Project Report for a Scholarship of the Austrian Marshall Plan Foundation

FH Salzburg
Informationstechnik &
System-Management

UMASS
AMHERST

Project Title:

Self-attention based System for Joint Search and Recommendation

Scientific Supervisors:

Hamed Zamani
Associate Director of the Center for Intelligent Information Retrieval (CIIR) and Assistant Professor in the College of Information and Computer Sciences (CICS)

Center for Intelligent Information Retrieval
College of Information and Computer Sciences
University of Massachusetts Amherst
140 Governors Drive
Amherst, MA 01003-9264

Cornelia Ferner
Lecturer and Researcher at the Applied Data Science Lab (ADSLab)

Stefan Wegenkittl
Academic Program Director Applied Image and Signal Processing (AIS)

Salzburg University of Applied Sciences (SUAS)
Urstein Süd 1
5412 Puch
Austria

Submitted by:

Stjepan Bijelonjić, BSc.
Master Student at Salzburg University of Applied Sciences
Degree Program: Information Technology & Systems Management (ITS)

Table of Contents

Figures

Tables

1. Introduction

The work described in this paper was done between March and August 2022 during a stay at the Center of Intelligent Information Retrieval at the University of Massachusetts Amherst. The Austrian Marshall Plan Foundation funded this Research Scholarship which made it possible for the author to work from Amherst and benefit from the wide knowledge of both, the *Center of Intelligent Information Retrieval* and *Salzburg's University of Applied Sciences* which have a high interest in information retrieval and natural language processing (NLP). Since the research scholarship was done during the authors master studies, all approaches and result will also be part of a master thesis which will deal with a similar topic.

Technical advancements have opened the door for using more complex algorithms to simplify our daily lives. With better internet connections, high performance CPUs and GPUs and more memory capabilities, even the smallest devices today are able to solve complex problems by using probabilistic models, machine learning or other technologies. While machine learning is finding more and more application in a wide variety of subject areas, there are still a wide range of applications that have not yet been researched enough. [1]

Search engines and recommendation systems two often machine learning based algorithms which help users to find desired information. Although both systems have similar goals, they are usually designed independently. Zamani and Croft have already put effort in exploring joint search and recommendation systems with user item interactions [2]. The work in this paper extends their research by combining search and recommendation with the current state of the art approach for natural language processing tasks – transformers [3], which could find application in multiple real-word scenarios, like e-commerce websites or streaming services. One main challenge in this paper is the size of the dataset. Recent advancements have shown that Bidirectional Encoder Representations from Transformers (BERT) [4] improves the results in many NLP tasks. However, the results depend on the size of the finetuning dataset and research has shown that BERT can have its difficulties with smaller datasets [5] [6]. Consequently, the goal of this work is to research if BERT can be used as part of a joint search and recommendation system and how it performs in combination with a small dataset.

2. Background

This topic explains the backgrounds of the components and methods used in this paper. This should help to gain an overview about the project's scope and depth. It also explains keywords and topics which are necessary for the understanding of the work described later in the report.

2.1. Machine Learning and Artificial intelligence

Machine learning in general is the term used to describe computer systems that automatically improve their own capabilities through experience [7]. Although the idea of machine learning was already discussed back in 1990 [7], there is still a lot of potential in many-many fields to exploit today. It is a term which is sometimes used interchangeably with Artificial Intelligence, although they are not the same. Artificial intelligence (AI) describes a machine which follows the concept of *thinking* by itself and is able to execute actions based on own decisions. For its *thinking process*, machine AI uses techniques like statistical learning, machine learning or other techniques. [8] Today, machine learning finds application in many different tasks and helps users to simplify their daily life when translating text [9], navigating through streets [10] or by recommending movies they may like [11]. This paper focuses on machine learning in search engines and recommendation systems in combination with Natural Language Processing.

2.1.1. Natural Language Processing (NLP)

The term Natural Language Processing, like artificial intelligence, is not new. It began back in the 1950s as combination of artificial intelligence and linguistics and was distinct from information retrieval (IR). Early applications were found in word-for-word translations, which could be trained based on dictionaries, but had their difficulties in homographs – words which have different meanings even then identically spelled. [12] With faster hardware and modern algorithms, NLP has evolved rapidly during the past years. Today millions of webpages can be processed within a second when browsing Google for specific terms [13]. In this paper natural language processing plays a key role by analysing English phrased user queries and therefore gaining a contextual understanding of the apps. However, the language understanding is hereby built up with a machine learning model, which is trained unsupervised but fine-tuned supervised.

2.1.2. Supervised vs. Unsupervised Machine Learning

As mentioned in chapter 2.1, a machine learning model tries to improve it skills automatically. To do so, it needs specific data and a learning goal. There are two main methods for training a machine learning algorithm, supervised learning and unsupervised learning. Supervised learning, also called supervised classification learning, encourages a system so find rules and test hypothesises. When training a model unsupervised, every data-pair must be labelled. [14] This means, when trying to train a model with images to find the difference between dogs and fish, every input image has to be labelled either as *dog* or *fish*. The model then tries to find an algorithm for differentiating them by setting up hypothesises and predicts a label based on its hypothesis. Afterward, it cross-checks the predictions with the true label of the image. This process is then repeated till a minimum error between predictions and true labels is achieved.

Unsupervised learning on the other side does not need labels. If a machine learning model is trained unsupervised, its goal is to group similar items by finding *natural* groupings [15]. Looking at the mentioned example with *dog* and *fish* this would mean the trained model would get only the images as input, without any labels. Its goal would then be to let the model find the difference between the images

by its own and group images of fish into one group and images of dogs into another one, without calling them *fish* or *dog*.

The model in this paper is trained both, supervised and unsupervised. All models have been pre-trained unsupervised based on large corpora of data and afterwards fine-tuned supervised for the specific task of clustering queries according to the assigned app. However, when the model has trained an algorithm to predict the specific app name by inputting a query, this can be seen as search engine where a database of app names is searched to find the best app to a specific query.

2.2. Search Engine

When talking about search engine, there are different types of search engines. For this paper two types of search engines are relevant:

1. Query based Search Engine
2. Recommendation System

Explained in an example: a web-shop can suggest items to a customer using different methods. It can either provide a search function, which the customer can use to search for a specific product by using a query, or it can recommend items to the customer without the need of a query.

Recommendation System

A recommendation system does not need a query as input. The goal of a recommendation system is to find items or answers to questions its users did not even ask yet. To achieve that, different methods can be applied depending on the use case. Recommendation systems can work with a user's browsing history, his reviews of already purchased items or even by comparing a user to other users. Therefore, it is necessary to not only search the database for specific, but to understand the context and find similarities or habits. The recommendation system in this paper works with a sequence of used apps per user session and has the goal of predicting the next app the user will use based on those he already used.

Query Based Search Engine

Query based search engines on the other hand are usually defined as a program or process which should return the best results for a given query, key words, or related terms. This means, search engines are designed to go through a given database, a collection of items or similar and calculate a similarity of those items to a user-defined query. [16] The search engine in this paper is designed as dense retriever and has the goal of suggesting the best app to a specific query.

There are different options to evaluate the performance of machine learning models. As the data for both systems, the search engine and the recommendation system, is labelled, all methods in this paper compare predicted labels to true labels and calculate a score accordingly.

2.3. Scores

In this chapter different methods of evaluating machine learning models are discussed.

2.3.1. Accuracy

The accuracy of a model can be only calculated when the ground truth is known. It is a very simply measurement and shows how many of the predicted labels are correct. Therefore, the accuracy for every predicted label can be either True (100%) or False (0%). The systems accuracy is calculated by summing up all correct predictions and divide them by the number of total predictions *n*.

$$accuracy = \frac{1}{n}\sum_{i=1}^{n} accuracy_i$$

2.3.2. Mean Reciprocal Rank (MRR)

While for the accuracy only takes one predicted label and can therefore be only 0% or 100% for a single prediction, the Mean Reciprocal Rank is a metric which can be used when the output to a query is a list of results of which only one is correct. In difference to the accuracy, the *Reciprocal Rank* considers the rank of the correct result, meaning if the correct result is ranked first in the predicted list, the RR is 1, if it is ranked second, the RR is 0.5 and so on. Therefore, the higher ranked the correct label in the predicted list is, the higher is the reciprocal rank of the prediction. However, the *Mean Reciprocal Rank* is, as the name implies, the average reciprocal rank over *n* predicted lists. [17]

$$RR = \frac{1}{rank_i}, \qquad MRR = \frac{1}{|n|}\sum_{i=1}^{|n|} RR = \frac{1}{|n|}\sum_{i=1}^{|n|}\frac{1}{rank_i}$$

2.4. Transformer

When it comes to Natural Language Processing (NLP), Transformer has outperformed neural models such as recurrent and convolutional neural networks in natural language generation as well as in natural language understanding. [18]

Transformer is a model architecture which was introduced in 2017 by a Google AI Team and relies on an attention mechanism to find dependencies between input and output. This architecture allows more parallelization and therefore reached speeds up the training process which reached a new state of the art for translation tasks. It is designed to process an entire input at once and consequently train an understanding of the context for any word or syllable in the input sentence. Using an attention function, to be specific, a *Scaled Dot-Product Attention* function, the Transformer architecture calculates the dot product of a query and a set of key-value pairs and maps it to an output. Afterwards, the softmax function calculates the weights on the values. [19]

2.4.1. HuggingFace

HuggingFace[1] is a GitHub repository where users can upload pretrained transformer models and make them accessible for others. Currently, HuggingFace has over 70.000 GitHub Stars, making it the sixtieth most popular GitHub repository worldwide[2]. It provides access to almost 80.000 models in over 190 different languages.

HuggingFace started as a chatbot for teenagers in 2017 and became popular among developers in 2018. At that time, its developers have begun to share parts of their code for free which made it attractive for AI developers, including those from companies like Google and Microsoft. Today the company is valued at two billion US-dollars. [20]

2.4.2. Bidirectional Encoder Representations from Transformers (BERT)

Since its introduction in 2018, BERT has outperformed a majority of other models when it comes to NLP tasks [3]. BERT consists of two steps. Firstly, it is designed to learn contextual representations of text by being pretrained on large amounts of texts. This process is called *pre-training* and happens unsupervised. After the *pre-training* is done, BERT can be *fine-tuned* for a specific task. This is done

[1] https://huggingface.co/models, accessed 10th October 2022
[2] https://gitstar-ranking.com/repositories, accessed 10th October 2022

by adding a classification layer on top of the representation if the first token, the CLS token. To achieve that, BERT is initialized with pretrained parameters which are changed slightly during the *fine-tuning* process. As shown in Figure 1, to every input for a BERT model a [CLS] token needs to be added. CLS hereby stands for *classification*. The [SEP] token on the other hand stands for *separation* and shows where one sentence ends, and another sentence starts. After the input has been processed, all for the classification necessary information gets stored in the CLS-token as 768-dimensional vector. The whole input is finally classified based on this single vector. [4]

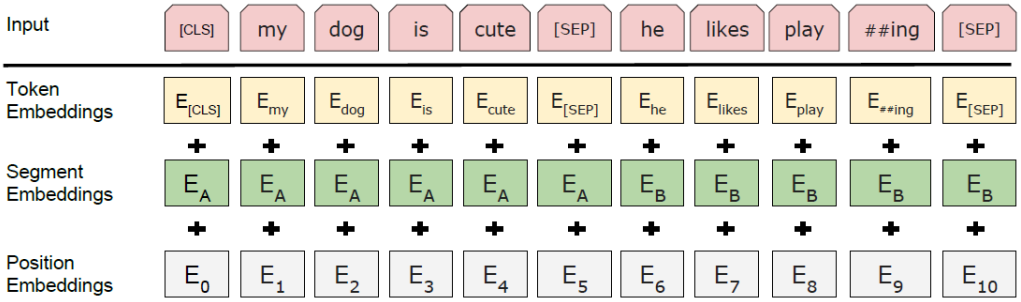


Figure 1 - Representation of BERT input. [4]

2.5. The Environment

When it comes to developing code, the first question is always which programming language to choose. Since the goal is not to develop everything from scratch but to use existing frameworks, it is necessary to check which options there are. The most used frameworks at the *Center for Intelligent Information Retrieval* for machine learning tasks are Pytorch[3] and Tensorflow[4], both used in Python. With Anaconda[5] as the most popular data science platform, providing uncomplicated framework imports and a broad offer of extensions, all code developed during this project was done in Anaconda as JupyterNotebook with Python as programming language and a mixture of PyTorch and Tensorflow.

[3] https://pytorch.org/, accessed 18th October 2022
[4] https://www.tensorflow.org/, accessed 18th October 2022
[5] https://www.anaconda.com/, accessed 18th October 2022

3. The Idea of Joint Search and Recommendation

Although search engines and recommendation systems can be seen as two sides of the same coin, the techniques used for developing and training them are different. They both should help users to find the information they need, which often can be in both systems identical. Still, the methodology of searching is different in both. While search engines rely on information retrieval approaches such as learning to rank, a common approach for recommender systems is collaborative filtering. Recently, Hamed Zamani and W. Bruce Croft have shown that search and recommendation can be designed jointly and therefore bring improvements to both systems [2].

The goal of this project is to apply those existing approaches to different projects and thereby extend them with Transformers like BERT, the current state of the art deep learning model. When it comes to *Natural Language Processing (NLP)* tasks, BERT has brought a big improvement since its introduction in 2018 [3]. Therefore, the aim of this paper is to replace the *Bag of Words* approach from [2] with BERT and see if it is suitable to learn a context for search and recommendation.

Since recommendation systems and search engines are trained on different data, the choice of datasets that can be used for this work is very limited. One dataset, which consists enough data to train both, search and recommendation, is the dataset created as part of the *Context-aware Target Apps Selection and Recommendation for Enhancing Personal Mobile Assistants* [21], which consists of query-app combinations and recordings of app sessions. As shown in Figure 2, the search engine needs queries and apps to be trained on, while the recommendation system needs apps and sessions of apps. However, both have the goal to predict a list of apps depending on the information they have. The search engine should predict the best app for the given query, while the recommendation system predicts the next app the user will use based on the apps he already did use.

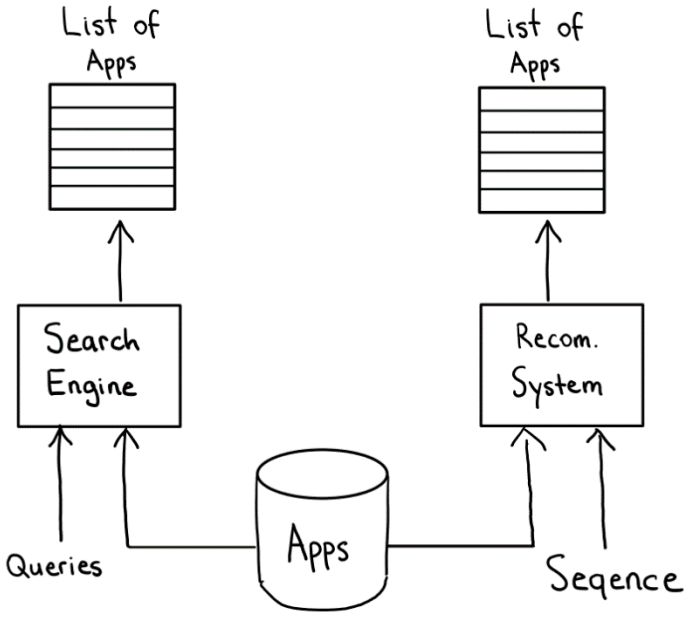


Figure 2 - Model of joint search and recommendation for given dataset

The idea is to design both systems independently but with a shared BERT Model. To achieve this, this project was divided into three steps which can be seen in Figure 3. The first step is to explore whether BERT is suitable at all for learning a context to the applications using the queries and apps during the training process. If this experiment is positive, the second step is to train a dense retriever that matches and fine-tunes a pre-trained BERT model to the app and query combinations. The goal of the Dense

Retriever is to predict a list of apps for a specific query, where those apps that best match the query are listed at the top. For example, if a user searches for "spiderman movie", there is a high probability that the user wants to watch the movie. Therefore, the Dense Retriever should suggest streaming services like *Netflix* or *Hulu*. This would complete the search itself.

The third step is then building a recommendation system. The input of the recommendation system is a sequence of apps, where the next system should predict the application, a user will open next based on those he already used in his surfing session. Meaning, if the user's session contains *Disney Plus*, *Prime Video* and *Hulu*, he is probably searching for a movie and would like to open *Netflix* next. The gained knowledge from the Search engine could bring a profit to this task. When training a BERT model with queries and apps, the goal for the model is to learn a context to an application. Therefore, it could learn from the queries that *Disney Plus*, *Hulu* and *Netflix* are all streaming services. Therefore, the idea is to check if this knowledge can help in predicting the next application. To achieve this, the trained BERT model from the search engine should be used as app encoder for the recommendation system. Every app name from a session is encoded using the trained BERT model and using the encoded app names, the next app should be predicted.

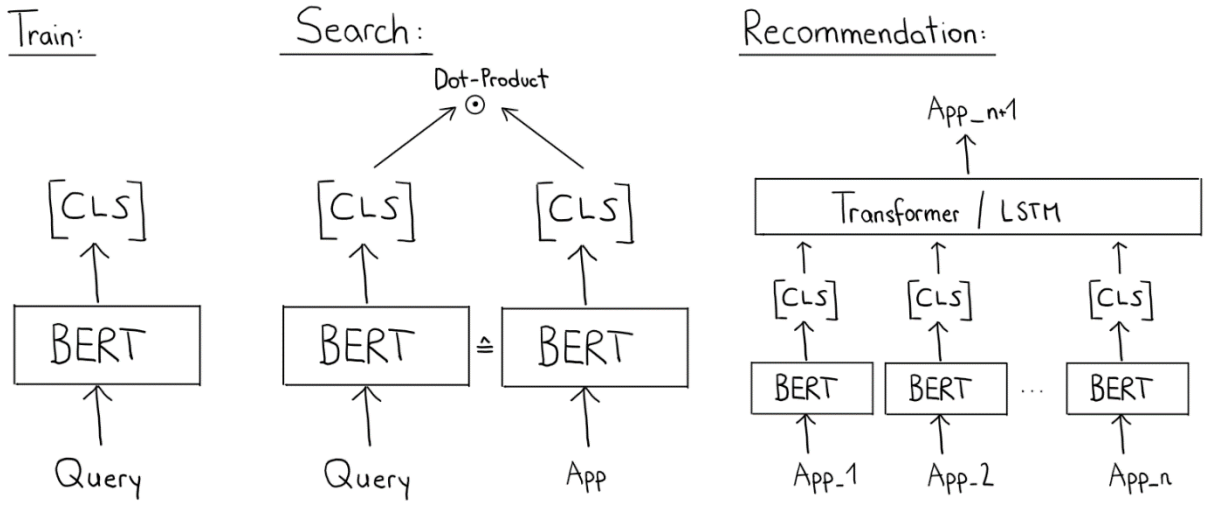


Figure 3 - How to join Search and Recommendation

4. The Dataset

As mentioned in the previous chapter, to build a *Joint Search and Recommendation* System, a dataset is needed which can be used for both, search and recommendation. As described in chapter 0, a query-based machine learning search engine needs a collection of queries and matching items for being trained accordingly. However, the recommendation system cannot be trained by query and item combinations as recommendation are used without queries. Summing up, this means that a dataset is needed which has both, query-item combinations and either a sort of user-item interactions or a sequence of items.

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani and W. Bruce Croft build a dataset for *Context-aware Target Apps Selection and Recommendation for Enhancing Personal Mobile Assistants* which meets those requirements. The dataset is divided into two subsets, the *"In Situ collection of cross-App mobile Search"* (ISTAS) and the collection of a *"Large dataset of Sequential mobile App usage"*. [21]

4.1. ISTAS Dataset

For collecting the ISTAS data 255 recruited participants let a custom-built app, called uSearch, running on their smartphones for a minimum period of one day. As shown in Figure 4, the app's user interface is split into three sections:

1. a list of apps installed on the phone, where the user has to check which app he used
2. a text box to enter the query the users executed in the selected app
3. a short survey to get a unique ID and some demographics and backgrounds of the user

In addition to those query-app-combinations, the app collected data via GPS, accelerometer, gyroscope, ambient light, WiFi and cellular sensors. For their participation all users got paid $0.2 per query entered into the application. [21]

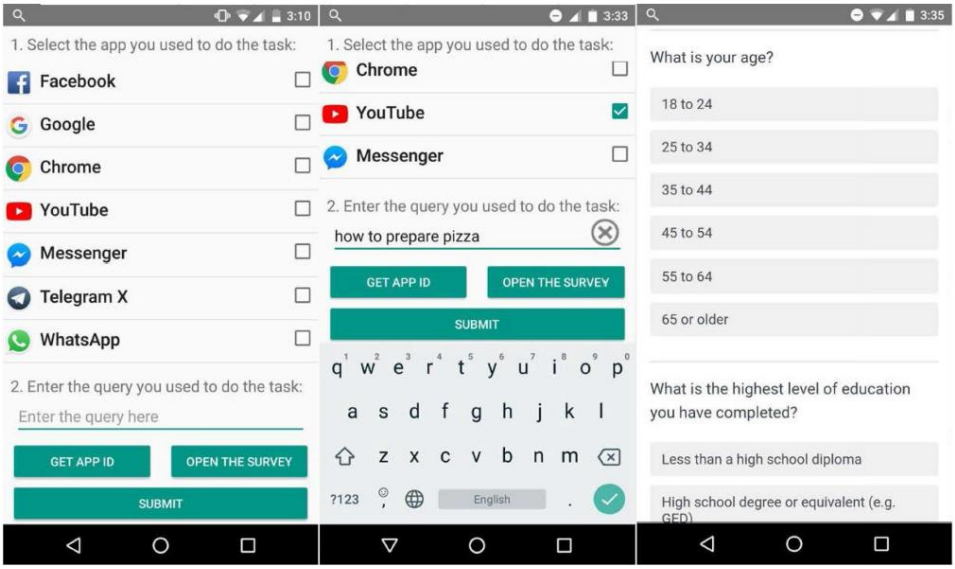


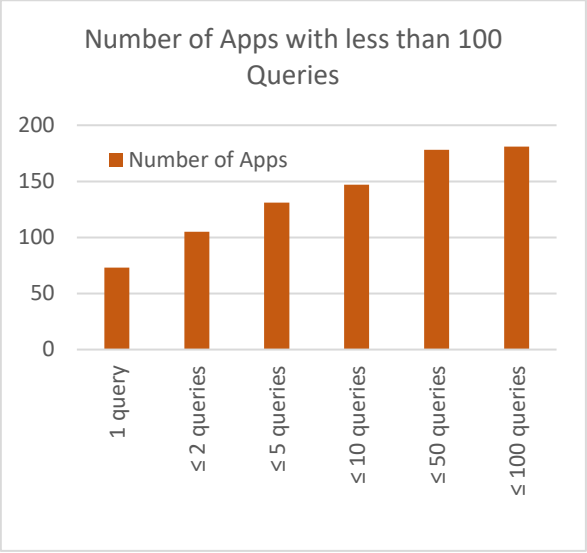
Figure 4 - Demonstration of uSearch interface as App [21]

All collected data was finally stored as JSON file containing entries with associated timestamps, UserIDs, Queries, Apps and AppUsages. Examples for those entries can be seen in Table 1. In total there are 6877 recorded queries in 192 different apps from 255 users.

	timestamp	UserID	Query	App	AppUsages
995	2018-04-16 14:41:01.883	126	why are the disney cartoons in spanish today	chrome	{'Duration': {'android_system': 63575, 'badgep...
997	2018-04-16 17:01:06.085	246	weather hope mills	google	{'Duration': {'ally_mobile': 50046, 'android_s...
998	2018-04-16 17:03:38.204	246	kakuriyo	chrome	{'Duration': {'ally_mobile': 50046, 'android_s...
999	2018-04-16 18:25:43.965	246	how much are tea candles	google	{'Duration': {'ally_mobile': 50046, 'android_s...

Table 1 - example for ISTAS entries

However, the uneven distribution of the 6877 queries over the 192 apps is a challenge for training a machine learning algorithm based on this data. As shown in Figure 5 and Figure 6, for more than one third of the apps (73 of 192) only one query was recorded while 49,2% of all queries (3386 of 6877) have been executed in either *google* or *chrome*, which are almost identical apps when it comes to browsing the internet via query.



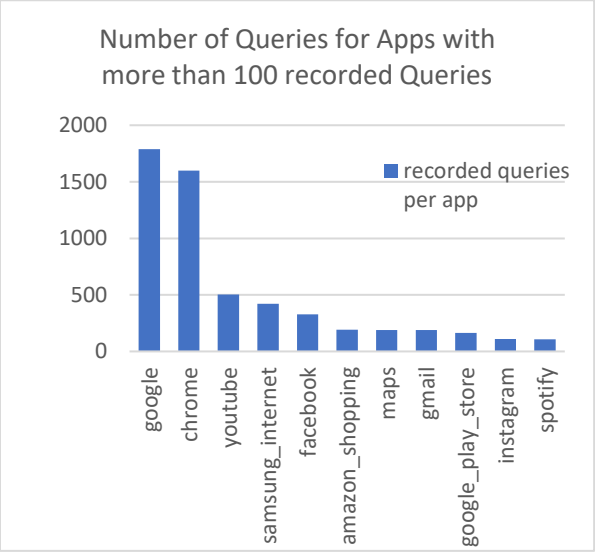


Figure 5 - (right) Distribution of Apps with more than 100 Queries

Figure 6 - (left) Queries / Apps distribution

4.2. LSApp Dataset

The other part of the given dataset is the LSApp dataset. LSApp is short for *Large dataset of Sequential mobile App usage* and was collected using uSearch[6] data collection tool. Over eight months data of 292 users was collected of which 255 users were the same users from the ISTAS dataset. The other 37 users did not submit a single valid query and are therefore not included in the app query dataset. While collecting the data, many repeated app usages within ten seconds have been recorded. Therefore, the authors defined multiple app usages of one app within ten seconds as one single entry. The median of unique apps per session is two while the mean session time length is 5:26. [21]

In total there were almost 600,000 app usages recoded, distributed over around 76,000 sessions. While the ISTAS dataset contains 192 recorded apps, the LSApp dataset has only 87 unique apps. 51 of these 87 apps could be matched to the 192 apps, for the other 36 apps no record in the ISTAS dataset was found.

[6] https://github.com/aliannejadi/uSearch

5. Search Engine

This section describes the approaches to optimize search via machine learning for the ISTAS dataset. The search in this section is defined as predicting the correct app based on a query. The benchmark for this task is the best score of the *Context-aware Target Apps Selection and Recommendation for Enhancing Personal Mobile Assistants* [21] paper. This means, the goal is to see if a machine learning model with BERT can achieve similar or better results on the same dataset as the best model in the according paper. The authors of the mentioned paper have documented the performances of 21 different methods using MRR and nDCG@n scores. The best result for the ISTAS dataset has been achieved using CNTAS-pairwise with a MRR score of 0.5637, followed by the NTAS-pairwise with 0.5257 [21]. Therefore, an MRR score of 0.5637 is the benchmark here.

5.1. Multilabel Classification Model using Simple Transformer

The first approach to check if BERT qualifies for an app prediction was to build a multilabel classifier using a pretrained BERT model using *Simple Transformer*. *Simple Transformer*[7] is a library for natural language processing tasks which is built using HuggingFace. As the name implies, Simple Transformer is designed to use basic functions of HuggingFace with a few simple lines of code. With this simplification also comes a restriction of the HuggingFace functionality. Although a fully functional multilabel transformer was built with a few lines of code, a training based on a *Mean Reciprocal Rank* could not be implemented.

5.2. Classification using HuggingFace and PyTorch

To use the full functionality of HuggingFace, the idea of building a prediction with Simple Transformer was discarded. Instead, the next approach was to build a model using HuggingFace and Pytorch. At this stage, three different, pretrained BERT models have been compared:

- bert-base-uncased[8]
- roberta-base[9]
- distilbert-base-uncased-finetuned-sst-2-english[10]

[ERGEBNISSE FÜR ROBERTA UND DISTILBERT EINFÜGEN]

Using HuggingFace and Pytorch, the next step was to check which pretrained BERT model is the best for the given dataset. Therefore, a model was built which predicts an app name based on the input query. All models have been evaluated using the *Mean Reciprocal Rank* (see chapter 2.3.2) so they can be compared to the models from the *Context-aware Target Apps Selection and Recommendation for Enhancing Personal Mobile Assistants* [21] paper.

Picking the best pretrained model can be challenging as the results do not only depend on the model itself, but also on the fine-tuning parameters. Since those settings strongly depend on the dataset, there is no one-size-fits-all setting a model, which means the best setting needs to be determined experimentally. If you go by the original BERT introduction paper, a batch size of 8, 16, 32, 64 or 128 and two to four epochs are a good starting point for finetuning a BERT model [4]. As shown in Figure 7, the mean reciprocal rank is higher when using four epochs than with two or three. Therefore, more

[7] https://simpletransformers.ai/about/
[8] https://huggingface.co/bert-base-uncased
[9] https://huggingface.co/roberta-base
[10] https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english

different options have been tried out than suggested by the authors of the BERT paper. The results show that the model performs best when trained with eight epochs.

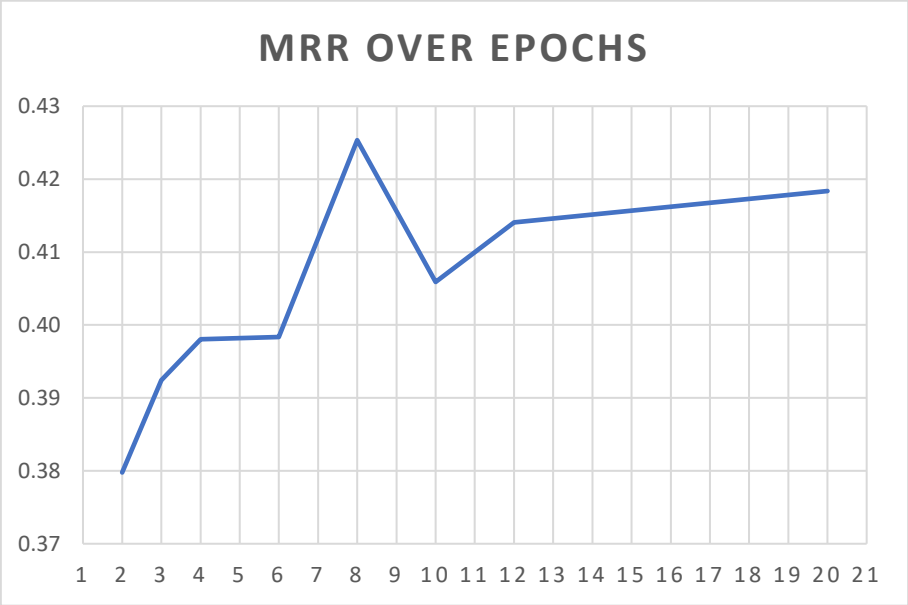


Figure 7 - MRR over epochs (with batch-size: 8)

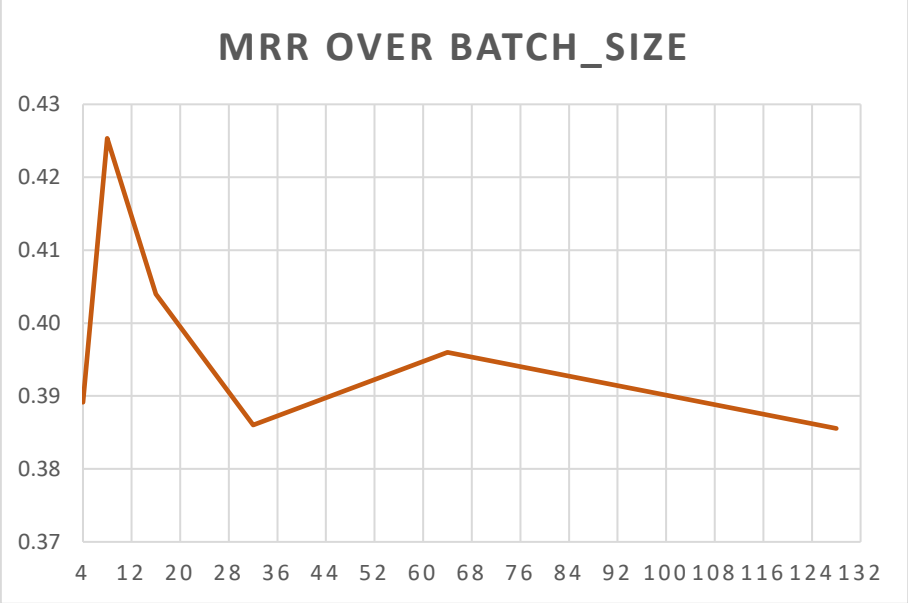


Figure 8 - MRR over batch sizes (with epochs: 8)

The next step was to research which batch size fits best for the given dataset. As the first experiments have shown that eight epochs work best, different settings with batch sizes between 4 and 128 have been tested with constantly using eight epochs. As shown in Figure 8, the best final setting for training the model is with *epochs: 8* and *batch_size: 8*.

5.2.1. Challenges caused by uneven distribution of data

As shown in Figure 9, when training a machine learning model, a percentage *n* is used for training the model and *1-n* for evaluating it. This means when training the given model here, one query-app-pair can either be used for training or for testing. Considering that 73 of 192 apps in the given dataset have

only one recorder query, the can either never be trained for those apps or if it was trained with them, they will not appear in the evaluation. This makes it impossible to get a correct evaluation when trying to predict those apps.

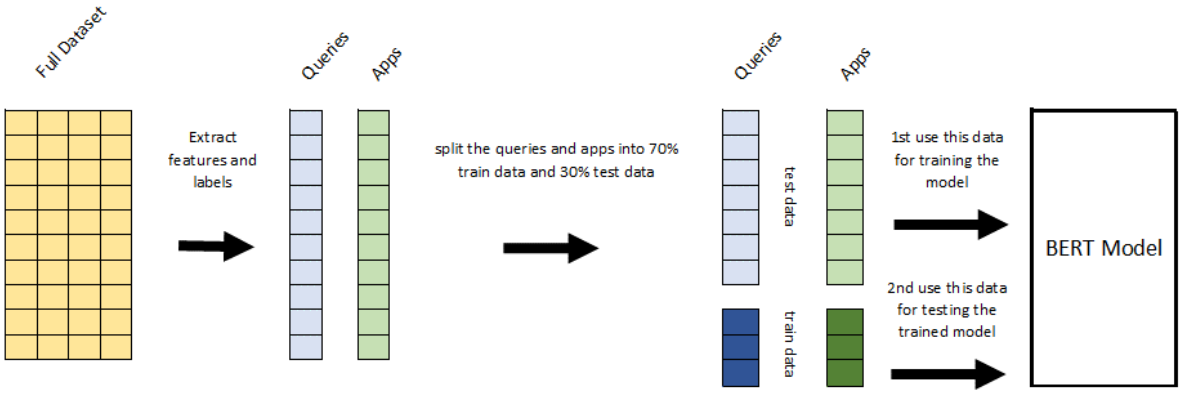


Figure 9 - Train-test split explained for given dataset

Another challenge caused by the uneven distribution of data is the optimization goal of the model. Since the model is designed to train for the best accuracy or MRR, it could happen that is trains itself mainly on those apps which have the most query-app-pairs. Since 49.2% of all queries have been executed in either *google* or *chrome*, a model could achieve an accuracy of almost 50% just by being trained to *google* and *chrome*, which would be easier than to train it to the other or to the other 190 apps.

5.2.2. Data pre-processing

To find a solution for handling the uneven distribution of data, different options have been tried. As some apps have over 1000 recorded queries while almost 75% of the apps have less than 50 queries, the first idea was reducing the number of queries for those apps with a high occurrence. To achieve that, a function has been developed which trims the number of queries for all apps to a specific amount. When this parameter, called *trim_to*, is set to *m*, only *m* query-app-pairs per app stay in the dataset.

As mentioned previously, if an app has only one recorder query, it cannot occur during training and evaluation and therefore, no conclusions can be drawn for those apps. To solve this problem as well, an option for *boosting* queries has been implemented. When the parameter *boost_to* is set to *x*, query-app-pairs for apps with less than *x* queries get duplicated in the dataset.

boost to \ *trim to*	**50**	**100**	**200**	**400**
50	0.859	0.843	0.795	0.756
100		0.933	0.908	0.881
200			0.945	0.938
400				0.961

Table 2 - change of the MRR based on the options boost_to and trim_to

Using the parameters *trim_to* and *boost_to* the dataset can be a little balanced. If an app on the one side has over 1000 entries but the parameter *trim_to* is set to 200, 200 of those 1000 query-app-pairs are picked randomly and used for training and testing the model. On the other, if an app has only two recorded queries but the *boost_to* parameter is set to 50, every query-app-pair for this app is replicated 25 times. These options have two positive effects on finetuning the model: on the one hand, this makes the model more sensitive to those apps with fewer queries, but on the other hand, it also prevents the

model from always predicting *google* or *chrome* just because these two apps account for almost 50 percent of the original data. Additionally, those two options also give other insides on the model behaviour like the impact of apps with high occurrences on the predicting accuracy or change of the MRR score depending on how balanced the input data is.

Two main conclusions can be drawn from the figures presented in Table 2. First, it shows that the prediction accuracy is not necessarily getting better with more input data. Looking at the setting *boost_to = 50* it can be seen that the performance gets worse when the apps with a high number of recorded queries are trimmed less. However, the *boost_to* and the *trim*_to parameter get higher, the model generally performs better. Therefore, it can be said that the model performs better when the distribution of queries per app is well balanced. However, it should be noted that by increasing the *boost_to* parameter the system tends to overfitting. If a query gets replicated 200 times, the model gets sensitive to that specific query and will most probably always classify it correctly, but a slightly changed query could already be wrong classified which is a strong indicator for overfitting. Nevertheless, if there are only one or two recorded queries for an app, the model in general cannot learn much context about it.

5.3. Dense Retriever

As the results from chapter 5.2 have shown that BERT is indeed able to learn a context between queries and apps, the next step is to build a search engine using BERT. Dense retrievers retrieve items using a usually high-dimensional representation. This representation can be learned by fine-tuning pre-trained models like BERT with queries and documents. [22]

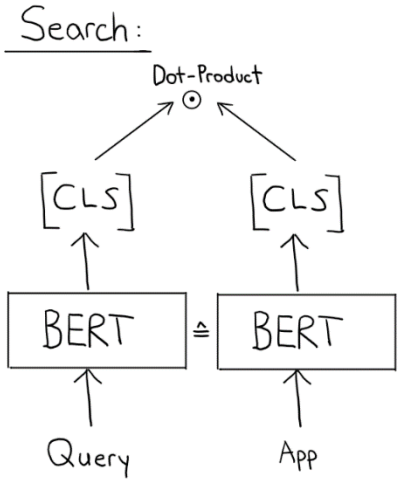


Figure 10 - Dense Retriever with BERT and dot product

The design of the built dense retriever is shown in Figure 10. Thereby, the BERT model is used as encoder for the inputs, which in this case are queries and apps. In addition to queries and apps the dense retriever receives a label which marks if the query-app pair is a positive or negative sample. Recent studies have shown that dense retriever results tend to be improved by adding some negative samples during the training process [23] [24]. According to the literature a common loss function for training dense retrievers is contrastive loss [25] [26].

As the dataset has no negative samples it was necessary to generate some before training the dense retriever. Therefore, a function was implemented which generates negative from the existing dataset and adds them to the train-dataset. Negative samples are created per app, meaning when given an app name like Netflix, a random query is taken which does not have Netflix as app assigned. However, as the dataset has 192 apps, there are a few apps which are very similar to each other. Therefore, a query like

Spiderman Movie could have been executed in Netflix but also in Hulu or IMDB. To prevent the creation of negative samples where the query and app combination would make sense, all 192 apps were divided manually into 10 categories.

Category Id	Categories	# apps in category
0	Browser	13
1	Streaming	32
2	purchase and living	30
3	social media	15
4	default apps / other apps	32
5	communication	19
6	finance	8
7	health	7
8	navigation & travel	10
9	information	14
10	games & other entertainment	12

Table 3 - app categories with number of apps per category

The category Browser for example contains apps like Google, Chrome, Firefox, Samsung_Internet and others, while the category social media has apps like Facebook, Instagram, Twitter etc. The distribution of apps per category can be seen in Figure 11. Using those categories, it is guaranteed that no query from an app in the same category is taken as negative sample for another app in the same category.

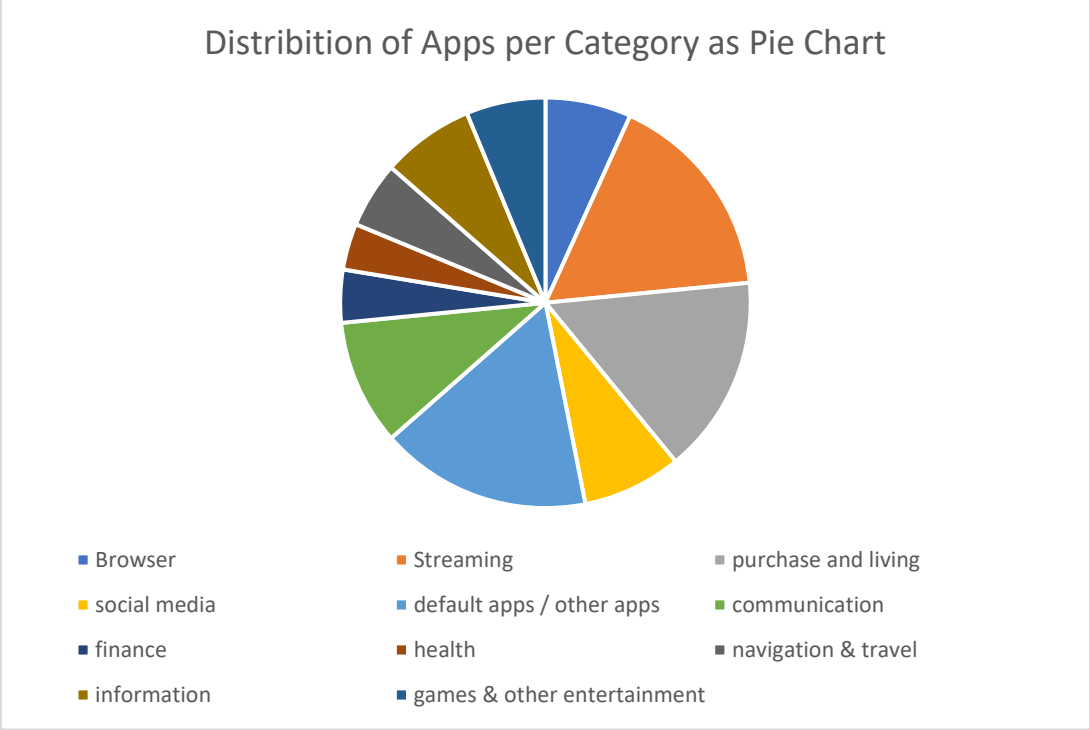


Figure 11 - Distribution of Apps per Category

Following different pre-trained models have been tested in combination with a dense retriever and the given dataset:

distilbert-base-uncased-finetuned-sst-2-english11

distilbert-base-uncased-finetuned-sst-2-english is a checkpoint based on distilbert-base-uncased but finetuned on the Stanford Sentiment Treebank (SST). Distilbert is, as the name implies, a lighter version of BERT. As the size of the data sets in machine learning tasks is crucial for the results, the models are in principle fed with more and more data. Since there is no one-fits-all solution for the parameters of a machine learning model, the training is mostly based on try and error. In order to accelerate the training of the models, investments are made in graphics cards on the one hand, and on the other hand, the models themselves must be designed to be as performant as possible. Since BERT is relatively computationally intensive, distilbert was created. While the size of distilbert is only around 60% of the original BERT model, its language understanding performance is around 97% compared to BERT. This makes distilbert around 60% faster and still very performant. [27]

The Stanford Sentiment Treebank12 is a dataset containing sentences for English sentiment language. It contains 215,154 unique phrases which were all annotated by 3 human judges. As the phrases are in English and the queries from the dataset in this paper are also mainly in English, the idea was to see if advantage can be taken from a model which is already trained on understanding English context.

microsoft/mpnet-base13 and all-mpnet-base-v214

MPNet stands for Masked and Permuted Pre-training for Language Understanding. It is a model which extends the functionality of BERT and XLNet, a Generalized Autoregressive Pretraining for Language Unterstanding [28], and takes position information into consideration. It is pre-trained on around 160GB of text corpora. While BERT uses masked language modelling (MLM) and XLNet permuted language modelling, MPNet uses a mixture of both. [29] In this paper two different MPNet models have been trained on the given dataset: mpnet-base and all-mpnet-base-v2. As mentioned, mpnet-base is pretrained on 160GB of text corpora, while the all-mpnet-base-v2 model checkpoint is pretrained on mpnet-base and fine-tuned on one billion sentence pairs dataset which come mostly from Reddit comments.

multi-qa-distilbert-cos-v115

The multi-qa-distilbert-cos-v1 checkpoint is based on a sentence transformers model and was designed for a semantic search, to be precise, on Sentence-BERT (SBERT). Sentence-BERT uses Siamese and triplet network structures to extend the BERT network and therefore improve the performance when it comes to similarity recognition in sentences. When searching for a similar sentence in a set of 10,000 sentences, SBERT needs around five seconds to find the most similar one while BERT takes about 65 hours. The comparison hereby is done using by calculating a cosine-similarity of the dense vectors. Therefore, the model is optimized to arrange similar sentences in a vector space near to each other and calculate the similarity of the vectors for finding the nearest neighbours. [30] Since the goal of the dense retriever in this paper is also to find similarities in the recorded queries per app, the sentence transformer architecture is similar to the dense retriever architecture.

random predicter and static ranker

Sometimes performance results can be hard to interpret or even vacuous when given to little context. For a better orientation about how good or bad a model is, two simple predicter have been built for

[11] https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english?text=I+like+you.+I+love+you
[12] https://huggingface.co/datasets/sst2
[13] https://huggingface.co/microsoft/mpnet-base
[14] https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[15] https://huggingface.co/sentence-transformers/multi-qa-distilbert-cos-v1

comparison. Firstly, a static predicter has been implemented. When entering a query, independently of the input the predicter always predicts the same list of apps. The first five apps on this list can be seen in Table 4. Although there is no prediction logic behind this, it can be used as orientation point about how good a trained model is compared to a static prediction.

prediction rank	app name
0	google
1	chrome
2	puffin
3	youtube
4	file_manager
5	google_play_store

Table 4 - static predicter prediction list

Secondly, a random predicter has been built which predicts any random app from the 'app' column of the given dataset. Although the predicter pics a random line, the prediction is also based on the distribution of app usages since it picks a random line from the 6877 app query pairs. Although this also does not implement any query-based logic, it can be informative index as the minimum goal of the search engine should be to beat those two predicters.

Results:

As shown in Table 5, the trained models were not able to outperform the existing solution. Nevertheless, this does not mean that BERT is not able to bring any improvement in this field of application since only a few settings have been tested. All models were trained using two epochs, a batch size of eight, a train / test ration of 90 / 10 and a positive / negative sample ration of one, meaning for every positive sample one negative sample was generated. The training was performed on a Tesla T4 GPU and the training duration was between two and ten hours. Therefore, due to time constraints, only a certain contingent of settings could be tested.

model	mrr
distilbert-base-uncased-finetuned-sst-2-english	0.1398
microsoft/mpnet-base	0.0266
all-mpnet-base-v2	0.2453
multi-qa-distilbert-cos-v1	0.2628
random predicter	0.2766
static predicter	**0.4280**

Table 5 - dense retriever training results

In total, the fine-tuned microsoft/mpnet-base model performed worst on with the configured settings on the given dataset. Using distilbert-base-uncased-finetuned-sst-2-english the results were already over five times better while the models using all-mpnet-base-v2 and multi-qa-distilbert-cos-v1 performed best. However, all trained models were worse then the static and random predicters and were therefore not able to compete with TempoLSTM (MRR: 0.6898) from the *Context aware Target App Aelection and Recommendation* paper [21].

6. Recommendation System

Although the results from the search engine were not satisfactory, the potential for improvement is still there. Therefore, to complete the overall setting, the next step was to develop a recommendation system based on the trained BERT model. Although the recommendation system uses the trained BERT model to encode the app names, the data used in this part is from the LSApp dataset which is, as mentioned in chapter 4.2, also based on the same paper as the ISTAS dataset [21].

However, the content of the datasets is very different even though they were both created by the same user groups and most probably on the same devices. While the ISTAS dataset has in total 192 apps recorded, the LSApp dataset has only 87 used unique apps in the recorded sessions, which implies that the data for these two datasets either was not collected at the same time, the app for recording the sessions was not able to record everything or the users did not enter the queries they submitted on the device where the sessions were recorded. Additionally, of those 87 recorded unique apps, only 51 could be matched to the 192 apps from the ISTAS dataset. Therefore, the BERT model used in the dense retriever was not able to train a context for over 40% of the apps which occur in the LSApp dataset.

6.1. Data pre-processing

In total almost 600,000 app usages were recorded. App usages in the mentioned paper [21] are opening and closing an app. The focus on this paper is to predict the next app based on already used apps in one session. Therefore, the information about opening or closing an app is irrelevant at this point and was filtered. Afterwards, also all entries where filtered where an app was used multiple times in one session without an app in between. After filtering duplicate usages as well as the opening and closing information per session, from almost 600,000 records only approximately 182,000 records were left. As shown in Figure 12**Error! Reference source not found.**, the majority of those 182,000 records has only one recorded app per session. The mean of apps used per session is 2.38, while the median is 2.0.

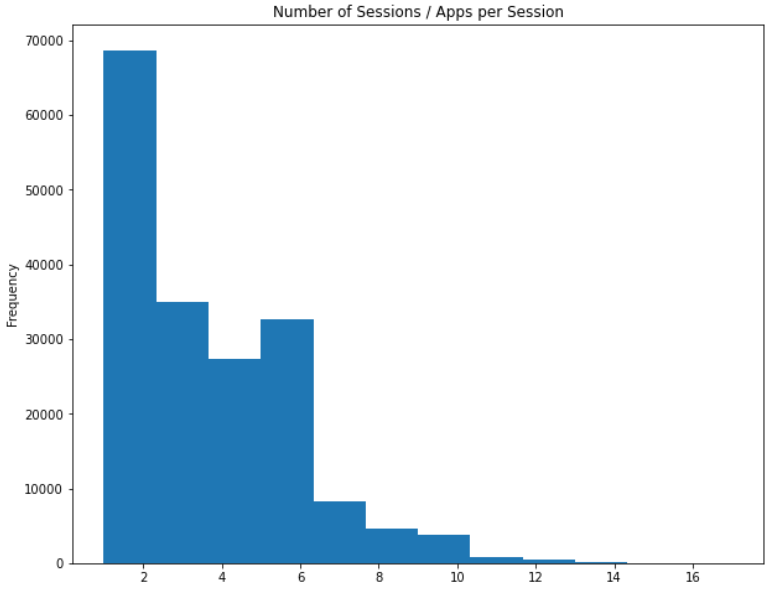


Figure 12 - Number of sessions per used apps per session

As the next app should be predicted, all sessions with only one recorded app are useless for training the system. Consequently, 29,955 of 76,247 recorded sessions cannot be used for training leaving the train dataset with 46,292 sessions.

6.2. CLS-Token based Recommender

Since the results from 5.3 still need to be improved to gain an improvement from training a search engine with BERT for this specific dataset, designing an advanced recommender on the current results is not possible. Nevertheless, this research has still shown that BERT is able to gain contextual knowledge of the apps. Consequently, the next step for is to research how a simple recommender based on the CLS tokens of the trained BERT model performs.

As explained in chapter 2.4.2, when trained, a BERT model computes a multidimensional dense / vector which contains the information needed for classification and is stored on the CLS-Token position. Thus, similar words are nearer neighbours in the vector space of calculated CLS-Tokens. Following that, when a BERT model is trained, it can be used to represent similar words as vectors. The recommendation system in this chapter is built based on this principle. All app names are transformed into a vector using the CLS-Tokens of a trained BERT model and stored into a lookup table. When recommending, the system takes multiple apps as input, replaces every app name with the corresponding vector from the lookup table and performs a mathematical operation on all input vectors. Afterwards, it looks for the most similar app in the vector space and takes it as prediction.

When it comes to predicting the next app, the number of already used apps per session is crucial. If a session has only two recorded apps, the predicter would have to predict the second app only based on the one previously used app, which is little information for training a predicter. However, the higher the number of input apps for the recommender is set, the less data remains for training / testing. Therefore, different modes have been implemented for predicting and evaluating. The recommendation is configured using two parameters: *n_apps* and *min_apps_per_session*.

n_apps: this parameter sets the number of apps the recommendation system gets. The last used app is always taken as label / ground truth and the others as input for the prediction. When *n_apps* is set for example to 4, the recommendation system tries to predict the fourth used app based on the first three apps used.

min_apps_per_session: this parameter sets the number of apps needed minimum for predicting and evaluating. It removes all sessions with less than *min_apps_per_session* apps recorded.

n_apps	min_apps_per_session	operation	mrr
1	1	Σ	0.1348
1	1	Π	0.1348
2	2	Σ	0.1349
2	2	Π	0.1349
3	3	Σ	0.1268
3	3	Π	0.1345
4	4	Σ	0.1351
4	4	Π	0.0541

Table 6 - mrr scores for CLS-token based simple recommender

As shown in Table 6, the simple recommender was not able to outperform the existing solutions from [21] by simply searching for the most similar apps. Nevertheless, for a fair competition with the existing solution, a TempoLSTM should be added on top of the current implementation, which takes the CLS encoded app names as input and predicts the next app based on them. Due to time constraints this implementation has not been implemented during this research period.

7. Conclusion and Outlook

Compared to the best existing search engines, NTAS-pairwise (MRR: 0.5257) and CNTAS-pairwise (MRR: 0.5637), the BERT-based dense retriever in this paper was not able to maintain the performance (MRR: 0.2628). Consequently, the recommendation system built on base of the trained BERT model (MRR: [TODO]) was also not able to compete with TempoLSTM (MRR: 0.6898) from [21].

Although the models built and trained in this paper were not able to maintain the performance of neither the search engine nor the recommendation system compared to other implementations [21], it has once again proven BERT's ability of gaining contextual knowledge with little data by giving a moderate performance on the given dataset. However, there are still plenty of other possible configuration possibilities for BERT systems which still could outperform the existing solutions.

During the research and implementation work of the documented systems and models, another idea regarding joint search and recommendation came up. This paper did extend similar solutions to Zamani's and Croft's research [2], which shows how search and recommendation can be designed jointly. However, with this solution the systems may be able to benefit from each other during their training but afterwards, still two separate systems are needed, one for searching and one for recommending. By extending the work from this paper an additional solution could be research where search and recommendation are designed as one final model, as shown in [ABBILDUNG]. The mentioned concept could be then trained and used jointly for both, search and recommendation. However, the first step of establishing this system would be to find a setting for the given search and recommendation in this paper which can maintain or even outperform existing solutions. Afterwards, both models could be designed as one pipeline and evaluated together.

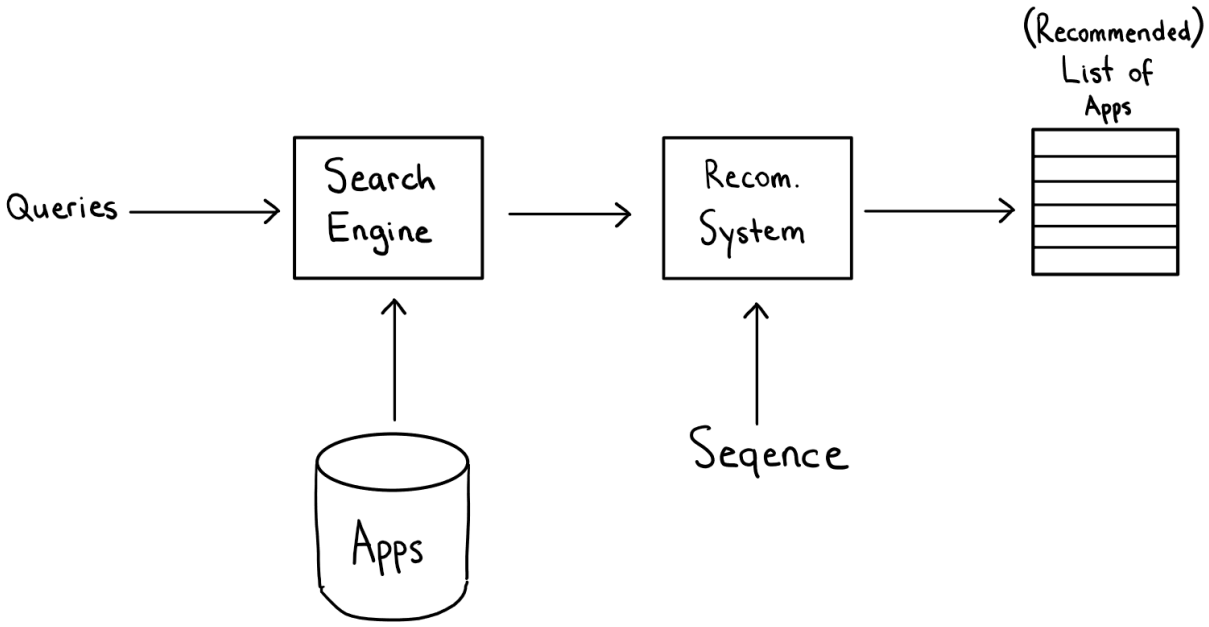


Figure 13 - Concept of Joint Search and Recommendation Model

All in all, this work has shown possible implementation possibilities for a *Joint Search and Recommendation* system in combination with *BERT*. Although the built models did not outperform existing solutions, the work done sets a direction for a joint search and recommendation in combination

with transformers. The next steps for continuing this work would be to test other pretrained BERT models with different settings to improve the prediction accuracy of the dense retriever. Afterwards, the recommendation can be built by using the fine-tuned model and adding another transformers or LSTM layer on top.

References

[1] E. Horvitz, "Machine Learning, Reasoning, and Intelligence in Daily Life: Directions and Challenges," 1 12 2006. [Online]. Available: https://www.semanticscholar.org/paper/Machine-Learning%2C-Reasoning%2C-and-Intelligence-in-Horvitz/fe1279cc8fe08bb143ad02bccd3561a435e8f27e?sort=is-influential. [Accessed 26 10 2022].

[2] H. Zamani and W. B. Croft, Learning a Joint Search and Recommendation Model from, Amherst: University of Massachusetts Amherst, 2020.

[3] D. D. E. P. Ian Tenney, "BERT Rediscovers the Classical NLP Pipeline," *Association for Computational Linguistics,* vol. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, p. 4593–4601, 2019.

[4] M.-W. C. K. L. K. T. Jacob Devlin, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *Association for Computational Linguistics,* Vols. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), p. 4171–4186, 2019.

[5] A. Ezen-Can, "A Comparison of LSTM and BERT for Small Corpus," 11 09 2020. [Online]. Available: https://arxiv.org/abs/2009.05451. [Accessed 26 10 2022].

[6] s. si, R. Wang, J. Wosik, H. Zhang, D. Dov, G. Wang and L. Carin, "Students Need More Attention: BERT-based Attention Model for Small Data with Application to Automatic Patient Message Triage," *Proceedings of the 5th Machine Learning for Healthcare Conference,* vol. 126, pp. 436-456, 2020.

[7] T. Mitchell, B. Buchanan, G. DeJong, T. Dietterich, P. Rosenbloom and A. Waibel, "MACHINE LEARNING," *Annual Review of Computer Science,* vol. 4, pp. 417-433, 1990.

[8] N. Kühl, M. Goutier, R. Hirt and G. Satzger, "Machine Learning in Artificial Intelligence: Towards a Common Understanding," in *Hawaii International Conference on System Sciences*, Hawaii, 2019.

[9] A. Yulianto and R. Supriatnaningsih, "Google Translate vs. DeepL: A quantitative evaluation of close-language pair translation (French to English)," *AJELP: Asian Journal of English Language and Pedagogy,* vol. 9 No. 2, pp. 109-127, 2021.

[10] G. Lewis-Kraus, "The Great A.I. Awakening," The New York Times Magazine, New York, 2016.

[11] F. Amat, A. Chandrashekar, T. Jebara and J. Basilico, "Artwork personalization at netflix," in *RecSys '18: Proceedings of the 12th ACM Conference on Recommender Systems*, Vancouver, 2018.

[12] P. M. Nadkarni, L. Ohno-Machado and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association,* vol. 18, pp. 544-551, 2011.

[13] E. Cambria and B. White, "Jumping NLP Curves: A Review of Natural Language Processing Research," *IEEE Computational Intelligence Magazine,* vol. 9, no. 2, pp. 48-57, 2014.

[14] B. C. Love, "Comparing supervised and unsupervised," *Psychonomic Bulletin & Review,* vol. 9, no. 4, pp. 829-835, 2002.

[15] J. G. Dy and C. E. Brodley, "Feature Selection for Unsupervised Learning," *Journal of Machine Learning Research,* vol. 5, pp. 845-889, 2004.

[16] S. Kavitha and K. Tarakeswar, "Search Engines:A Study," *Journal of Computer Applications (JCA),* pp. 29-33, 2011.

[17] N. Craswell, "Mean Reciprocal Rank," in *Encyclopedia of Database Systems*, Boston, Springer, 2009, p. 1703.

[18] T. Wolf, L. Debut, S. Victor and e. al, "Transformers: State-of-the-Art Natural Language Processing," *Association for Computational Linguistics,* vol. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38-45, 2020.

[19] A. Vaswani, N. Shazeer, N. Parmar and e. al., "Attention Is All You Need," in *31st Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 2017.

[20] K. Cai, "The $2 Billion Emoji: Hugging Face Wants To Be Launchpad For A Machine Learning Revolution," Forbes, 09 05 2022. [Online]. Available: https://www.forbes.com/sites/kenrickcai/2022/05/09/the-2-billion-emoji-hugging-face-wants-to-be-launchpad-for-a-machine-learning-revolution/?sh=664d2ec0f732. [Accessed 10 10 2022].

[21] M. Aliannejadi, H. Zamani, F. Crestani and B. W. Croft, "Context-aware Target Apps Selection and Recommendation," *ACM Transactions on Information Systems,* vol. 39, no. 3, pp. 29:1-29:30, 2021.

[22] H. Zeng, H. Zamani and V. Vinay, "Curriculum Learning for Dense Retrieval Distillation," in *45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 22)*, Madrid, 2022.

[23] S. Hofstätter, S.-C. Lin, J.-H. Yang and e. al., "Efficiently Teaching an Effective Dense Retriever," in *44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 21)*, Canada, 2021.

[24] H. Zhang, G. Yeyun, S. Yelong, L. Jiancheng, D. Nan and C. Weizhu, "ADVERSARIAL RETRIEVER-RANKER FOR DENSE," in *The International Conference on Learning Representations (ICLR 2022)*, Virtual, 2022.

[25] X. Lee, X. Chenyan, L. Ye, T. Kwok-Fung, L. Jialin, P. N. Bennet, A. Junaid and O. Arnold, "Approximate nearest neighbor negative contrastive learning for dense text," in *ICLR 2021*, virtual, 2021.

[26] K. Vladimir, O. Barlas, S. Min, P. S. H. Lewis, L. Wu, S. Edunov, D. Chen and W.-t. Yih, "Dense passage retrieval for open-domain question answering," in *EMNLP*, virtual, 2020.

[27] V. Sanh, D. Lysandre, J. Chaumond and T. Wolf, "DistilBERT, a distilled version of BERT: smaller,," in *The 5th EMC2 - Energy Efficient Training and Inference of Transformer Based Models*, Vancouver, 2019.

[28] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining," in *33rd Conference on Neural Information Processing Systems*, Vancouver, 2019.

[29] S. Kaitao, T. Xu, Q. Tao, L. Jianfeng and T.-Y. Liu, "MPNet: Masked and Permuted Pre-training for," in *34th Conference on Neural Information Processing Systems*, Vancouver, 2020.

[30] I. G. Nils Reimer, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, Hong Kong, 2019.

[31] G. Boesch, "Pytorch vs Tensorflow: A Head-to-Head Comparison," visio.ai, [Online]. Available: https://viso.ai/deep-learning/pytorch-vs-tensorflow/. [Accessed 18 10 2022].