, task=tfdf.keras.Task.REGRESSION)
20
21     model_1 = tfdf.keras.RandomForestModel(task = tfdf.keras.Task.
22         REGRESSION, features=all_features, exclude_non_specified_features=
23         True, num_trees=300)

```

```

20  model_1.compile(metrics=["mse", "mae"])
21  model_1.fit(x=train_ds)
22  [...]

```

Listing 4.22: The code which trains the random forest and the gradient boosted trees model.

## 4.4.2 Synthetic Data / Oversampling the Data

During the implementation of Stage 2 of the DWH (see Section 4.3.5), it was found out that machine learning cannot be performed on ad level since the traffic source can only be gathered on the campaign level and therefore it is not possible to figure out which mobile app user clicked on which ad before installing the app (see also Section 5.3). Even if it was possible to gather the data on the ad click level, it is not sure that it would be a sufficient data volume to perform the machine learning on it. Since April 2021 only three real campaigns were run with Google Ads. 895 out of 1510 created accounts can be associated with one of the three campaigns. The calculation of KPIs is very good possible, but nevertheless, the machine learning ends up with three examples of campaign parameters connected with KPIs as input data. This is the reason why synthetic data was needed.

Additional data was generated in two ways. Firstly, a Generative Adversarial Network (GAN) algorithm for structured data, *Tabular GAN* [23] [5], was used to generate data automatically and secondly, since *Tabular GAN* only works with a minimum of 10 training and 10 test examples, the data was synthesized manually. For the manual approach the three examples were copied and the parameters and KPI were adjusted by hand and mostly in a way that seemed realistic. However, care was taken to ensure that unexpected data and outliers were also present.

Listing 4.23 shows the code developed to synthesize the data with the Tabular GAN generator. This method could be called between lines 10 and 12 in the regression model code (see Listing 4.21) and between lines 14 and 15 in the decision forests model code (see Listing 4.22). First, the data is sampled into train and test features for the GAN algorithm (lines 4-5) and the label on which the algorithm is to be optimized is deleted from the test and training data set (lines 6-7). In line 9 the `GANGenerator` is initialized and then, in line 17, the pipeline is run. Most of the values are the default ones. Important is line 10, where the multiplier (how many examples should be generated) is passed, and lines 18-21, where the train features, the train labels and the test features are passed. Also line 15 is adjusted to disable the post processing because the validation fails when there are so few records available. Lines 24-25 appends the test data back to the `new_train` data set so no original data is lost. Line 26 merges the `new_labels` with the `new_train` data resulting in one data set again, which is returned in line 28. Line 27 saves the synthesized data as CSV file.

```

1  #!pip install -q tabgan
2  from tabgan.sampler import GANGenerator
3  def synthesize_data(dataset, label, gen_x_times, train_frac=0.2):
4      gan_train_features = dataset.sample(frac=train_frac, random_state=0)

```

```

5     gan_test_features = dataset.drop(gan_train_features.index)
6     gan_train_labels = gan_train_features.pop(label)
7     gan_test_labels = gan_test_features.pop(label)
8
9     new_train, new_labels = GANGenerator(
10         gen_x_times=gen_x_times,
11         cat_cols=None, bot_filter_quantile=0.001, top_filter_quantile
12             =0.999,
13         is_post_process=False, # default is True
14         adversarial_model_params={[...]},
15         pregeneration_frac=2,
16         only_generated_data=False,
17         epochs=500
18     ).generate_data_pipe(
19         gan_train_features,
20         gan_train_labels.to_frame(),
21         # a minimum of 10 test features are needed
22         gan_test_features,
23         deep_copy=True, only_adversarial=False, use_adversarial=True)
24
25     new_train = new_train.append(gan_test_features, True, True)
26     new_labels = new_labels.append(gan_test_labels, True, True)
27     new_train.insert(len(new_train.columns), label, new_labels)
28     new_train.to_csv(BUCKET_NAME + "/data/" + nowString + "
29         _synthesized_data_" + label + ".csv")
30     return new_train

```

Listing 4.23: The code to call the Tabular GAN generator to synthesize the data to more examples.

### 4.4.3 Model Training

This section describes how the execution of the previously described ML code works. Two approaches were implemented to compare them with each other. First, the models were trained on the local machine and exposed as API endpoints and second, Google Cloud's Vertex AI was used. The model training code was dockerized to run on any machine and environment. The base image used is `tensorflow/tensorflow:2.6.0`. The decision forest models need the decision forest TensorFlow extension which can be installed in the Docker container using `pip3`. The ML code is added to the container and the entry point runs both python programs: `ENTRYPOINT ["/bin/sh","-c", "python -m trainer.regression && python -m trainer.decision_forest"]`

#### Self-hosted Training

To train the model once, only the Docker container needs to be booted. The following command (Listing 4.24) builds and starts the container, reads the DWH view from the `data_transfer` folder and saves the models and the training log visualizations on the host machine (in the `models` folder).

```

1 docker build ./ -t prediction-models && \
2 docker run -v "$(pwd)/models:/models" \
3   -v "$(pwd)/data_transfer:/data" \
4   prediction-models

```

Listing 4.24: The Docker command to build and run the model training on a local machine.

For the weekly automated training of new models, the previously mentioned Docker container has been extended. Additionally, cron is installed, with `crontab /etc/cron.d/ml-crontab` the `ml-crontab` file is added and, instead of the entry point above, it is now `ENTRYPOINT ["cron", "-f"]`. The crontab entry looks as follows (Listing 4.25). Every Sunday at 4:30 AM, first, the environment variables are set and then the two python modules are executed. Since the DWH view is defined to be exported every night (see Listing 4.20), every week, the new data is automatically used to train the models.

```

1 30 4 * * 0 export PATH=/opt/conda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin; /bin/sh -c "cd / && python -m trainer.regression && python -m trainer.decision_forest" >> /var/log/model-training.log 2>&1

```

Listing 4.25: The models are freshly trained every week.

## Training with Vertex AI

The model training with Vertex AI was implemented as follows. The CSV file of the input data was uploaded to the GCS bucket. The lines where the CSV files are read (line 1 in Listing 4.21) were changed to point to the GCS bucket path. The same was done with the paths for the model saving, so that the trained models are also stored in the bucket instead of the local machine. The generating and saving of the diagrams was disabled for Vertex AI training and the comparison of the two approaches. The rest of the ML code remained the same. All changes can be implemented based on program arguments or environment variables, so no different Docker images have to be used for the local machine and the Vertex AI training.

The Docker image was rebuilt and pushed to the Google Container Registry. This requires authentication with the Google Cloud (preferably with service account credentials). After the Docker image was successfully uploaded, a Vertex AI training instance could be created: In the UI Console, a *Custom Training* with *no managed dataset* was configured. The uploaded Docker image was selected as the *Custom [training] container*. As a final step, the machine type to train on was selected (the cheapest *Standard* machine, which has 4 vCPUs and 15 GiB memory allocated to it) and the training was started. The training was finished after 13 minutes and 6 seconds and the models were successfully saved to the GCS bucket. The models can now be deployed as Prediction API endpoints, whether using Vertex AI or a self-hosted architecture (see Section 4.4.4).

Figure 4.3 shows the configuration and the stats of the last Vertex AI model training with a custom Docker image uploaded to the GCR at `gcr.io/<GLOUD_PRODJECT_ID>/ml/v1`.

The automated weekly recurring training with Vertex AI has not been implemented. The implementation of a GCF connected to Cloud Scheduler can be studied in Section 4.3.2.

Vertex AI	← campaign-predictions																														
<ul style="list-style-type: none"> <li>Dashboard</li> <li>Datasets</li> <li>Features</li> <li>Labeling tasks</li> <li>Notebooks</li> <li>Pipelines</li> <li><b>Training</b></li> <li>Experiments</li> <li>Models</li> <li>Endpoints</li> <li>Batch predictions</li> <li>Metadata</li> </ul>	<div> <p><b>Training pipeline was completed on Sep 21, 2021, 9:52:01 AM.</b></p> <table> <tr><td>Status</td><td>Succeeded</td></tr> <tr><td>Training pipeline ID</td><td>5551830032827023360</td></tr> <tr><td>Created</td><td>Sep 21, 2021, 9:38:55 AM</td></tr> <tr><td>Start time</td><td>Sep 21, 2021, 9:38:55 AM</td></tr> <tr><td>Elapsed time</td><td>13 min 6 sec</td></tr> <tr><td>Region</td><td>europa-west1</td></tr> <tr><td>Encryption type</td><td>Google-managed key</td></tr> <tr><td>Custom job</td><td><a href="#">1116857523215794176</a></td></tr> </table> <table> <tr><td>Machine type (Worker pool 0)</td><td>n1-standard-4</td></tr> <tr><td>Machine count (Worker pool 0)</td><td>1</td></tr> <tr><td>Container Location (Worker pool 0)</td><td>gcr.io/.../ml/v1@sha256:f22a5ct</td></tr> </table> <table> <tr><td>Dataset</td><td>No managed dataset</td></tr> </table> <table> <tr><td>Algorithm</td><td>Custom training</td></tr> <tr><td>Objective</td><td>Custom</td></tr> <tr><td>Container (Training)</td><td>Custom</td></tr> </table> </div>	Status	Succeeded	Training pipeline ID	5551830032827023360	Created	Sep 21, 2021, 9:38:55 AM	Start time	Sep 21, 2021, 9:38:55 AM	Elapsed time	13 min 6 sec	Region	europa-west1	Encryption type	Google-managed key	Custom job	<a href="#">1116857523215794176</a>	Machine type (Worker pool 0)	n1-standard-4	Machine count (Worker pool 0)	1	Container Location (Worker pool 0)	gcr.io/.../ml/v1@sha256:f22a5ct	Dataset	No managed dataset	Algorithm	Custom training	Objective	Custom	Container (Training)	Custom
Status	Succeeded																														
Training pipeline ID	5551830032827023360																														
Created	Sep 21, 2021, 9:38:55 AM																														
Start time	Sep 21, 2021, 9:38:55 AM																														
Elapsed time	13 min 6 sec																														
Region	europa-west1																														
Encryption type	Google-managed key																														
Custom job	<a href="#">1116857523215794176</a>																														
Machine type (Worker pool 0)	n1-standard-4																														
Machine count (Worker pool 0)	1																														
Container Location (Worker pool 0)	gcr.io/.../ml/v1@sha256:f22a5ct																														
Dataset	No managed dataset																														
Algorithm	Custom training																														
Objective	Custom																														
Container (Training)	Custom																														

Figure 4.3: Configuration and Stats of the last Vertex AI Model Training with a custom Docker image

Only the content of the GCF would be slightly different, triggering a Vertex AI training instead of a Big Query export.

#### 4.4.4 Model Serving

The model serving was also implemented with the two approaches: once self-hosted and once with Vertex AI.

##### Self-hosted Prediction API

To achieve the fully automated continues-learning chain, the newly trained models need to be re-deployed in an automated way. The deployment of the models is enabled by TensorFlow-Serving [84]. A suitable TensorFlow-Serving Docker image can be downloaded from Docker Hub using `docker pull tensorflow/serving`. However, this only works with the standard TensorFlow and not (yet) with the TensorFlow Decision Forests framework. Fortunately, the team of ML6 [72] has implemented a fork that closes this gap. Their fork's Docker image can be downloaded using `docker pull ml6team/tf-serving-tfddf`.

With one command, the models can be hosted as prediction endpoints. What is needed is just a configuration file that references all the models which want to be deployed. Listing 4.26 contains a subset of the configuration file. For each label that wants to be predicted, there must exist a configuration object. The command in Listing 4.27 deploys the configured models as an API on port 8501 of the local machine. The flag `--file_system_poll_wait _seconds=604800` in line 6 causes the service to check the



file system for new model versions every seven days. Thus, the newly trained models are also provided automatically.

The prediction API is then accessible at `http://localhost:8501/v1/models/`. For example, an AUM prediction can be requested with a POST request to `http://localhost:8501/v1/models/linear_regression_aum:predict`.

```

1 model_config_list {
2   config {
3     name: 'linear_regression_aum'
4     base_path: '/models/linear_regression/aum'
5     model_platform: 'tensorflow'
6   }
7   config {
8     name: 'linear_regression_aum_per_user'
9     base_path: '/models/linear_regression/aum_per_user'
10    model_platform: 'tensorflow'
11  }
12  config {
13    name: 'linear_regression_cac_micros'
14    base_path: '/models/linear_regression/cac_micros'
15    model_platform: 'tensorflow'
16  }
17  [...]
```

Listing 4.26: The configuration file for serving the trained models with TensorFlow-Serving [84].

```

1 docker run --name serve-ml-models \
2   -p 8501:8501 \
3   -v "$(pwd)/models/:/models/" \
4   ml6team/tf-serving-tfdd \
5   --model_config_file=/models/models.config \
6   --file_system_poll_wait_seconds=604800 &
```

Listing 4.27: The command which serves all models as prediction endpoints and renews the models every week.

## Vertex AI Prediction Endpoints

The import and model hosting at Vertex AI was done, like the training configuration, via the Vertex UI Console. In the console, under the tab *Models*, the functionality *Import* was selected. To *import model artifacts into a new pre-built container* the model framework (*TensorFlow*) and the used version (*2.6*) had to be chosen. After that, the model artifacts saved previously to the GCS bucket were selected. For all other options, the default values were kept.

After successfully importing the model, it could be deployed to the first model endpoint by again choosing the cheapest machine available (2 vCPUs and 7.5 GiB memory) and keeping the default parameters for all other configuration options. This needs to be done for every model that should be exposed as an API endpoint. After deploying an endpoint, Vertex proposes sample requests to get the predictions via a CURL statement. For getting a prediction, the client needs to be authenticated with the Google Cloud.



## 4.5 Prediction Front-End

The predictions can be obtained with simple HTTP(S) requests to the host, mentioned in Section 4.4.4. CURL sample commands to obtain the predictions were created for both trained model types, because they are not identical. Listing 4.28 and Listing 4.29 show that the request body for the linear regression model looks different from that for the random forest model. For the random forest model request, an *instances* array of objects must be passed with the feature names as properties, while for the linear regression model request, the feature values in a numbers array are filled into the *instances* array. The requests return a response object with a *predictions* array (see how it is parsed in Listing 4.30).

```
1 curl -d '{"instances": [
2     [ 1.97,
3       1,
4       [...]]
5   ]}' \
6 -X POST http://localhost:8501/v1/models/
7     linear_regression_fully_onb_users_f:predict
```

Listing 4.28: CURL Command to get predictions from a served regression model.

```
1 curl -d '{"instances": [{
2     "duration_in_month": [1.97],
3     "start_month": [1],
4     [...]]
5   }]' \
6 -X POST http://localhost:8501/v1/models/random_forest_fully_onb_users_f:
7     predict
```

Listing 4.29: CURL Command to get predictions from a served decision forests model.

The admin dashboard front-end was extended with the prediction functionality. An additional sidebar entry, *Campaigning*, was added. On the *Campaign Predictions* page, the marketing specialist can enter the parameters of the planned campaign (see Figure 4.4 and Figure 4.5) and by clicking *Predict Outcome*, the predictions for the five KPIs (*no. of fully onboarded female users*, *no. of fully onboarded male users*, *no. of paying users*, *total amount of assets under management* and *amount of AUM per user*) are returned. In addition, the campaign costs are calculated from the specified parameters and, in combination with the retrieved KPIs, the predicted CAC and the costs per AUM are calculated and displayed. The prediction results can be seen in Figure 4.6.

Listing 4.30 shows a code snippet of the prediction request from the front-end implemented in React.

```
1 const onSubmit = useCallback(async () => {
2     setSubmitting(true);
3     try {
4         const stringifiedBody = JSON.stringify({
5             instances: [
6                 {
```

## Campaign Predictions

- Übersicht
- App-Inhalte
- App-Einstellungen
- Referrals
- Users
- Postcard
- Export
- Campaigning
- Predictions

Channel Type  
Multi Channel

Channel Sub Type  
App Campaign

App Store  
Google App Store

Bidding Strategy  
Optimize towards in app conversion

Campaign Duration  
01.11.2021 → End Date

November 2021

Dezember 2021

Cost per Action Target (in CHF)  
3

No. of Ad Headlines  
3

Figure 4.4: Campaign Predictions Page of the Prediction Front-End, the Campaign Parameter Input Fields, Part 1

The screenshot displays the 'Campaign Predictions' page of a web application. On the left is a dark blue sidebar with navigation links: Übersicht, App-Inhalte, App-Einstellungen, Referrals, Users, Postcard, Export, Campaigning, and Predictions (highlighted). At the bottom of the sidebar is a 'Logout' button. The main content area has a date range selector at the top set to '01.11.2021 → 31.12.2021'. Below this are four language selection options: 'Language German' (checked), 'Language English' (checked), 'Language French' (unchecked), and 'Language Italian' (unchecked). A series of input fields follow, each with a label, a value, and a dropdown arrow: 'Daily Budget (in CHF)' with value '500', 'Cost per Action Target (in CHF)' with value '3', 'No. of Ad Headlines' with value '3', 'Avg. Length of Ad Headlines' with value '28', 'No. of Ad Descriptions' with value '2', 'Avg. Length of Ad Descriptions' with value '78', 'No. of Ad Images' with value '4', and 'No. of Ad Videos' with value '2'. At the bottom of the main area is a large dark blue button labeled 'Predict Outcome'.

01.11.2021 → 31.12.2021

Language German ☒

Language English ☒

Language French ☐

Language Italian ☐

Daily Budget (in CHF)  
500

Cost per Action Target (in CHF)  
3

No. of Ad Headlines  
3

Avg. Length of Ad Headlines  
28

No. of Ad Descriptions  
2

Avg. Length of Ad Descriptions  
78

No. of Ad Images  
4

No. of Ad Videos  
2

**Predict Outcome**

Figure 4.5: Campaign Predictions Page of the Prediction Front-End, the Campaign Parameter Input Fields, Part 2

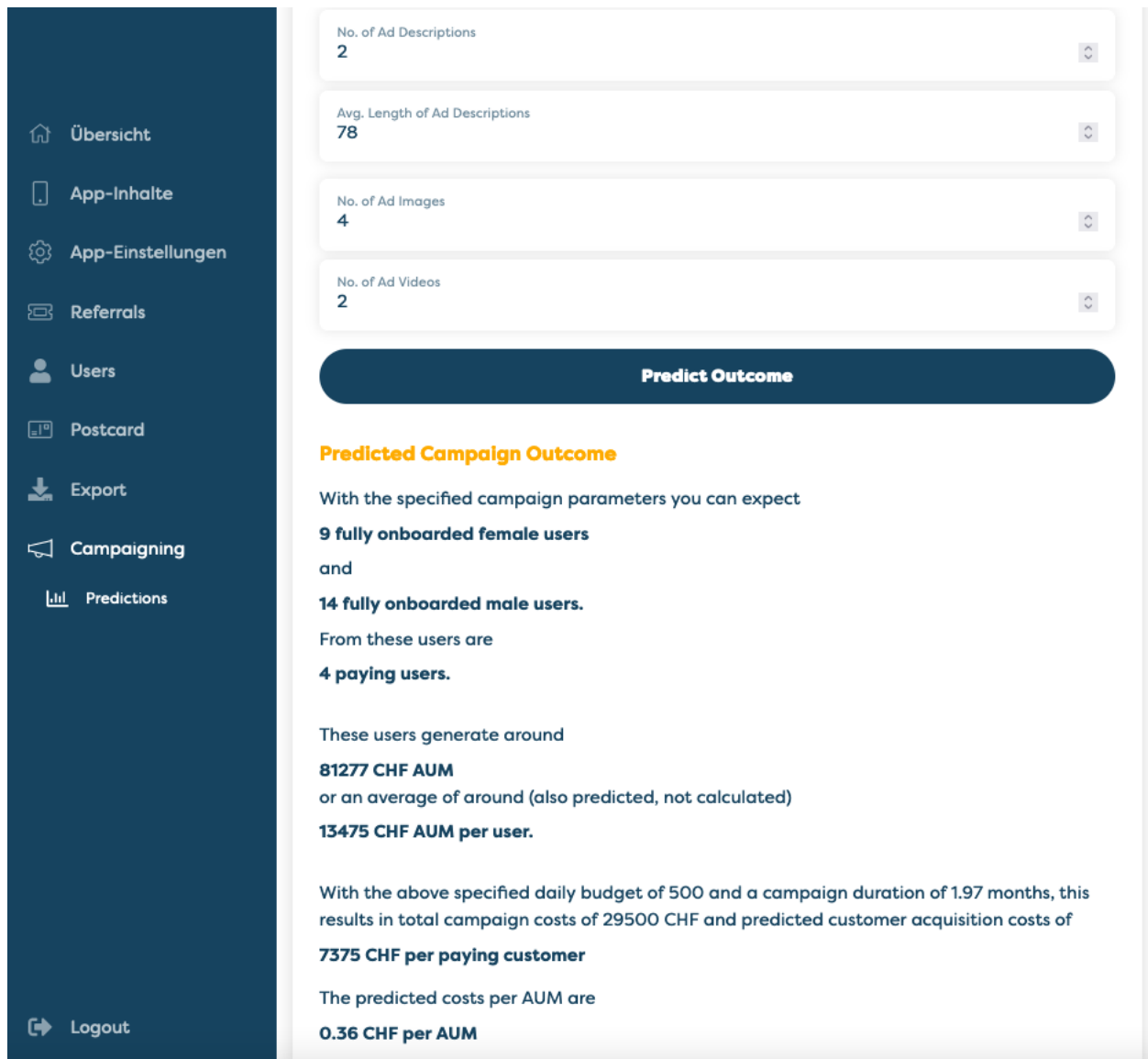


Figure 4.6: Campaign Predictions Page with the retrieved Prediction Results and the calculated KPIs

```

7      [...]
8      campaign_target_cpa_target_cpa_micros: [(cpaTarget ?? 0) *
          1000000],
9      daily_campaign_budget_amount_micros: [(dailyBudget ?? 0) *
          1000000],
10     start_month: [Number(dateRange?.start?.format('M'))],
11     end_month: [Number(dateRange?.end?.format('M'))],
12     duration_in_month: [
13         dateRange?.end?.diff(dateRange?.start, 'months', true)
14     ],
15     language_constant_code_de: [languageDe ? 1 : 0],
16     [...]
17 }
18 ]
19 });
20 let predictions = await makeRequestToOtherDomain(
21     'http://localhost:8501/v1/models/random_forest_fully_onb_users_f:
        predict',
22     HTTP_REQUEST_METHODS.POST,
23     stringifiedBody
24 );
25 setFullOnbUsersF(predictions.predictions[0][0]);

```

Listing 4.30: Code Snippet of the Prediction Request from the React front-end

## 4.6 BI Tool

Google Data Studio and Tableau were tested as BI tools. While the Tableau Desktop application is able to connect directly to the DWH Cockroach DB on the local machine, Google Data Studio required a CSV export. Since Data Studio is a browser app, connecting to the data on the local machine is not possible. However, if the DWH was accessible on a unique domain, connecting would not be a problem either.

Various visualizations can be made with the connected data from the different sources. To illustrate this, two charts from the Google Data Studio BI tool are shown in Figures 4.7 and 4.8. Figure 4.7 shows a dashboard with a lot of information. Among other things, it shows in which on-boarding steps how many users from the campaigns got stuck. Figure 4.8 shows all fully on-boarded users segmented by the campaign from which the users arrived and by the (newly generated) gender-age-group cluster. From the graph, it can be read that the best performing campaign is *App-promotion\_Android\_In-App-Registered*. This is a Google Ads Android campaign that optimizes towards the Registered event instead of the Install event. In addition, it can be seen that age group 25-34 perform well for both genders, with males 35-44 years old on-boarding the most.

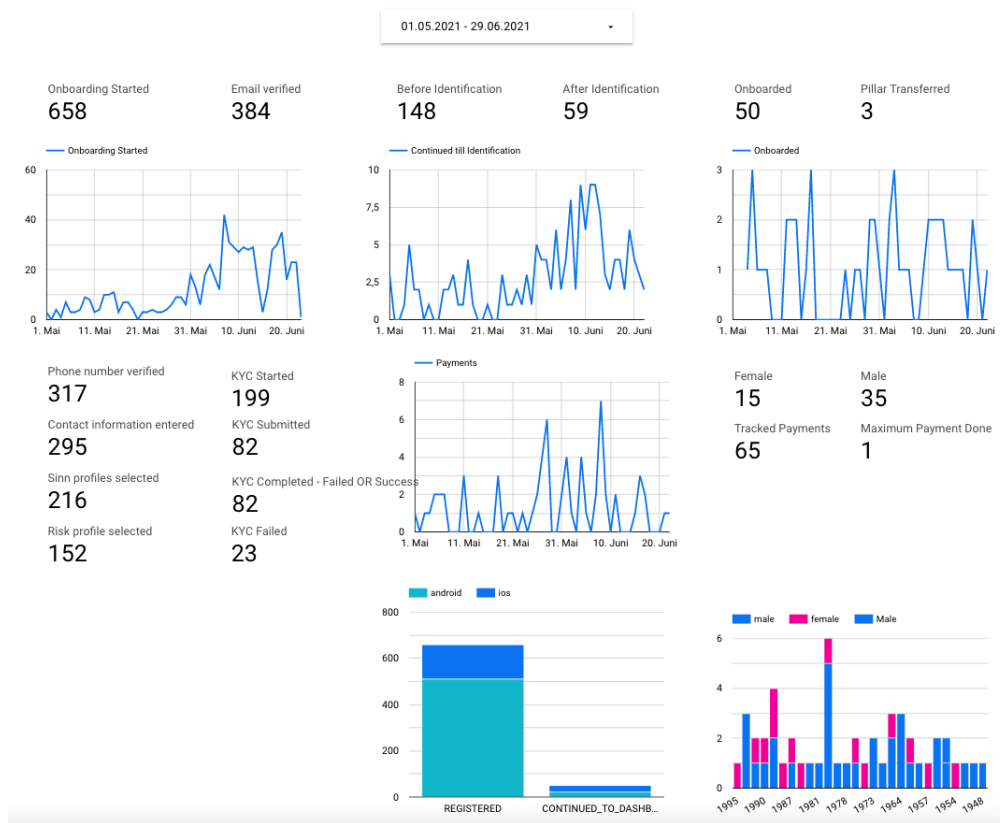


Figure 4.7: BI Dashboard built in Google Data Studio

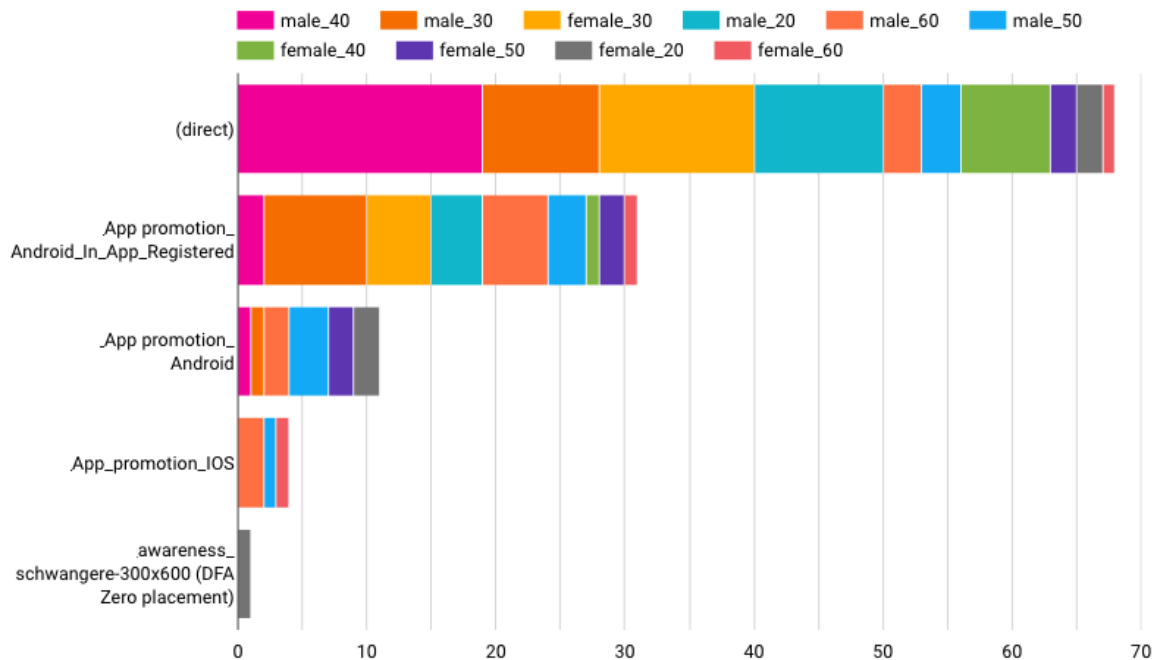


Figure 4.8: BI Chart that visualizes the fully on-boarded users split by traffic source and gender-age-group

# Chapter 5

## Evaluation & Discussion

This chapter first presents the results of the machine learning and the prediction system cost analysis. Then, the challenges of implementing such an architecture are discussed.

### 5.1 Results of ML

The results are divided into two categories. The results of the regression models are described first, followed by the results of the decision forests models.

#### 5.1.1 Regression Models

The models were trained with different parameters and the outcomes were compared with each other. Each model was trained with 50, 100, 150, 200, 250 and 300 epochs. As shown in the implementation section, the MAE was defined as loss function. As evaluation metrics, the Mean Squared Error (MSE) was additionally calculated as well as the mean of the training labels in order to show the percentage deviation of the MAE. The validation split was always set to 20%. No further tweaking of the parameters was performed, as optimizing the models makes little sense with such a large amount of fake data. However, this should be done when the models can be trained with more real data.

The models were trained once with only 24 examples (resp. with only manually synthesised data) and once the 24 examples were extended to 136 examples with the help of the synthesis with the GAN generator (see Section 4.4.2). The 24 examples are part of the 136. Table 5.1 lists the results of the training with only 24 rows and Table 5.2 shows the evaluation of the models with the automatically synthesized data. It can be seen that especially the deep neural network performs better with the 24 examples. But also the linear model can handle the few 24 examples at least as good as the generated data. All DNN models perform better than the linear regression models. To be able to compare the individual models with each other, the deviation was calculated with the MAE and the mean of the training data. The best performing model is the DNN model to predict the

Table 5.1: ML Results of training the Regression models with 24 examples

Model Identifier	Nr. of Epochs	MAE	MSE	Mean	Deviation (Mean / MAE)
linear_model-fully_onb_users_f	125	9.093	101.229	11.053	0.823
dnn_model-fully_onb_users_f	125	<b>3.804</b>	30.648	11.053	<b>0.344</b>
linear_model-fully_onb_users_m	125	17.957	512.088	17.421	1.031
dnn_model-fully_onb_users_m	125	9.466	144.605	17.421	0.543
linear_model-paying_users	125	3.556	15.988	6.105	0.582
dnn_model-paying_users	50	<b>1.759</b>	4.665	6.105	<b>0.288</b>
linear_model-aum	150	6.180	58.562	8.195	0.754
dnn_model-aum	50	<b>2.225</b>	8.674	8.195	<b>0.272</b>
linear_model-aum_per_user	125	1.453	2.522	1.481	0.981
dnn_model-aum_per_user	125	0.943	1.195	1.481	0.637

Table 5.2: ML Results of training the Regression models with GAN-synthesized data (136 examples)

Model Identifier	Nr. of Epochs	MAE	MSE	Mean	Deviation (Mean / MAE)
linear_model-fully_onb_users_f	200	8.479	131.594	6.661	1.273
dnn_model-fully_onb_users_f	200	11.772	209.994	6.661	1.767
linear_model-fully_onb_users_m	200	14.167	349.341	20.266	0.699
dnn_model-fully_onb_users_m	200	13.801	312.981	20.266	0.681
linear_model-paying_users	100	4.588	39.924	5.321	0.862
dnn_model-paying_users	100	4.927	36.955	5.321	0.926
linear_model-aum	100	8.245	101.720	12.031	0.685
dnn_model-aum	100	7.584	94.993	12.031	0.630
linear_model-aum_per_user	100	1.886	7.087	1.625	1.161
dnn_model-aum_per_user	100	2.325	11.713	1.625	1.431

AUM generated by a campaign. Followed by the number of paying users model and the model to predict the number of female fully on-boarded users. The evaluated MAE of the best model is 2.225, which corresponds to a deviation from the mean of the training data of 27.2%. To put this in perspective: With a true value of 82'000 AUM, the predicted value is on average 59'696 or 104'304. The other DNN models performed with a deviation between 28.8% and 75.4%.

Figures 5.1 and 5.2 show the visualized training of the best two performing DNN models. In addition, Figure 5.3 visualizes the training of the best linear regression model. Looking at the visualized training is important to detect over-fitting and to improve the model performance in general. For example, the DNN AUM model could be tweaked to the model with the best evaluation, as it was found out that the model was strongly over-fitted after the 50th epoch. Figure 5.4 shows the visualized training of this model with 125 epochs. The over-fitting is clearly visible. Therefore, the training was reduced to 50 epochs and the performance could be improved.



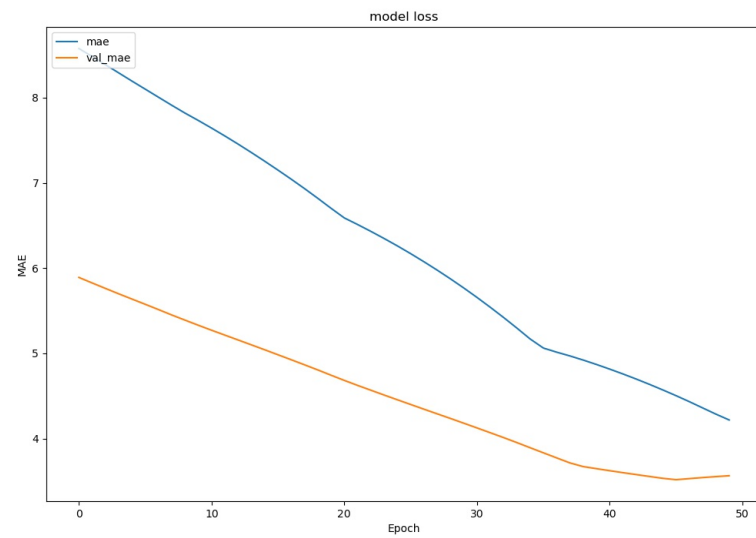


Figure 5.1: Training Visualization of the DNN-Model for the AUM prediction

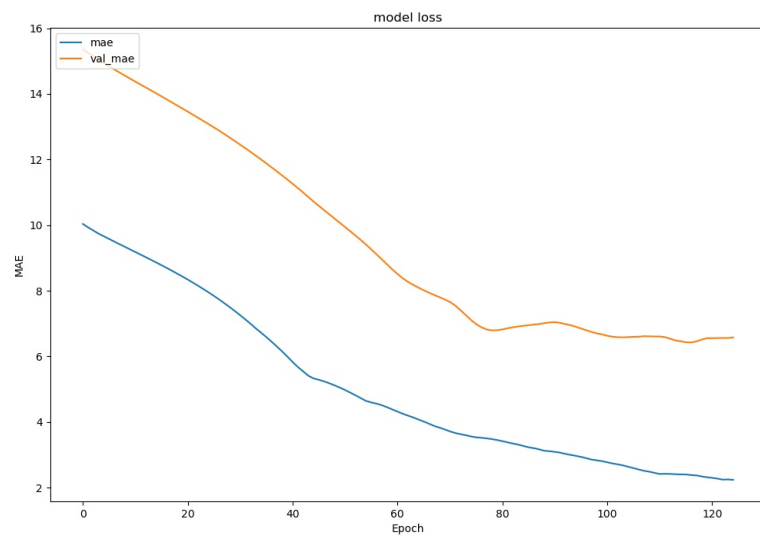


Figure 5.2: Training Visualization of the DNN-Model for the fully on-boarded female users prediction

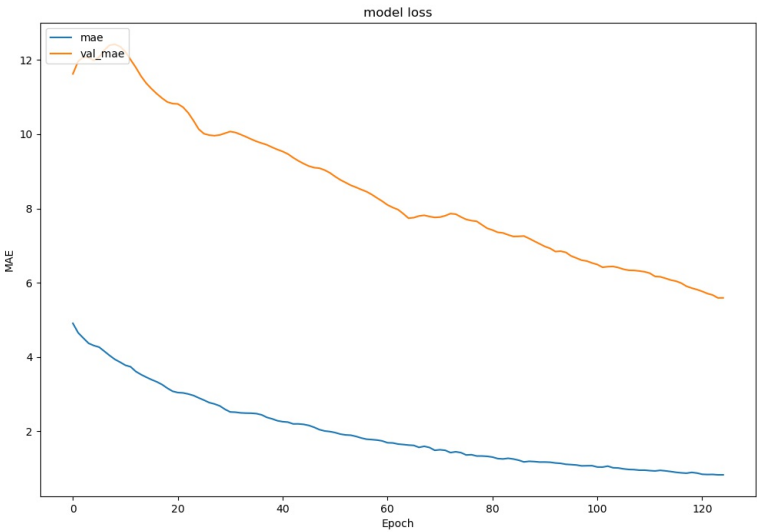


Figure 5.3: Training Visualization of the best performing Linear Regression Model: the number of paying users prediction

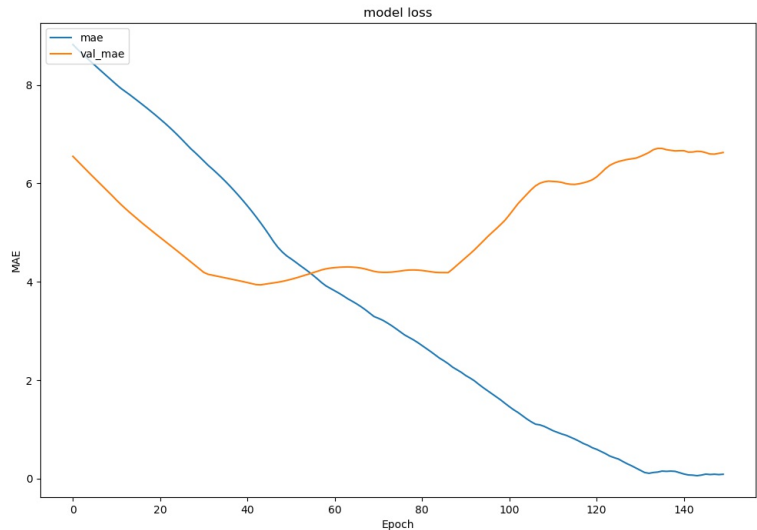


Figure 5.4: Training Visualization of the DNN-Model for the AUM prediction before reducing the number of epochs to 50

### 5.1.2 Decision Forests Models

When training the decision forest models, the parameters were also changed to see how the results would vary. For decision forest models, however, changing the number of trees does not have a large effect like it has with changing the epochs for the regression models. In Figure 5.5 it can be seen that the MSE changes only minimally after the first few iterations. The MSE is set as the loss function. As with the regression models, the two evaluation metrics MAE and MSE are output and additionally, the mean of the training labels and the average deviation are calculated with the MAE and the mean. The validation split is unchanged with 20%. The models were trained with the 24 examples as well as with the additional 112 GAN-synthesized examples.

Table 5.3 shows the results of the training with only 24 rows and Table 5.4 shows the evaluation of the models with the automatically synthesized data. Also, for the decision forest models, the smaller data set with 24 examples performs at least as good as the one with the synthesized data. It can be stated that an automatic synthesis with the GAN generator does not bring much of an improvement. The GAN generator would be helpful to increase a data set of 5-10 campaign samples, since most algorithms, including the decision forest models, for example, cannot handle such a small amount of data. However, since the GAN generator itself needs at least 10 examples each for the training and the test data set, this is also not possible. Additionally, by evaluating the models, it can be concluded that the performance of the random forest and the gradient boosted trees algorithm is relatively similar. Through the multiple training of the models with different train and test data set splits, it was determined that the performances of the two algorithms correlate relatively strongly. This can be seen for example in rows 6 and 7 of Table 5.3 or in rows 2-3, 4-5, 8-9 and 10-11 of Table 5.4. The models with the 24 examples perform better. Which of these models performs best can be determined less clearly as with the regression models. When repeating the training runs with other data splits, the best models could not often be confirmed. The best models in Table 5.3 are those to predict AUM per user (random forest, row 10), fully onboarded male users (gradient boosted trees, row 5), and fully onboarded female users (random forest, row 2). The best model performs with a MAE of 7300 AUM per user and thus a deviation from the mean of 41.5%. In other words, with a true value of 18'000 AUM per user, the predicted value is on average at 10'530 or 25'470 AUM per user. The other models perform with a deviation between 47.6% and 105.2%.

Figures 5.5 and 5.6 show the visualized training of the two best performing random forest models. One is the prediction of AUM per user and the other is the prediction of fully onboarded female users. Only the visualization of the loss function (MSE) and not the MAE is available. It can be seen that random forest models are resistant to over-fitting when increasing the number of trees. In the first visualization (see Figure 5.5), the MSE swings slightly above 1.8 from about the 25th tree on-wards. In Figure 5.6, the MSE swings between 9.25 and 9.50 from the 100th tree on-wards.

For decision forests models, in addition to the training visualizations, the decision trees can be visualized and viewed. Figure 5.7 shows the decision tree for predictions of fully onboarded female users with the random forest model. It is to see that the tree has only a depth of 2. This is due to the small number of examples. If we visualize the tree of

Table 5.3: ML Results of training the Decision Forests models with 24 examples

Model Identifier	Nr. of Trees	MAE	MSE	Mean	Deviation (Mean / MAE)
random_forest-fully_onb_users_f	300	<b>5.595</b>	41.664	11.053	<b>0.506</b>
gradient_boosted_trees-fully_onb_users_f	300	8.282	121.912	11.053	0.749
random_forest-fully_onb_users_m	300	10.977	146.175	17.263	0.636
gradient_boosted_trees-fully_onb_users_m	300	<b>8.211</b>	171.203	17.263	<b>0.476</b>
random_forest-paying_users	300	4.611	42.698	4.647	0.992
gradient_boosted_trees-paying_users	300	4.641	32.163	4.647	0.999
random_forest-aum	300	6.495	90.497	6.174	1.052
gradient_boosted_trees-aum	300	5.030	65.273	6.174	0.815
random_forest-aum_per_user	300	<b>0.728</b>	1.030	1.751	<b>0.415</b>
gradient_boosted_trees-aum_per_user	300	1.041	1.803	1.751	0.595

Table 5.4: ML Results of training the Decision Forests model with GAN-synthesized data (136 examples)

Model Identifier	Nr. of Trees	MAE	MSE	Mean	Deviation (Mean / MAE)
random_forest-fully_onb_users_f	300	10.810	198.527	14.578	0.742
gradient_boosted_trees-fully_onb_users_f	300	10.408	189.073	14.578	0.714
random_forest-fully_onb_users_m	300	19.368	618.528	18.088	1.071
gradient_boosted_trees-fully_onb_users_m	300	19.849	663.570	18.088	1.097
random_forest-paying_users	300	3.731	18.772	3.439	1.085
gradient_boosted_trees-paying_users	300	3.345	16.752	3.439	0.973
random_forest-aum	300	7.260	75.662	10.599	0.685
gradient_boosted_trees-aum	300	6.940	72.883	10.599	0.655
random_forest-aum_per_user	300	1.973	5.849	1.133	1.742
gradient_boosted_trees-aum_per_user	300	2.034	6.688	1.133	1.795

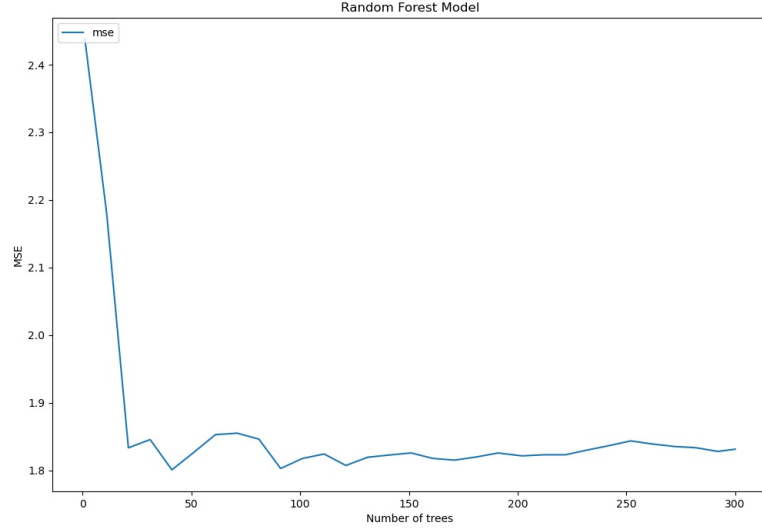


Figure 5.5: Training Visualization of the Random Forest Model for the AUM per user prediction

a model trained with the 136 examples, the visualized tree is larger. In Figure 5.8 the decision tree of the AUM prediction random forest model trained with 136 examples was visualized. The visualization is limited to depth 4.

## 5.2 Cost Analysis

Besides considering and interpreting the machine learning results, the prediction system design was evaluated in terms of its operating costs. The cost analysis was focused on the prediction system part and not extended to the cost estimation of a DWH, because the DWH costs depend significantly on the amount of data it contains. The prediction system, on the other hand, contains only the data used for the machine learning, which is already aggregated. In addition, a DWH is already often used by default, whereas with the prediction system component the question of whether such a component is worthwhile in terms of costs probably arises more frequently. The costs of such a ML system depend on the performance of the ML code and on the frequency the ML code is run. For each of the prototypes, the training performance was measured and the operating costs were calculated.

As mentioned in the description of the implementation, the cheapest *Standard* machine was chosen for the Vertex AI training. This type of machine provides 4 vCPUs and 15 GiB of memory to train the models. Although the performance of Virtual Central Processing Units (vCPU) is not identical to that of regular Central Processing Units (CPU), they are still comparable to each other [88]. A one-to-one comparison is sufficient for this performance comparison and for the resulting cost estimation. Therefore, the Docker Engine on the local machine was provided with 4 CPUs and 16 GB of memory (15 GiB

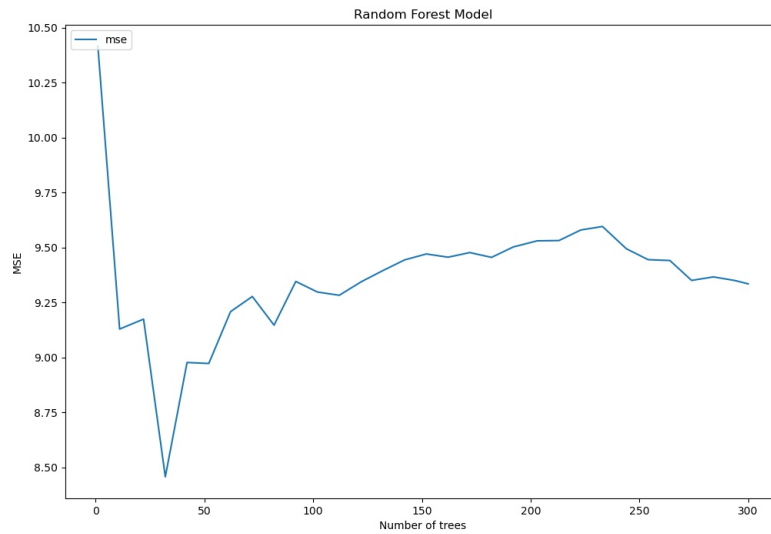


Figure 5.6: Training Visualization of the Random Forest Model for the fully on-boarded female users prediction

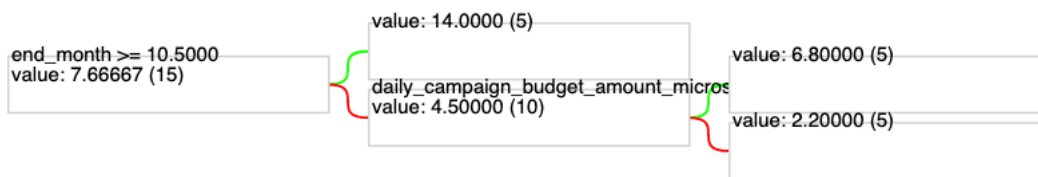


Figure 5.7: Decision Tree for the fully on-boarded female users prediction random forest model trained with 24 examples

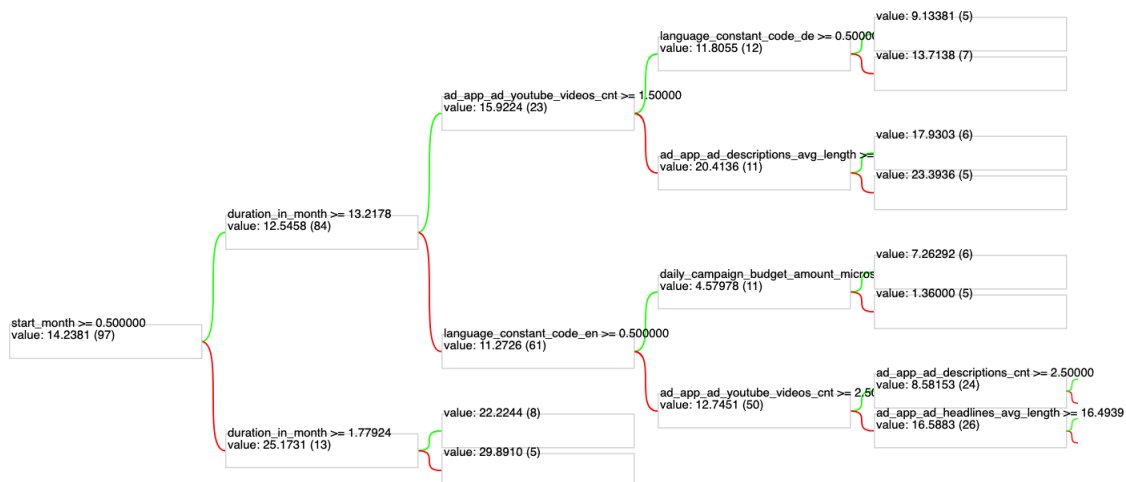


Figure 5.8: Decision Tree for the AUM prediction random forest model trained with 136 examples

= 16.11 GB). Both training systems were run with the same ML code training only the DNN models. The only difference in the code was the file path of the input data and the storage path of the models. Before training, the 24 input examples were multiplied with the GAN synthesizer and stored as new input CSV file, so that the training time and cost correspond to a ML with 136 instead of 24 data records.

The training on the local machine took 21 seconds (with 24 records it took 20 seconds). In Vertex AI, the training job took 12 minutes and 35 seconds. Interestingly, the training also took exactly 12 minutes and 35 seconds when the models were trained with only 24 examples. Two more test runs lasted 2 minutes and 2 seconds each. Another test run after that took 12 minutes and 35 seconds again. A reason for this could not be determined. As a basis for the cost calculation, the higher value was used and rounded up.

The prototype code trains five labels (`fully_onb_users_f`, `fully_onb_users_m`, `paying_users`, `aum` and `aum_per_user`). The proposed design includes 50 labels consequently resulting in 50 models that need to be trained. Since the models are trained in a for-loop in the prototype code and would be trained sequentially in the production code, the training time of the 5 labels is extrapolated by a factor of 10 for the cost estimation. In Table 5.5 the costs for a prediction system with Vertex AI are listed and calculated. The costs were calculated for one year. The following costs were considered: Storing the Docker image in the GCR (which consumes the free tier of the GCS storage), storing the CSV files and the models, the model training, the triggering of the training by a GCF, scheduling this GCF, and hosting the model endpoints. The model training is the most expensive cost item and still very low at \$25 per year. The cost of the Cloud Scheduler job (\$1.20 per year), the model hosting with Vertex AI (\$1.30 per year) and the Google Cloud Storage (\$1.90 per year) are also reasonable, and the GCF costs fall into the Google Cloud Free Tier. Rounded up, the total costs of the predictions are 30 USD per year.

If you compare the cloud costs with a server specifically purchased for this machine learning task, the dedicated server obviously cannot keep up with the costs of the cloud. For a standard server, acquisition costs of 600 CHF are assumed. If this is linearly depreciated over three years, this equals in 200 CHF per year. Electricity and internet costs have to be added to this. Since the server would only be used once a week, an average overall capacity usage of 10-20% can be assumed. For the power consumption at a 10-20% capacity usage, 120 Watt are assumed [75]. With a run time of 24 hours during 365 days, this results in a power consumption of about 1050 kWh. The average energy rate in Switzerland is assumed to be CHF 0.20/kWh [80], which results in total electricity costs of 210 CHF per year. The internet costs are assumed to be relatively low at 50 CHF per month. This results in operating costs of 810 CHF per year (incl. internet) and 210 CHF per year (excl. internet). Adding the server investment results in a rounded total costs of 1000 CHF resp. 400 CHF per year.

## 5.3 Challenges

The implementation of the prediction system prototype brought to light several challenges, some of which were then flowed back into the design through an iterative approach.

Table 5.5: Cost Analysis of Model Training with Google Vertex AI

	Pricing	Usage	Calculated Costs
Docker Container Registry	Google Container Registry uses Standard GCS buckets to store the Docker images. The Docker image only needs to be uploaded once. 5GB per month is free, after that \$0.020 in Belgium (europe-west1) and \$0.025 in Zurich (europe-west6), while in Zurich there is no Vertex AI model training and hosting available yet [59].	The Docker image is 8.31 GiB.	8.31GiB - 5GB = 3.892 GB, 3.892 * \$0.020 * 12 = <b>\$0.93</b>
Storage of CSV Files and Models in GCS bucket	5GB per month is free, after that \$0.020 in Belgium (europe-west1) and \$0.025 in Zurich (europe-west6) per month, while in Zurich there is no Vertex AI model training and hosting available yet [59].	The current bucket size is 259 MB and includes several unused files. Assuming, because of growing data and training models for 50 instead of 5 labels, a bucket size of max. 4 GB.	4.0 * \$0.020 * 12 = <b>\$0.96</b>
Model Training	The cost in Europe regions are \$0.54 per hour, per training unit. For the cheapest machine (n1-standard-4) it is \$0.2200 (0.4074 units) per hour [60].	Training weekly for around 13 minutes (for 5 labels).	\$0.2200 * 10 * 13/60 * 52 = <b>\$24.79</b>
Triggering Weekly Training with GCF	The first 2 million GCF invocations are free. In addition, a free-tier of 400'000 GB-seconds, 200'000 GHz-seconds compute time and 5GB of Internet egress traffic exists. After that, the GCF with 512MB memory and a 200MHz CPU provisioned costs \$0.000000231 per 100ms [58].	Triggering the model training is a very short task. Assuming this function runs for 1 second every week, the free tier is used.	400'000 - 0.512 * 1 * 52 = 399'973, 200'000 - 0.200 * 1 * 52 = 199'990 <b>\$0.00</b>
Triggering the GCF with Cloud Scheduler		One job for triggering the GCF is set up.	\$0.10 * 1 * 12 = <b>\$1.20</b>
Model Import/Configuration		Importing the models to Vertex AI and configuring it needs to be done only once. The models are stored in the GCS bucket.	<b>\$0.00</b>
Model Hosting on End-points	The costs for predictions hosting in Europe regions for the cheapest machine (n1-standard-2) are \$0.1100 per node hour [60].	Assuming every week predictions are made and the node runs 1 hour per week in total.	\$0.1100 * 12 = <b>\$1.32</b>
		<b>Total costs per year</b>	<b>\$29.20</b>



### 5.3.1 Complexity of the Online Advertisement Universe and the Restrictions on the Data Access

In general, it should be noted that the online marketing environment, in which this thesis is located, is very much in development and changes to platforms and APIs happen continuously. The online ad marketing world is fixed on the two universes Facebook (with Instagram and WhatsApp) and Google (with *e.g.*, Google Ads, Youtube, or DV360). Especially on Google, services and companies are acquired and combined with each other, and renewed again, so that the overview can be lost quickly. For example, Google Analytics still has the old product called Universal Analytics in operation and the documentation prominently available, even though Google Analytics 4 is already recommended. Furthermore, there are several projects for marketing campaigns in the mobile app area, one for promoting an app in the browser and the other for advertising in another mobile app. Moreover, and as a third example, the AdWords platform, that was bought by Google, and its API is named *Google AdWords* in the first version and then *Google Ads*, while both APIs and the documentation of it can be accessed. In addition, the ad platforms are very sparse with information about what and how their data can be exported. Especially, when it comes to raw click data, Facebook doesn't allow it at all and Google Ads only allows a small portion that is not available for mobile app campaigns.

### 5.3.2 Linking Data on Ad Level for Mobile App Campaigns

Initially, the goal was to implement predictions at a lower data level. Predictions or at least analytics on the ad level would have been provided, so that the BI analyst would get the information which ad of the campaign performs best. To implement this, the Google Analytics data (which can be connected to the application data without issues) would need to be linked to the Google Ads data.

Generally, there are two ways to do so. Either the Google Click ID, which is tracked by the Firebase SDK and forwarded to Google Analytics, is linked to the Google Click ID referenced in the Click View Report of Google Ads. Or Google Analytics tracks the origin of the click automatically or via UTM parameter (see Section 2.3.2). With either option, mobile app campaigns face unsolvable challenges.

Google Analytics uses the event property *Traffic Source* to automatically track the Google Ads campaign (the campaign name) from which the user originated. Also the so-called *source* and the *medium* is automatically stored. The Google Analytics UI can additionally, if Google Ads is connected to it, segment the user acquisitions by Google Ads' Ad Group. However, this data is not available in the export. To get a deeper segmentation, the use of UTM parameters is required. This means that each ad or its linked URL could be equipped with a different URL query parameter that identifies the ad. This UTM parameter (*e.g.*, `utm_content=imageAdCouple`) would then be automatically tracked by GA. This way, Facebook campaigns and their ads could additionally be tracked by GA. The problem: For mobile app campaigns, neither Google Ads nor Facebook Ads use URLs. Instead, the ads are linked to the app store entry without the possibility to specify

a URL or UTM parameters. In the future, it will be interesting to see if, how and when there will be a UTM tracking option for mobile app campaigns.

The first way mentioned above (linking the GCLID with the Click View Report) looked promising for a long time. As described in Section 2.3.1, there are two methods to get the Click View Report. The Big Query Data Transfer exports a Click View Report table, but it remains empty. After several attempts and contacting Google Cloud Support, it was disclosed that the Click View Report is only available for non-mobile app campaigns. This is another example showing the complexity and chaos between Google platforms: In each case, the Big Query Data Transfer documentation references Google Ads, while Google Cloud Support referenced the AdWords documentation, which indicates which reports are available for app campaigns. This information is not available in the Google Ads documentation. As an alternative to the Big Query Data Transfer, the Google Ads API can be used, which successfully delivers the Click View Report filled with data. The Click View Report includes a reference to the campaign, the ad group and the ad group ad entity. Technically, the campaign, campaign criterion, ad group, ad group ad and ad asset entities can all be successfully connected. However, during the implementation it turned out that the Click View Report did not contain valid references to the ad group and to the ad group ad, while the campaign reference pointed to the expected campaign. The ad group and also the ad group ad ID referenced in the Click View Report are missing in the other tables. Thus, the Google Ads API Support was contacted. They have come to the following conclusion and a solution has not been communicated yet:

*I can confirm that those ad group IDs are unexpected. I've sent this over to someone who specializes in this area to take a look to see if we can filter them out. Thanks for bringing this to our attention!*

This finding means that the Click View Report data cannot be linked to the ad group and ad group ad reports. As a result, the click data is only available at the campaign level. It is not possible, at least not for mobile app campaigns, to extract which user clicked on which ad, but only to which campaign the clicked ad belongs.

Another inconsistency was found when the number of users with a GCLID were compared to the number of users with a set traffic source. Only 62 accounts identified by GA have a GCLID, while 895 users can be assigned to a campaign via the traffic source field. When the GCLID can be retrieved and when it cannot was not found out.

### 5.3.3 Audience Restrictions for Mobile App Campaigns

A final difficulty related to mobile app campaigns is that the audience to which the campaign is directed cannot be extracted. The audience for app campaigns is controlled via Firebase. Existing users can be grouped together. These user lists can then be used in new campaigns to include or exclude users. These user lists are extractable via the Google Ads API, but they do not contain the criteria used to create the user list (*e.g.*, gender). Besides that, the audience can only be defined in a limited way: Only the language and geographic location can be specified in the campaign settings and they can

also be exported as campaign parameters. For non-app campaigns, however, the campaign parameters Gender, Age Group, Income Cluster and Parental Status can also be set as audience properties and can be extracted as campaign parameters for ML.

### 5.3.4 Data Volume & Synthesized Data

For the machine learning part, the amount of data is a big challenge. The above mentioned problems led to the fact that the machine learning could only be performed on campaign level. Since April 2021, only three Google Ads campaigns have been run promoting the app. Two campaigns for Android users (one optimized towards the install conversion and one optimized towards the registration conversion) and one for iPhone users optimized towards installation. Therefore, the data had to be synthesized manually and in an automated way. Obviously, with this synthesized data, the quality of the ML models could not be neither optimized nor verified content-wise. The findings of this thesis show that if this design of a prediction system is to be used for mobile app campaigns, more but shorter campaigns should be run, otherwise the amount of data needed for the predictions cannot be reached. Also, campaigns, ad groups or ads should not be edited, but the old one should always be closed and new ones should be created. However, it remains to be clarified whether this strategy is compatible with the best practices and algorithms of Google or Facebook.

### 5.3.5 Conclusion

It would be interesting to see whether all the challenges mentioned can be overcome with non-app campaigns. In a non-app campaign, for example key word or display campaigns for a browser web application, each ad could have its own UTM identifier. Furthermore, the click view report could be exported via Big Query Data Transfer and the audience parameters of the campaigns, ad groups or individual ads could be used as ML features (see Future Work in Chapter 6). This would mean that all KPIs could be collected and calculated on the ad level and, consequently, the predictions could be made on this level too. For the machine learning, this would mean that the three real examples on campaign level would become more than 15 real examples on ad level (at least five ads per campaign).



## Chapter 6

# Summary, Conclusion, and Future Work

In this thesis, it was researched what data can be exported from online marketing ad platforms, such as Google Ads and Facebook Ads, and how it can be connected to the data collected by the promoted application. With this knowledge, the goal was to find out if and to what extent analyses and predictions regarding the performance of online campaigns can be made that go beyond the metrics of the campaign platforms by using KPIs based on the connected data from different sources. Another goal was to find suitable solutions also for data (application) owners who do not want to send any data, or as little data as possible, to the big cloud providers.

A modular architecture design was developed that extends a client-server architecture (a mobile app with application back-end and separate IAM connected to a production DB) with the integration of ad platforms, a data warehouse accessed with BI tools, and a prediction system component. The design includes the fully automated and recurring extraction of ad click event data, account and event data from the production DB, as well as the data and properties of the campaigns that have been run. In the data warehouse, the data from the different sources is processed and linked to each other before KPIs based on the combination of that data are calculated (*e.g.*, the customer acquisition costs of a campaign or the costs per asset under management). With BI tools, these KPIs can be analyzed and segmented. In the prediction system component, machine learning is used to train models to predict these KPIs.

The design contains the feature and target selection for the machine learning as well as a concept for satisfying the requirement of a system that continuously learns with the freshly gathered data. It also contains the concept for hosting the models as API endpoints and accessing them with prediction requests. A system prototype was developed to evaluate what challenges such a system implementation entails, what machine learning results might be expected and what operational costs the deployment and hosting of such a prediction system would incur. This was done to see if it would be worthwhile to have such a prediction system in place to try to reduce the customer acquisitions costs of campaigns by consulting the machine learning predictions of future campaign outcomes.

The implementation of the prototype revealed that such a system in combination with a mobile app is exposed to several challenges. The main reason for this is that for mobile

app campaigns, which advertise the installation of an app and therefore directly redirect the user to the app store when clicking on the ad, only the app store identifier is specified instead of a link URL, as it is the case for website campaigns. Thus, no custom query parameters (UTM parameter) can be set and the advertiser can therefore not submit additional tags and is dependent on what traffic information is automatically collected. This automatically collected traffic information is limited to the campaign name to which the ad belongs that the user clicked on. This leads to the discovery that the account and usage data of the app can only be connected to the marketing data on campaign level and therefore the KPIs can only be calculated on this level. Consequently, answering **RQ1** (see Section 1.2), the machine learning and predictions are also limited to this level. Since data from only three campaigns was available, the three records with the real campaign parameters and KPIs had to be manually synthesized into 24 examples so that at least the machine learning functionality could be tested.

Due to the small amount of real data, the prediction results of the prototype are useless in terms of campaign-insights and their significance could not be assessed (**RQ2**, Section 1.2). To make this possible, approximately 20-50 real campaign records are required (RQ2). Nevertheless, it was found that for the present regression problem with few data records, a deep neural network model performs best compared to a linear regression, a random forest, and a gradient boosted trees model. Another finding of this work is that the use of the prediction system component in addition to a data warehouse, with the goal to improve the campaign performance and thus reduce customer acquisition costs, might be worth trying, presuming the data volume and ML model quality are high enough. Since using this design of the prediction component with a weekly retraining of the models (hosted in the Google Cloud), costs only about 30 USD per year.

As future work, it would be interesting to test the system prototype with campaigns advertising a normal website or browser web application (non-mobile app campaigns). Since with this kind of advertising, it is possible to add additional, track-able parameters to the URL link of the ads, chances are high, that the analytics and the predictions can be taken to the lower ad level instead of the campaign level. This way, users could predict the outcome of an additional ad instead of an additional campaign. The possibility to use URL parameters also allows to integrate Facebook Ads, Newsletters and any other campaign. This also increases the amount of data, and the prediction models can be optimized and verified based on real data. Once enough real data is available (no matter if at ad or campaign level), the deep neural network model could be optimized from a single output to a multi output model.

# Bibliography

- [1] AdColony - Elevating mobile advertising & monetization, <https://www.adcolony.com/> (accessed 17 August 2021).
- [2] All About Customer Acquisition Cost (CAC) | Propeller CRM Blog, <https://www.propellercrm.com/blog/customer-acquisition-cost> (accessed 14 August 2021).
- [3] Apple Inc, App Store - Apple, <https://www.apple.com/app-store/> (accessed 13 March 2021).
- [4] Apple Inc, Apple Search Ads, <https://searchads.apple.com> (accessed 18 August 2021).
- [5] I. Ashrapov, Tabular GANs for uneven distribution, arXiv:2010.00638 [cs], Oct. 2020, Accessed: Sep. 12, 2021. [Online]. Available: <http://arxiv.org/abs/2010.00638>.
- [6] Basic regression, <https://www.tensorflow.org/tutorials/keras/regression> (accessed 17 September 2021).
- [7] CAC Benchmarks: Is your Customer Acquisition Cost above average, <https://www.profitwell.com/content-marketing-benchmarks> (accessed 14 August 2021).
- [8] Campaign URL Builder, <https://ga-dev-tools.web.app/campaign-url-builder/> (accessed 19 August 2021).
- [9] Challenges faced by companies in adopting online onboarding process (15 September 2020), <https://www.idcentral.io/blog/challenges-faced-by-companies-in-adopting-online-onboarding-process/> (accessed 14 March 2021).
- [10] T. Cui, Y. Wang, and B. Namih, "Build an Intelligent Online Marketing System: An Overview," IEEE Internet Comput., vol. 23, no. 4, pp. 53-60, Jul. 2019, doi: 10.1109/MIC.2019.2924637.
- [11] Customer.io, Marketing Automation for the Whole Customer Lifecycle, <https://customer.io/> (accessed 13 March 2021).
- [12] Datorama | AI-powered marketing intelligence, <https://datorama.com> (accessed 19 August 2021).
- [13] C. Doyle, A dictionary of marketing, Fourth edition. Oxford; New York: Oxford University Press, 2016.

- [14] Facebook Inc, Facebook Ads Manager: Ads management for Facebook, Instagram | Facebook for Business, <https://en-gb.facebook.com/business/tools/ads-manager> (accessed 14 August 2021).
- [15] Facebook Inc, Facebook ads: Online advertising on Facebook | Facebook for Business, <https://www.facebook.com/business/ads/> (accessed 13 March 2021).
- [16] Facebook Inc, Graph API, <https://developers.facebook.com/docs/graph-api/> (accessed 16 August 2021).
- [17] Facebook Inc, Insights API - Marketing API, <https://developers.facebook.com/docs/marketing-api/insights/> (accessed 15 August 2021).
- [18] Facebook Inc, Platzierungen im Werbeanzeigenmanager | Facebook for Business, <https://www.facebook.com/business/help/407108559393196?id=369787570424415> (accessed 14 August 2021).
- [19] Finma reduces obstacles to FinTech (17 March 2016), <https://www.finma.ch/en/news/2016/03/20160317-mm-fintech/> (accessed 13 March 2021).
- [20] GDA vs. DV360: Comparing Google's Display Platforms, <https://www.merkleinc.com/emea/blog/where-should-you-run-your-display-activity-a-comprehensive-comparison-of-googles-display-platforms> (accessed 14 August 2021).
- [21] A. Géron, Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems, First edition. Beijing; Boston: O'Reilly Media, 2017.
- [22] C. Giri, U. Johansson, and T. Lofstrom, Predictive Modeling of Campaigns to Quantify Performance in Fashion Retail Industry, in 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, Dec. 2019, pp. 2267-2273. doi: 10.1109/BigData47090.2019.9005492.
- [23] GitHub Inc., Diyago/GAN-for-tabular-data: We well know GANs for success in the realistic image generation. However, they can be applied in tabular data generation. We will review and examine some recent papers about tabular GANs in action., <https://github.com/Diyago/GAN-for-tabular-data> (accessed 14 September 2021).
- [24] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. Cambridge, Massachusetts: The MIT Press, 2016.
- [25] Google Big Query - Tableau, [https://help.tableau.com/current/pro/desktop/en-us/examples\\_googlebigquery.html](https://help.tableau.com/current/pro/desktop/en-us/examples_googlebigquery.html) (accessed 24 August 2021).
- [26] Google Inc, ad\_group\_ad | Google Ads API | Google Developers, [https://developers.google.com/google-ads/api/fields/v8/ad\\_group\\_ad](https://developers.google.com/google-ads/api/fields/v8/ad_group_ad) (accessed 18 August 2021).



- [27] Google Inc, Add gtag.js to your site | Universal Analytics for Web (gtag.js), <https://developers.google.com/analytics/devguides/collection/gtagjs> (accessed 18 August 2021).
- [28] Google Inc, Analytics, Google Analytics Homepage, <https://analytics.google.com> (accessed 14 August 2021).
- [29] Google Inc, Analytics for Firebase - Android | Google Developers, <https://developers.google.com/analytics/devguides/collection/firebase/android> (accessed 18 August 2021).
- [30] Google Inc, Analytics for Firebase - iOS | Google Developers, <https://developers.google.com/analytics/devguides/collection/firebase/ios> (accessed 18 August 2021).
- [31] Google Inc, Android Apps on Google Play, <https://play.google.com/store/apps> (accessed 13 March 2021).
- [32] Google Inc, Auto-tagging: Definition - Google Ads Help, [https://support.google.com/google-ads/answer/1752125?hl=en\\_US#null](https://support.google.com/google-ads/answer/1752125?hl=en_US#null) (accessed 19 August 2021).
- [33] Google Inc, BigQuery: Cloud Data Warehouse | Google Cloud, <https://cloud.google.com/bigquery> (accessed 14 August 2021).
- [34] Google Inc, [GA4] Big Query Export Schema - Analytics Help, [https://support.google.com/analytics/answer/7029846?hl=en&ref\\_topic=9359001](https://support.google.com/analytics/answer/7029846?hl=en&ref_topic=9359001) (accessed 18 August 2021).
- [35] Google Inc, click\_view | Google Ads API | Google Developers, [https://developers.google.com/google-ads/api/fields/v8/click\\_view](https://developers.google.com/google-ads/api/fields/v8/click_view) (accessed 18 August 2021).
- [36] Google Inc, Client Libraries | Google Ads API | Google Developers, <https://developers.google.com/google-ads/api/fields/v8/overview> (accessed 14 August 2021).
- [37] Google Inc, Cloud Computing Services | Google Cloud, <https://cloud.google.com/> (accessed 14 August 2021).
- [38] Google Inc, Cloud Pub/Sub | Google CloudCloud Pub/Sub | Google Cloud, <https://cloud.google.com/pubsub/docs/overview> (accessed 25 August 2021).
- [39] Google Inc, Cloud Scheduler | Google Cloud, <https://cloud.google.com/scheduler> (accessed 25 August 2021).
- [40] Google Inc, Collect campaign data with custom URLs - Analytics Help, <https://support.google.com/analytics/answer/1033863?#zippy=\%2Cin-this-article> (accessed 19 August 2021).
- [41] Google Inc, Control API access with domain-wide delegation - Google Workspace Admin Help, <https://support.google.com/a/answer/162106> (accessed 25 August 2021).

- [42] Google Inc, Dashboarding & Data Visualization Tools - Google Data Studio, <https://marketingplatform.google.com/about/data-studio/> (accessed 23 September 2021).
- [43] Google Inc, Data Transfer v2.0 | Display & Video 360 | Google Developers, <https://developers.google.com/bid-manager/dtv2/overview> (accessed 18 August 2021).
- [44] Google Inc, End to End Campaign Management - Google Display & Video 360, <https://marketingplatform.google.com/about/display-video-360/> (accessed 13 March 2021).
- [45] Google Inc, Enterprise Advertising & Analytics Solutions - Google Marketing Platform, <https://marketingplatform.google.com/about/enterprise/> (accessed 13 March 2021).
- [46] Google Inc, FAQs & Advertising Resources - Google Ads, <https://ads.google.com/home/faq> (accessed 14 August 2021).
- [47] Google Inc, Firebase, <https://firebase.google.com/> (accessed 18 August 2021).
- [48] Google Inc, Free Business Analytics Solutions - Google Marketing Platform, <https://marketingplatform.google.com/about/small-business/> (accessed 13 March 2021).
- [49] Google Inc, Get started | DV360 API | Google Developers, <https://developers.google.com/display-video/api/guides/getting-started/overview> (accessed 14 August 2021).
- [50] Google Inc, Get Started | Google Ads scripts | Google Developers, <https://developers.google.com/google-ads/scripts/docs/your-first-script> (accessed 14 August 2021).
- [51] Google Inc, Google Ads - Get More Customers With Easy Online Advertising, <https://ads.google.com/intl/en/home/> (accessed 13 March 2021).
- [52] Google Inc, Google Ads Transfers | BigQuery Data Transfer Service | Google Cloud, <https://cloud.google.com/bigquery-transfer/docs/adwords-transfer> (accessed 14 August 2021).
- [53] Google Inc, Google Tag Manager (GTM), <https://tagmanager.google.com/> (accessed 16 August 2021).
- [54] Google Inc, Measurement Protocol (Google Analytics 4), <https://developers.google.com/analytics/devguides/collection/protocol/ga4> (accessed 18 August 2021).
- [55] Google Inc, Mobile App Monetization - Google AdMob, <https://admob.google.com/home/> (accessed 17 August 2021).
- [56] Google Inc, OAuth Service Account Flow | Google Ads API | Google Developers, <https://developers.google.com/google-ads/api/docs/client-libs/java/oauth-service> (accessed 25 August 2021).

- [57] Google Inc, Overview Analytics Reporting API v4 | Google Developers, <https://developers.google.com/analytics/devguides/reporting/core/v4> (accessed 18 August 2021).
- [58] Google Inc, Pricing | Cloud Functions | Google Cloud, <https://cloud.google.com/functions/pricing> (accessed 11 September 2021).
- [59] Google Inc, Pricing | Cloud Storage | Google Cloud, <https://cloud.google.com/storage/pricing> (accessed 11 September 2021).
- [60] Google Inc, Pricing | Vertex AI | Google Cloud, [https://cloud.google.com/vertex-ai/pricing#custom-trained\\_models](https://cloud.google.com/vertex-ai/pricing#custom-trained_models) (accessed 11 September 2021).
- [61] Google Inc, Report Types | AdWords API | Google Developers, <https://developers.google.com/adwords/api/docs/appendix/reports> (accessed 14 August 2021).
- [62] Google Inc, Reports | Google Ads API | Google Developers, <https://developers.google.com/google-ads/api/fields/v8/overview> (accessed 14 August 2021).
- [63] Google Inc, Using OAuth 2.0 for Server to Server Applications | Google Identity, <https://developers.google.com/identity/protocols/oauth2/service-account#delegatingauthority> (accessed 25 August 2021).
- [64] Google Inc, Vertex AI | Google Cloud, <https://cloud.google.com/vertex-ai> (accessed 1 September 2021).
- [65] Google Inc, Youtube, <https://www.youtube.com/> (accessed 15 September 2021).
- [66] Keras: the Python deep learning API, <https://keras.io/> (accessed 17 September 2021).
- [67] J. Kreps, N. Narkhede and J. Rao. 2011. Kafka: A distributed messaging system for log processing. Proc. 6th Int. Workshop Netw. Meets Databases (NetDB). pp. 1-7.
- [68] LinkedIn Ads: Targeted Self-Service Ads | LinkedIn Marketing Solutions, <https://business.linkedin.com> (accessed 17 August 2021).
- [69] Marketing Data Intelligence Platform for Digital Businesses, <https://www.easyinsights.ai/> (accessed 19 August 2021).
- [70] Measured: Marketing Attribution & Incrementality Testing, <https://www.measured.com> (accessed 19 August 2021).
- [71] Microsoft Inc, Microsoft Advertising | Search Engine Marketing (SEM) & more, <https://ads.microsoft.com> (accessed 18 August 2021).
- [72] ML6 | Machine Learning & AI Experts - Drive business impact with data and AI, <https://www.ml6.eu/> (accessed 3 September 2021).
- [73] Overview of Docker Compose | Docker Documentation, <https://docs.docker.com/compose/> (accessed 21 September 2021).

- [74] B. K. Pak, B. Mocan, S. Y. Yoldas, and N. Baz, "Development of Autonomous Intelligent System for Google Ads," in 2018 Thirteenth International Conference on Digital Information Management (ICDIM), Berlin, Germany, Sep. 2018, pp. 102-107. doi: 10.1109/ICDIM.2018.8847128.
- [75] Power Management Statistics: Information Technology - Northwestern University, <https://www.it.northwestern.edu/hardware/eco/stats.html> (accessed 11 September 2021).
- [76] PyTorch, <https://pytorch.org/> (accessed 3 September 2021).
- [77] React Native - Learn once, write anywhere, <https://reactnative.dev/> (accessed 4 September 2021).
- [78] Reduce your ad cost by up to 30% - nexoya, <https://www.nexoya.com/> (accessed 19 August 2021).
- [79] Salesforce: We Bring Companies and Customers Together, <https://www.salesforce.com> (accessed 19 August 2021).
- [80] Electricity tariffs, <https://www.elcom.admin.ch/elcom/en/home/topics/electricity-tariffs.html> (accessed 11 September 2021).
- [81] scikit-learn: machine learning in Python - scikit-learn 1.0 documentation, <https://scikit-learn.org/stable/> (accessed 3 September 2021).
- [82] Singular | Leaders in Marketing Analytics and Attribution, <https://www.singular.net> (accessed 19 August 2021).
- [83] TensorFlow, <https://www.tensorflow.org/> (accessed 3 September 2021).
- [84] TensorFlow Serving with Docker | TFX, <https://www.tensorflow.org/tfx/serving/docker> (accessed 3 September 2021).
- [85] The Mobile Measurement Company | Adjust, <https://www.adjust.com/> (accessed 13 March 2021).
- [86] Twitter Ads, <https://ads.twitter.com> (accessed 19 August 2021).
- [87] Verbessere deine Conversion Rate & Gewinne mehr Stammkunden | maatoo.io, <https://maatoo.io/> (accessed 13 March 2021).
- [88] What is a VMware vCPU? - Hyve Managed Hosting, <https://www.hyve.com/what-is-a-vmware-vcpu/> (accessed 11 September 2021).
- [89] Yelp Ads | Advertise your local business | Yelp for Business, <https://business.yelp.com/products/yelp-ads> (accessed 17 August 2021).

# Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AUC	Area Under the Curve
AUM	Assets Under Management
AWS	Amazon Web Services
BI	Business Intelligence
CA	Classification Accuracy
CAC	Customer Acquisition Costs
CLI	Command Line Interface
CMEK	Customer-Managed Encryption Key
CPA	Cost Per Action
CPC	Cost Per Click
CPE	Cost Per Engagement
CPI	Cost Per Impression
CPU	Central Processing Unit
CRM	Customer Relation Management
CTR	Click-through Rate
DNN	Deep Neural Network
DB	DataBase
DT	Decision Trees
DV360	Display & Video 360
DWH	Data Warehouse
ETL	Extract Transform and Load
GAN	Generative Adversarial Network
GCF	Google Cloud Function
GCLID	Google Click ID
GCR	Google Container Registry
GCS	Google Cloud Storage
GTM	Google Tag Manager
IAM	Identity Access Management
JDBC	Java Database Connectivity
KPI	Key Performance Indicator
LTV	Life Time Value
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron

ML	Machine Learning
MSE	Mean Squared Error
NLP	Natural Language Processing
PPC	Pay Per Click
RMSE	Rooted Mean Squared Error
ROC	Receiver Operating Characteristic
ROI	Return on Investment
SDK	Software Development Kit
SQL	Structured Query Language
SVM	Support Vector Machines
UDF	User-defined Function
URI	Uniform Resource Identifier
UTM	Urchin Tracking Module
vCPU	Virtual Central Processing Unit

# List of Figures

2.1	Google Analytics' User Interface . . . . .	17
2.2	A simple ANN: The Multi-Layer Perceptron . . . . .	30
2.3	Multi-touch Attribution [10] . . . . .	32
2.4	The 28 Campaign Data Attributes as Model Features [22] . . . . .	33
2.5	Clustering of the Campaign Performances [22] . . . . .	34
2.6	Confusion Matrices of the Classifier Predictions [22] . . . . .	34
2.7	ROC Curves for the Classifier Predictions . . . . .	35
3.1	Designed Architecture Overview . . . . .	38
3.2	Google Cloud Data Warehouse Design . . . . .	40
3.3	Self-hosted Data Warehouse Design . . . . .	44
3.4	Overall Design with a Cloud DWH and a Vertex AI Prediction Component	51
3.5	Self-hosted Model Training and Serving Design . . . . .	53
3.6	Design with a self-hosted DWH, Vertex AI Training and self-hosted Model Serving . . . . .	54
4.1	Architecture of the Prototype Implementation . . . . .	58
4.2	Configuration for the Google Cloud Scheduler job which triggers the Google Cloud Functions . . . . .	68
4.3	Configuration and Stats of the last Vertex AI Model Training with a custom Docker image . . . . .	85
4.4	Campaign Predictions Page of the Prediction Front-End, the Campaign Parameter Input Fields, Part 1 . . . . .	88
4.5	Campaign Predictions Page of the Prediction Front-End, the Campaign Parameter Input Fields, Part 2 . . . . .	89

4.6	Campaign Predictions Page with the retrieved Prediction Results and the calculated KPIs . . . . .	90
4.7	BI Dashboard built in Google Data Studio . . . . .	92
4.8	BI Chart that visualizes the fully on-boarded users split by traffic source and gender-age-group . . . . .	92
5.1	Training Visualization of the DNN-Model for the AUM prediction . . . . .	95
5.2	Training Visualization of the DNN-Model for the fully on-boarded female users prediction . . . . .	95
5.3	Training Visualization of the best performing Linear Regression Model: the number of paying users prediction . . . . .	96
5.4	Training Visualization of the DNN-Model for the AUM prediction before reducing the number of epochs to 50 . . . . .	96
5.5	Training Visualization of the Random Forest Model for the AUM per user prediction . . . . .	99
5.6	Training Visualization of the Random Forest Model for the fully on-boarded female users prediction . . . . .	100
5.7	Decision Tree for the fully on-boarded female users prediction random forest model trained with 24 examples . . . . .	100
5.8	Decision Tree for the AUM prediction random forest model trained with 136 examples . . . . .	100



# List of Tables

2.1	Click View Report from the Google Ads API . . . . .	15
2.2	Google Ads Export Data [34] . . . . .	20
2.2	Google Ads Export Data [34] . . . . .	21
2.2	Google Ads Export Data [34] . . . . .	22
2.2	Google Ads Export Data [34] . . . . .	23
2.2	Google Ads Export Data [34] . . . . .	24
2.3	Similar Platforms Comparison . . . . .	27
3.1	Selected features for the ML models . . . . .	48
3.2	Selected KPIs (labels) for the ML models . . . . .	49
5.1	ML Results of training the Regression models with 24 examples . . . . .	94
5.2	ML Results of training the Regression models with GAN-synthesized data (136 examples) . . . . .	94
5.3	ML Results of training the Decision Forests models with 24 examples . . . . .	98
5.4	ML Results of training the Decision Forests model with GAN-synthesized data (136 examples) . . . . .	98
5.5	Cost Analysis of Model Training with Google Vertex AI . . . . .	102



# Appendix A

## Installation Guidelines

### A.1 Repository Structure

The source code contains a directory for each component of the implemented prototype. In the following it is described what directories the repository includes and what can be found in there.

- **application-backend:** Contains the Java Spring Boot Application Back-end. The existing application was reduced to the basics such that it now contains only the *Account* and *Event* entity and the added endpoints for sending events.
- **application-db**
  - *export:* This sub folder contains the SQL scripts and the cockroach commands to export the data from the application DB. In addition, the mentioned *kubernetes-cron-job.yaml* file is included.
  - *init-db.sh:* This script is used to initialize the Cockroach DB.
  - *.data:* The data from the DB is saved in this folder.
- **cloud-data-export:** Contains utilities to export data from Big Query and Google Ads.
  - *big-query:* Contains the SQL scripts for creating the GA events views in Big Query.
  - *google-cloud-functions:* Contains the Java code for running the GCF locally. The GCFs *exportGoogleAdsData* (to export Google Ads data), *flattenGaEvents* (to unnest the GA events table in Big Query table) and *gaEventsCsvToGcs* (to export GA events to GCS) are included.
- **data-warehouse:** Contains a folder for each stage of the DWH.
  - *0-schema-creation:* Contains the SQL scripts to create the tables.

- *1-imports*: Contains the import scripts and commands to import the data into the tables. Most of the commands are also included in the *dwh-cron-job* directory.
  - *2-views*: Contains the views which transform the data for each source.
  - *3-connecting-sources*: Contains the view which connect the data from different sources
  - *4-export*: Contains the script to export the data for the ML.
  - *init-db.sh*: This script is used to initialize the DWH Cockroach DB.
  - *.data*: The data from the DWH DB is saved in this folder.
- **document-storage**: This is the mocked internal document storage. The services save and import the CSV files from this folder.
- **dwh-cron-job**: Contains the scripts and the definition of the *cron job* Docker container.
    - *application-db-export*: Contains the scripts which export the data form the application DB.
    - *dwh-export*: Contains the script to export the ML input data view from the DWH.
    - *dwh-import*: Contains the commands for importing the data from the CSV files in the document storage to the DWH.
    - *Dockerfile*: Defines the Docker container for starting the *cron job* service
    - *dwh-crontab*: The crontab file which is used bei de *cron job*.
- **ml-model-serving**: Contains the files used for serving the models.
    - *models.config*: The model configuration file. It defines where the model can be found and the name of the model.
    - *README.md*: Deploy and usage instructions for the ML Serving service.
- **ml-model-training**:
    - *trainer*: Contains the machine learning code for the decision forests and the regression models.
    - *Dockerfile\_Cron*: The Dockerfile for starting the *cron job* container which starts the training of the models every week.
    - *Dockerfile\_Train*: The Docker file which is used to train the model instantly.
    - *evaluation tables.xlsx*: The Excel file which was used for comparing and evaluating the ML results and for the cost analysis.
    - *ml-crontab*: The crontab which defines the weekly training of the models.
    - *README.md*: Contains instructions how to train the models locally and with Vertex AI.

- *training.log*: The manual training.log which was done during training the models.
- **prediction-frontend**: Contains the source code for the React admin & analytics dashboard.
- **.env**: The environment variables which are needed by the docker-compose file.
- **docker-compose.yml**: The Docker Compose configuration file specifies all services resulting in a container network.
- **README.md**: The README on root level contains instructions how to use the Docker Compose system to build and run all the services. It also contains information how to connect to a running container and how to send prediction requests to the Model Serving service.

## A.2 Running the System on the Local Machine

All instructions can also be found in the *README.md* files.

### Prerequisites

- Docker needs to be installed.
- (For accessing Google Ads data) It need to be applied for a Google Ads Developer Token. The received token needs to be added to *cloud-data-export/google-cloud-functions/ads.properties*. See here for more information: <https://developers.google.com/google-ads/api/docs/first-call/dev-token>
- (For accessing Google Ads and Google Analytics data) A Google Cloud service account needs to be created. The service account credentials need to be downloaded as JSON and copied to the *dwh-cron-job* and the *cloud-data-export/google-cloud-functions* folder. See here for more information: <https://developers.google.com/google-ads/api/docs/client-libs/java/oauth-service>
- For the service account *Delegating domain-wide authority to the service account* is important. See here: <https://developers.google.com/identity/protocols/oauth2/service-account#delegatingauthority>

### Build the Services

The whole system resp. all services can be build with this command:

```
1 docker compose --env-file .env build
```

Build a single service:

```
1 docker compose --env-file .env application-db
```

## Run the Services

The built system services can be run with this command:

```
1 docker compose --env-file .env up --abort-on-container-exit
```

Run a single service:

```
1 docker compose --env-file .env up --abort-on-container-exit application-  
  db
```

Connecting to a running container is possible with:

```
1 docker exec -it application-db /bin/sh
```

## Retrieving Predictions

When all services are run, the system is ready to receive prediction requests.

From a regression model:

```
1 curl -d '{"instances": [[1.4838871833555929,  
2   1.8659883497083019,  
3   2.234620276849616,  
4   1.0187816540094903,  
5   -2.530890710602246,  
6   -1.6046416850441676,  
7   -0.4651483719733302,  
8   -0.4952254087173721,  
9   -0.4952254087173721,  
10  -0.4952254087173721,  
11  -0.4952254087173721,  
12  -0.4952254087173721,  
13  -0.4952254087173721,  
14  -0.4952254087173721,  
15  -0.4952254087173721,  
16  -0.4952254087173721,  
17  -0.4952254087173721,  
18  -0.4952254087173721,  
19  -0.4952254087173721,  
20  -0.4952254087173721,  
21  0.7746763768735953]]  
22 }' \  
23 -X POST http://localhost:8501/v1/models/  
    linear_regression_fully_onb_users_f:predict
```

From a decision forest model:

```
1 curl -d '{"instances": [{  
2   "ad_app_ad_descriptions_avg_length": [1.0],  
3   "ad_app_ad_descriptions_cnt": [1],  
4   "ad_app_ad_headlines_avg_length": [1.0],  
5   "ad_app_ad_headlines_cnt": [1],  
6   "ad_app_ad_images_cnt": [1],
```

```

7   "ad_app_ad_youtube_videos_cnt": [1],
8   "campaign_advertising_channel_sub_type": ["APP_CAMPAIGN"],
9   "campaign_advertising_channel_type": ["MULTI_CHANNEL"],
10  "campaign_app_campaign_setting_app_store": ["GOOGLE_APP_STORE"],
11  "campaign_app_campaign_setting_bidding_strategy_goal_type": ["
    OPTIMIZE_IN_APP_CONVERSIONS_TARGET_CONVERSION_COST"],
12  "campaign_target_cpa_target_cpa_micros": [1],
13  "daily_campaign_budget_amount_micros": [1],
14  "duration_in_month": [1.0],
15  "end_month": [1],
16  "language_constant_code_de": [1],
17  "language_constant_code_en": [1],
18  "language_constant_code_fr": [1],
19  "language_constant_code_it": [1],
20  "start_month": [1]
21  }]
22 }' \
23 -X POST http://localhost:8501/v1/models/random_forest_fully_onb_users_f:
    predict

```

### Retrieving Predictions from the Prediction Front-End

The front-end can be built and started in developer-mode with:

```
1 cd prediction-frontend && yarn && yarn start
```

### Retraining the Models

The models are automatically retrained every seven days. If the training should start immediately, the following two steps should be done.

1. Uncomment the `ml-model-training-now` service (line 88-94) in the `docker-compose.yml` file.
2. Build and run the container with
 

```
1 docker compose --env-file .env ml-model-training-now && \
2 docker compose --env-file .env up --abort-on-container-exit ml-
    model-training-now
```

### Serving the Models

A new version of models is pulled automatically every seven days. If new models should be served immediately, this can be done by restarting the service:

```
1 docker compose restart ml-model-serving
```

or, if the service does not run, by starting it:

```
1 docker compose --env-file .env up --abort-on-container-exit ml-model-
    serving
```

## Running the Google Cloud Functions

If the GCF want to be run locally for debugging or to get the current Google Ads data, a Google Ads Developer Token and a Google Cloud Service Account needs to be created. If they are at hand the following steps can be done:

1. Place the service account JSON file at *cloud-data-export/google-cloud-functions*
2. Adjust the information in *cloud-data-export/google-cloud-functions/ads.properties* with the path to the service account JSON file and the developer token. See here for help: <https://developers.google.com/google-ads/api/docs/client-libs/java/config-file>
3. Run the following command:
 

```
1 ./gradlew runFunction -Prun.functionTarget=LocalHttpRunner -Prun.port=8093
```
4. Click on the link in the console to trigger the GCF.

## A.3 Running the Prediction Component on Vertex AI

### Training the Models

These steps need to be followed to train the models with Vertex AI:

1. Set the `BUCKET_NAME` in *trainer/decision\_forests.py* and in *trainer/regression.py* referencing to the GCS bucket
2. Build the container:
 

```
1 cd ml-model-training && docker build ./ -t "gcr.io/ml-model-training/ml/v1"
```
3. Get the service account JSON. See <https://cloud.google.com/container-registry/docs/advanced-authentication> for help.
4. Login to GCR with Docker:
 

```
1 cat <SERVICE_ACCOUNT_JSON>.json | docker login -u _json_key --password-stdin https://gcr.io
```
5. Push the Docker image to GCR:
 

```
1 docker push "gcr.io/ml-model-training/ml/v1"
```
6. Do not forget to upload the input data CSV file to the GCS bucket referenced in the code.



7. Create training in Vertex AI:
8. Choose *no managed dataset* and *custom training*
9. Add a model name
10. Search the uploaded container image in the GCR
11. Choose *no hyper parameter tuning*
12. Choose the machine type: *e.g.*, the cheapest one: *Standard* -> *Standard* -> *n1-standard-4*, *4 vCPUs*, *15 GiB memory*
13. Start the training

After the training, the models can be found in the specified GCS bucket.

### Serving the Models

Import a model from the GCS bucket:

1. Go to the Models section of Vertex AI and choose *Import Models*
2. Select *Import Model artifacts into a new pre-built container*.
3. Choose *TensorFlow* with the version *2.6* as pre-built container.
4. Select the GCS bucket path where the model is saved.
5. *Import*

Hosting the model endpoint:

1. Go to the Endpoints section of Vertex AI and click *Create Endpoint*.
2. Choose the model which was created before.
3. Use the default settings for everything else.

### Retrieving Predictions from Vertex AI Endpoints

Set the `ENDPOINT_ID` and the Google `PROJECT_ID` as environment variables. Then, predictions from the DNN-model can be retrieved with the following command.

The Google Cloud SDK is needed for getting the credentials in line 5.

```
1 ENDPOINT_ID="3465731019494129664" \  
2 PROJECT_ID="<GCLLOUD_PROJECT_ID>" \  
3 bash -c 'curl \  
4 -X POST \  
5 -H "Authorization: Bearer $(gcloud auth print-access-token)" \  
6 -H "Content-Type: application/json" \  
7 https://europe-west1-aiplatform.googleapis.com/v1/projects/${PROJECT_ID  
    }/locations/europe-west1/endpoints/${ENDPOINT_ID}:predict \  
8 -d "{  
9   "instances": [[  
10    1.4838871833555929,  
11    1.8659883497083019,  
12    2.234620276849616,  
13    1.0187816540094903,  
14    -2.530890710602246,  
15    -1.6046416850441676,  
16    -0.4651483719733302,  
17    -0.4952254087173721,  
18    -0.4952254087173721,  
19    -0.4952254087173721,  
20    -0.4952254087173721,  
21    -0.4952254087173721,  
22    -0.4952254087173721,  
23    -0.4952254087173721,  
24    -0.4952254087173721,  
25    -0.4952254087173721,  
26    -0.4952254087173721,  
27    -0.4952254087173721,  
28    -0.4952254087173721,  
29    -0.4952254087173721,  
30    0.7746763768735953  
31  ]]]"  
32 '
```

# Appendix B

## Contents of the CD

The attached CD and a ZIP file contains the following directories and files:

- **System Source Code:** Contains a repository with the source code of all implemented services and for each service a *README.md* file with instructions. The sub-folders contain also the ML training logs and Excel files used for the ML results comparison and the cost analysis calculations.
- **Related Work:** Contains the discussed related work as PDF documents.
- **Written Thesis Report:** This directory contains the written report with the following files:
  - *Abstract.txt*: Plain text file with the English abstract.
  - *Zusammenfassung.txt*: Plain text file with the German abstract.
  - *Thesis.pdf*: The written report in PDF format.
  - *Latex Sources*: The ZIP file of all Latex sources including all referenced images.