BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

THESIS RECOMMENDATION SYSTEM

SYSTÉM DOPORUČOVÁNÍ DIPLOMOVÝCH PRACÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **IVAN ONUFRIIENKO**
AUTOR PRÁCE

SUPERVISOR **Ing. ANTON FIRC,**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

160862

Institut:	Department of Intelligent Systems (DITS)
Student:	**Onufriienko Ivan**
Programme:	Information Technology
Title:	**Thesis recommendation system**
Category:	Information Systems
Academic year:	2024/25

Assignment:

1. Familiarise yourself with the fundamentals of recommendation systems, focusing on different types (content-based, collaborative filtering, and hybrid methods, ...) and their common uses in various industries.
2. Learn about models for natural language processing (NLP) and describe their key features and self-deployment options.
3. Design your own Natural Language Processing-based system to help select a final (bachelor's, master's) thesis for Faculty of Information Technology students. The system can draw knowledge from the published assignments in the information system and then interactively recommend a suitable bachelor's or master's thesis according to the user's preferences.
4. Implement the proposed system and test its functionality.
5. Discuss the usability, accuracy and hardware requirements of the implemented system.

Literature:

- BOBADILLA, Jesús, et al. Recommender systems survey. *Knowledge-based systems*, 2013, 46: 109-132. https://www.sciencedirect.com/science/article/pii/S0950705113001044#ab005
- Z. Zhao et al., "Recommender Systems in the Era of Large Language Models (LLMs)," in IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 11, pp. 6889-6907, Nov. 2024, doi: 10.1109/TKDE.2024.3392335. https://ieeexplore.ieee.org/abstract/document/10506571

Requirements for the semestral defence:
Points 1 - 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Firc Anton, Ing.**
Head of Department:	Kočí Radek, Ing., Ph.D.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	25.3.2025

Abstract

This thesis is devoted to the development of recommendation systems that can suggest a final thesis using keywords given by user. It is realized by utilizing natural language processing (NLP) models: TF-IDF, BERT, SBERT, ALBERT, DistilBERT, RoBERTa, and XLNet. The content-based type of recommendation systems was used. The test was conducted with the participation of students who have already chosen their thesis and who know which words best describe their thesis. The testing showed 85% accuracy of the system and an overall positive assessment of the testers.

Abstrakt

Tato práce se věnuje vývoji doporučovacích systémů, které mohou navrhovat závěrečné práce na základě klíčových slov zadaných uživatelem. Je realizován s využitím modelů zpracování přirozeného jazyka (NLP): TF-IDF, BERT, SBERT, ALBERT, DistilBERT, RoBERTa a XLNet. Byl použit doporučovací systém typu content-based. Testování bylo provedeno za účasti studentů, kteří si již vybrali svou diplomovou práci a kteří vědí, která slova nejlépe vystihují jejich práci. Testování ukázalo 85% přesnost systému a celkově pozitivní hodnocení testujících.

Keywords

Recommendation System, Natural language processing, RS, RecSys, NLP, Content-Based Filtering, Transformers

Klíčová slova

Doporučovací systém, Zpracování přirozeného jazyka, RS, RecSys, NLP, Filtrování podle obsahu, Transformeři

Reference

ONUFRIIENKO, Ivan. *Thesis recommendation system.* Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Anton Firc,

Rozšířený abstrakt

Ve světě informací, kde se množství informací stává tak velkým, že je prostě nemožné je všechny zobrazit, je nutné si vybírat, které informace sledovat. Za tímto účelem byly vyvinuty doporučovací systémy, existují dva hlavní typy, jeden doporučuje na základě interakcí uživatele, druhý na základě popisu položek, obě metody jsou populární, ale v určitých situacích a mají své nevýhody.

Mnoho studentů má problém s výběrem závěrečné práce, protože si musí ze stovek prací vybrat tu správnou, a to není snadný úkol, s čímž jsem měl problém i já. Právě na pomoc takovým studentům je zaměřena tato práce, jejímž cílem je vytvořit doporučující systém pro závěrečné práce.

Protože tento systém pracuje s diplomovými pracemi, a tedy s textem, jsou zapotřebí speciální modely, které tento text převedou na jakési vektory pro vzájemné porovnávání. Tyto modely se nazývají modely zpracování přirozeného jazyka, pro tuto práci jsme zvolili TF-IDF, statistický model založený na jediném vzorci, BERT a jeho vylepšení či verze, sémantický model, který je schopen na základě strojového učení vytvářet informační vektory, které udávají význam zpracovávaného textu. BERT a jeho napodobeniny byly vybrány proto, že jsou nejvhodnější pro zpracování malých textových dokumentů, jako jsou například diplomové práce.

Systém byl implementován v jazyce Python, protože je to jediný jazyk, který poskytuje přístup k výše uvedeným modelům prostřednictvím různých knihoven. Samotný systém je webová stránka, na které uživatel zadá své preference a na jejich základě mu systém poskytne doporučení. To je však to, co uživatel vidí, zatímco samotný systém přebírá informační vektory ze souboru dat stažených mým skriptem z informačního systému Vysokého učení technického v Brně a vypočítává je pro každou práci a ukládá je. Při nasazení stránky však systém na vyžádání s preferencemi uživatele nejprve vypočítá informační vektory a poté je porovná s jednotlivými vektory v datasetu. A systém vrátí pět nejlepších tezí, které se uživateli zobrazí na webové stránce.

Pro ověření, zda systém skutečně funguje správně, byl proveden test se studenty, který ukázal, že v 85 % případů byli studenti schopni najít požadovanou diplomovou práci. A že 76,2 % by tento systém rádo vidělo v provozu.

Samotný systém není z hlediska systémových nároků příliš hladový, protože tyto modely jsou malé, ale každý z nich zabírá 300 megabajtů RAM, takže po spuštění a provozu zabere tento systém kolem 2 gigabajtů

Thesis recommendation system

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Anton Firc. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Ivan Onufriienko
April 25, 2025

Acknowledgements

I would like to express my gratitude for the guidance and advice provided by my supervisor Ing. Anton Firc. Many thanks to my family who supported me throughout my studies and my country that by paying enormous price protects peace in our houses.

Contents

List of Figures

Chapter 1

Introduction

We all know about services such as YouTube, Netflix, and Amazon, which we use almost every day. But have you ever wondered how these platforms manage to suggest the perfect video, movie, or product for you? The answer lies in what are known as recommendation systems. They are the tools behind the scenes that analyze your preferences and behavior to make these suggestions. Although the short answer is „recommendation systems," the long answer is much more fascinating and complex, which will be explored in this thesis. These systems analyze user behavior, preferences, and interaction history to predict and suggest items with which the user is most likely to engage [5, 2]. To enable such analysis, especially when dealing with textual data, natural language processing (NLP) models are frequently employed. These models convert raw text into numerical representations—such as vectors or embeddings—that can be processed using statistical or machine learning methods [17, 22, 16].

This bachelor's thesis focuses on designing and building a system to recommend theses based on user input. Users can provide keywords, and the system will search through a list of theses to find the ones that match best. The goal is to make it easier for users to find theses relevant to their interests. A Content-Based Filtering system was implemented, which takes thesis topics from the FIT BUT information system and uses NLP models and keywords provided by the user to make recommendations. The final implementation achieves a success rate of approximately 85%, indicating that most recommendations are considered relevant.

I chose this topic because I felt that I needed some help in choosing a thesis, because I simply did not know what topic to write my thesis on. And I believe that I am not the only one who needed help, and that other students could benefit from assistance in choosing a thesis.

This thesis will explain the concept of recommendation systems, including how they function. The process of developing such systems will be covered, with a focus on modern methods used to enhance their efficiency, including various natural language processing models. In addition, these models will be evaluated both in comparison to traditional methods and among themselves. By the end of the thesis, the reader will have a deeper understanding of how these systems work and the best approaches to creating them.

The next chapter explores what recommendation systems are, their types, and how the world's leading companies use them. In the second chapter, there will be an explanation of what natural language processing is, what types of models are based on them, and what models will be used for the project. The third chapter will cover the design of the recommendation system and its components. The fourth chapter is devoted to the imple-

mentation of the system and the problems that had to be solved during its development. The last chapters will be about testing and evaluating the effectiveness and accuracy, as well as a general discussion about the system as a whole.

Chapter 2

Recommendation Systems

Recommendation systems, often known as **Recommender systems** (**RS** or **RecSys**), are a type of filtering system that sorts suggested items based on user data (age, gender, country, type of work, hobbies, etc.) and/or actions (purchasing items, liking/disliking, watching some content, subscribing to someone, etc.) and provides the best matches among the items. Items here are products (movies, goods, videos, etc.) that the product provider is trying to sell/show to the user. [2]

In everyday human interactions, recommendation systems can be likened to how we provide suggestions to others. For example, when someone asks what movie to watch, what book to read, or whether a blue or yellow shirt looks better, we instinctively recall their preferences, style, and personality traits. Based on this information, we **provide recommendations**, acting as a natural recommendation system. In fact, this is a reflection of how recommendation algorithms work, taking information about the user from the database (our memory) to provide them with personalized suggestions.

Although this analogy simplifies the concept for better understanding, the formal definition captures the computational structure of recommendation systems with more precision [2]:

> Let C be the set of all users and let S be the set of all possible items that can be recommended, such as books, movies, or restaurants. The space S of possible items can be very large, ranging in hundreds of thousands or even millions of items in some applications, such as recommending books or CDs. Similarly, the user space can also be very large—millions in some cases. Let u be a utility function that measures the usefulness of item s to user c, i.e., $u : C \times S \to R$, where R is a totally ordered set (e.g., nonnegative integers or real numbers within a certain range). Then, for each user $c \in C$, we want to choose such an item $s' \in S$ that maximizes the user's utility. More formally:

$$\forall c \in C, \quad s'_c = \arg\max_{s \in S} u(c, s). \tag{1}$$

Not only do the right recommendations improve the user experience by showing users what they want and increasing the likelihood of purchase or consumption, but they also benefit vendors by helping them retain customers and increase customer engagement. The applications of recommendation systems extend beyond e-Commerce [32] to fields such as healthcare [21], education [28], and more. These systems are an integral part of the

decision-making process, offering personalized solutions and effectively connecting users with the most relevant products or services.

2.1 Types of Recommendation Systems

In essence, different types of recommendation systems represent different methods, resources (type of data in their databases) and processing efficiency (in terms of time and computing resources). These systems differ in how they interpret data and provide recommendations, often combining multiple approaches to achieve specific goals. The main methods are Content-Based Filtering and Collaborative Filtering, followed by improvements in a specific direction and more particular methods. These improvements are designed to solve the problems associated with recommendation systems. The main one is the Cold Start Problem, which occurs when something new is added to the system, whether it is a subject, a user, or an entire community. [5]

- New items: Lack of initial ratings makes comparisons difficult.
- New users: No history of interaction, which prevents personalized recommendations.
- New communities: A completely fresh system with no prior user or item data.

Another common issue is Data Sparsity, where interactions or attributes are incomplete. In large datasets, this is inevitable; for instance, a user cannot possibly read and rate every available book. This limitation impacts recommendation accuracy and requires techniques to infer missing information.

2.1.1 Content-Based Filtering

Content-Based Filtering (CBF) [33] focuses on recommending items based on their attributes rather than user-to-user interactions. It assumes that items positively rated by the user (e.g., via numerical ratings or likes) share attributes that align with the user's preferences. The attributes of unseen or unrated items are compared with those of positively rated items, and the most similar ones are recommended. This technique is domain-dependent, requiring detailed knowledge of item attributes, which makes it particularly useful in scenarios with rich metadata, such as books, movies, or academic papers [15].

For example, as shown in Figure 2.1, consider a user who has read and liked an article. The attributes of this article, such as topic, text length, approximate reading time, and number of images, are used to create a user profile. When another article shares similar attributes, such as the same topic and comparable reading time, it is identified as a good match and recommended to the user.

Since there is no need to collect user data, Content-Based Filtering is considered more privacy-friendly and anonymous compared to Collaborative Filtering [13]. Another advantage is that if there is enough information about the items, this type of system can overcome the Cold Start problem, but this is also a disadvantage of this type of system because the system needs a sufficiently deep knowledge of the item, which may not be possible in some cases. [29]

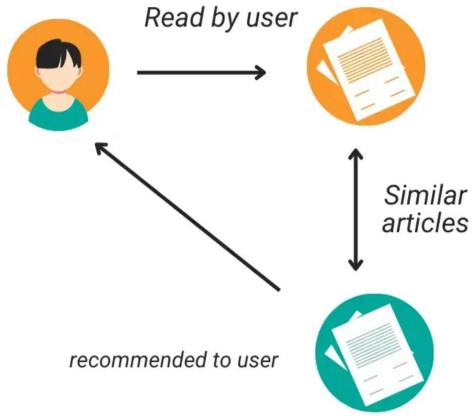

Figure 2.1: An example of how content-based filtering works. [6]

2.1.2 Collaborative Filtering

Unlike Content-Based Filtering, which focuses on item attributes, Collaborative Filtering (CF) is domain-independent and relies on user behaviors. This approach does not require detailed information about items but instead relies on user data such as age, gender, location, interests, and past interactions (e.g., purchases, ratings, or views). Recommendations are made by identifying patterns of similarity between users. The underlying principle is that if two users have had similar preferences in the past, they are likely to enjoy the same items in the future. [31]

For instance, as shown in Figure 2.2, consider two users who have both watched two movies. One of these users has also watched a third movie. Based on the assumption that users with similar tastes are likely to enjoy similar content, the second user may be recommended the third movie, as their viewing history resembles that of the first user.

Collaborative Filtering tends to suffer from the Cold Start problem and has privacy issues because it requires sharing information about users. However, unlike CBF systems, there is no need to fill in any information about the items to make recommendations. [29]

2.1.3 Knowledge-Based Systems

When addressing challenges such as the cold-start problem or new user problem, Collaborative Filtering and Content-Based Filtering often fall short. In a Knowledge-Based System, users are prompted to specify their preferences, constraints directly, or attributes relevant to their needs. For instance, in the healthcare domain, where precision is critical, a patient's attributes — such as age, weight, medical history, or allergies — are explicitly defined before making recommendations. Unlike other approaches, the system does not infer attributes dynamically, but instead uses predefined input, effectively solving the problem of a new user. However, this introduces a trade-off: entering user attributes can be time-consuming, which may affect usability in less critical domains. [34]

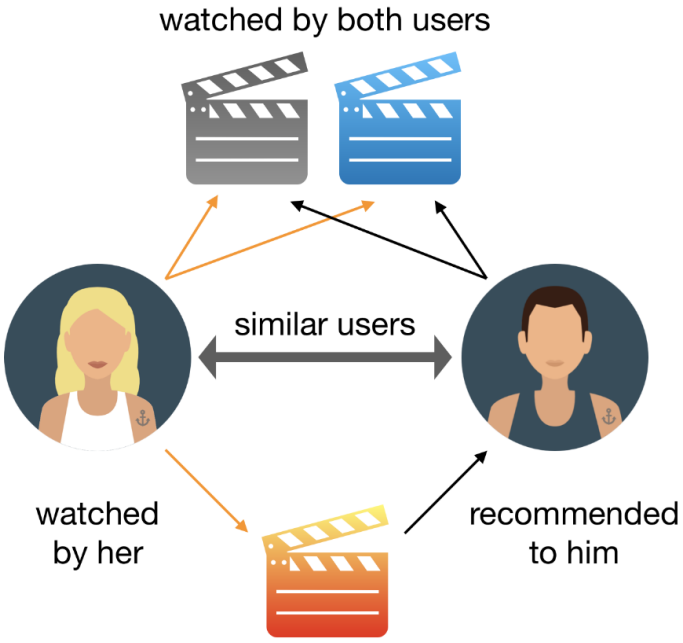

Figure 2.2: An example of how collaborative filtering works. [19]

2.1.4 Context-Aware Systems

Since Collaborative Filtering and Content-Based Filtering, in their simplicity, do not account for the context in which the user's actions take place, Context-Aware Systems have been developed to address this limitation. These systems are designed to understand and incorporate contextual factors, such as time, location, device, or the current situation of the user, that influence their behavior and preferences. If a user is looking for a book for their child and explores several products before making a purchase, the system would recognize that this search is driven by a different context than when the same user is looking for a book for work. In this case, Context-Aware Systems would aim to distinguish between these two types of search and avoid recommending children's books during the work-related search, ensuring more relevant and personalized recommendations. [1]

2.1.5 Demographic Filtering

Demographic filtering is a method used to improve the accuracy of recommendations by taking into account the demographic characteristics of users. This approach organizes users into groups based on common characteristics such as age, gender, nationality, location, and other personal attributes. By identifying users with similar demographic profiles, the system can offer recommendations that are more relevant to each group. [3, 24]

For example, people with similar cultural backgrounds or in a similar age range may have similar tastes and preferences, so the system can suggest products that are popular or suitable for that demographic segment.

This filtering method can be seen as a complement to collaborative filtering, which usually depends on user behavior and product interaction data. However, demographic filtering can help overcome limitations such as the cold start problem when there is not enough user interaction data by relying on demographic features to provide initial recommendations. [35]

2.1.6 Hybrid Systems

Hybrid recommendation systems combine two or more filtering techniques, such as Collaborative Filtering and Content-Based Filtering [7], or Collaborative Filtering and Demographic Filtering [35]. Other combinations are also widely used [7]. The primary motivation behind hybrid systems is to mitigate common challenges in recommendation, such as the cold start problem, by leveraging the strengths of different methods to provide more accurate recommendations, especially in situations with sparse or no prior data.

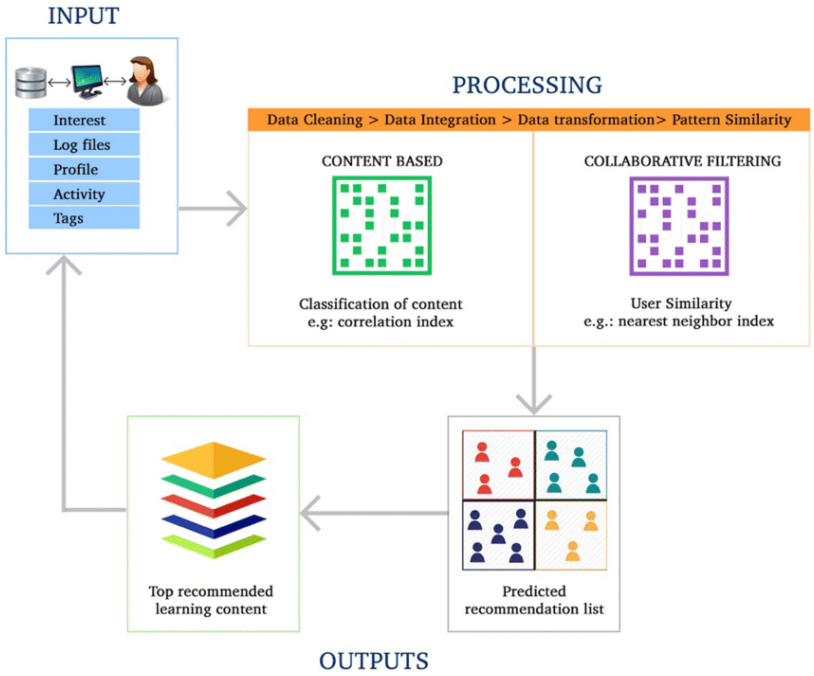

Figure 2.3: Overview of a hybrid recommendation system framework. The system processes user inputs (e.g., interests, profile, activity logs) through content-based filtering and collaborative filtering methods to generate a personalized list of top recommended items. [18]

As shown in Figure 2.3, a hybrid system typically comprises three components: data collection, processing, and system response. Data Collection involves obtaining information either through registration forms, user interactions, or both. This ensures that user preferences and actions are adequately captured. In the Processing stage, the collected data is cleaned and filtered to remove irrelevant elements, making it suitable for further analysis. The data is then processed by Collaborative Filtering and Content-Based Filtering systems. These systems apply their respective algorithms to generate personalized recommendations tailored to the user. Finally, the System Response involves comparing and combining these recommendations before presenting them to the user in a clear and actionable format. [18]

Chapter 3

Natural Language Processing Models

Before analyzing individual models, it is important to understand what Natural Language Processing (NLP) is and how it works. NLP is a scientific field that focuses on the use of computational methods to analyze and represent human language [22].

Humans process natural language every day, whether speaking, reading, or writing, by assigning meaning to words based on context and prior experience. For example, when reading a book, a person understands its content by interpreting words, giving them specific meanings influenced by their life experiences.

NLP models try to mimic humans, replicate their ability to assign meanings to individual words, and adapt to the context in which these words are used. [16]

The following sections are devoted to different types of transformers and the TF-IDF statistical model as a comparison. Transformers were chosen because they are the most effective and popular solution for natural language processing [17].

3.1 TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that reflects the importance of a word in a document in the collection of documents [9].

$$\text{TF-IDF}\,(t,d) = \text{TF}\,(t,d) \cdot \text{IDF}\,(t) \tag{2.1}$$

Formula (2.1) [4] consists of two parts: Term Frequency (2.2), which means how often a particular word appears in the document

$$\text{TF}\,(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.2}$$

- $f_{t,d}$: Frequency of term t in document d.
- $\sum_{t' \in d} f_{t',d}$: Total number of terms in document d.

and Inverse Document Frequency (2.3), which means how infrequently a particular word appears among all documents, which makes it clear whether this word is important or not.

$$\text{IDF}\,(t) = \log\left(\frac{N}{1+n_t}\right) \tag{2.3}$$

- N: Total number of documents in the corpus.
- n_t: Number of documents containing the term t.
- The 1 in $1 + n_t$ avoids division by zero.

So, the output is a table with words and their weights, where, for example, the words 'and' and 'the' have the lowest weight because they occur most often in the text.

The main advantage of TF-IDF compared to other models lies in its simplicity and computational efficiency, as it requires minimal resources and is quick to compute. The self-deployment of TF-IDF models is relatively simple and can be achieved through libraries like `Scikit-learn`[1] and its `TfidfVectorizer`[2] class.

3.2 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based NLP model introduced by Google in 2018 [11].

Its discovery has revolutionized the NLP field as it allows performing NLP tasks such as answering questions, classifying sentences, and others. The main advantage is that BERT is context aware, i.e., unlike TF-IDF, it can distinguish the meaning of the word "Python" in the sentences "He got bit by Python" and "His favorite language is Python", when context-free methods such as word2vec and TF-IDF would give the same meaning to the word. [26]

„Bidirectional" means that, unlike models such as GPT, which process text in a single direction (either left-to-right or right-to-left), BERT processes text from both directions simultaneously. This allows it to understand the full context of a word by considering the words that come both before and after it in a sentence.

The term „encoder" indicates that BERT is primarily designed for natural language understanding (NLU) tasks, where its goal is to convert text into meaningful vector representations. Although BERT can also be adapted for text generation tasks, it is less effective in this area compared to models like GPT, which are specifically optimized for natural language generation (NLG).

BERT is available through the Hugging Face's `Transformers`[3] library, `TensorFlow Hub`[4], or through the ONNX platform, which can be used with other libraries.

3.3 GPT

This section is mainly based on this article [37].

Generative Pre-trained Transformer is a family of large language models that can generate large amounts of contextually appropriate text from a relatively small text input. Unlike older models, which were typically pre-trained on labeled datasets specific to a single domain, which prevented them from performing tasks outside of that domain, GPT was pre-trained on huge amount of unlabeled data from diverse sources.

GPT models require very large computing power to be fully functional as they need to process large amounts of data. But in today's world, this is not as big a problem as it

[1] https://scikit-learn.org/stable/
[2] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[3] https://huggingface.co/docs/transformers/model_doc/bert
[4] https://www.tensorflow.org/hub/tutorials/bert_experts

was 20 years ago. OpenAI offers GPT services through its API, which comes with its pros and cons. The pros are that there is no need to worry about computing resources, but the cons include the cost and potential loss of privacy, as data falls into the hands of a private company. On the other hand, there are libraries such as `PyTorch`[5] or `TensorFlow`[6] that enable the creation of GPT-like models.

3.4 SBERT

Sentence-BERT (SBERT) [27] introduced in 2019 is an extension of the BERT model designed to generate meaningful embeddings at the sentence level. It was introduced because of the inefficiency of BERT in calculating similarities between pairs of sentences. SBERT modifies the original architecture by adding a pooling layer that converts token embeddings into fixed-size sentence embeddings, as shown in Figure 3.1.

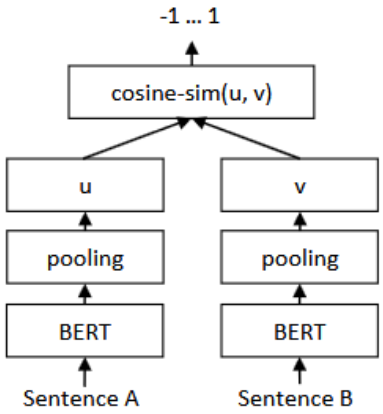

Figure 3.1: Example of SBERT architecture for computing similarity scores [27]. The „BERT“ box represents the BERT model, the „pooling“ box represents the pooling layer, „u“ and „v“ denote the generated sentence embeddings, and „cosine-sim (u,v)“ is the function used to compute the cosine similarity between the embeddings.

The ability to use SBERT is available through the Hugging Face's `Transformers`[7] or `Sentence-Transformers`[8] libraries.

3.5 ALBERT

A Lite BERT (ALBERT) [20] offers reduced memory requirements and accelerated BERT training. These improvements are made possible by the optimization techniques Factorized embedding parameterization, Cross-layer parameter sharing, and Sentence-Order Prediction (SOP).

[5] https://github.com/pytorch-labs/gpt-fast
[6] https://www.tensorflow.org/
[7] https://huggingface.co/sentence-transformers
[8] https://sbert.net/

3.5.1 Factorized Embedding Parameterization

In BERT, the size of token embeddings (E) is related to the size of the hidden layer (H). BERT uses a vocabulary (V) of 30,000 tokens, assuming $E = 768$, resulting in a matrix of size $V \times E$, i.e., 30,000 by 768, as shown in Figure 3.2. When adding a hidden unit, the size of token embeddings increases, leading to a higher memory load.

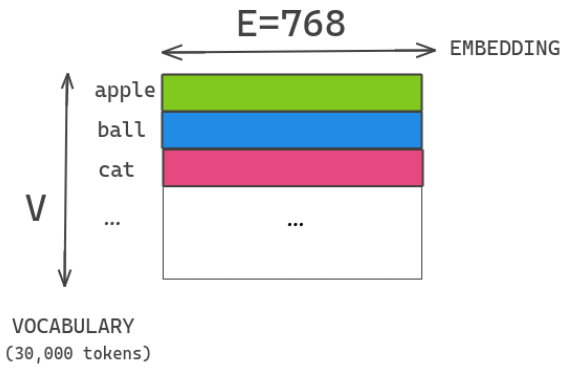

Figure 3.2: Visualization of the embedding matrix in BERT, where each token in the vocabulary ($V = 30,000$) is mapped to a dense vector of size $E = 768$, forming a matrix of dimensions 30,000 $\times$ 768. [8]

To address this issue, ALBERT factorizes the large matrix into two smaller ones: $V \times E$ and $E \times H$. Figure 3.3 illustrates this factorization process.

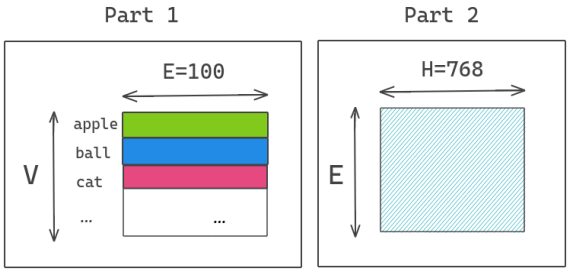

Figure 3.3: Factorization of the embedding matrix in ALBERT. Part 1: The vocabulary (V) is mapped to a smaller embedding size ($E = 100$). Part 2: A second matrix projects these smaller embeddings (E) into the hidden layer size ($H = 768$). [8]

With this factorization of adding hidden layers, the memory impact becomes much smaller.

3.5.2 Cross-layer parameter sharing

In BERT, adding more layers to the model requires adding new parameters, which leads to an exponential growth in the total number of parameters. This growth is shown in Table 3.1.

To prevent this growth, ALBERT introduces the concept of cross-layer parameter sharing. Instead of learning a new set of parameters for each layer, ALBERT allows layers to share parameters, reducing the overall parameter count while maintaining model performance. The differences between the models are shown in Figure 3.4.

Version	Hidden units	Layers	Parameters
BERT-large	1024	24	334 million
BERT-base	768	12	108 million

Table 3.1: Exponential growth in parameters as a result of increasing amount of layers. [20]

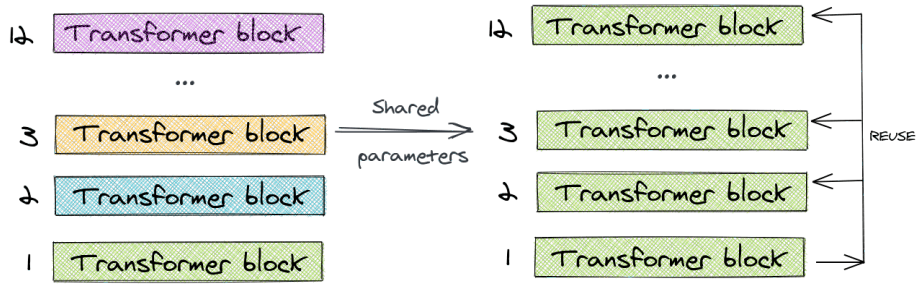

Figure 3.4: Comparison of parameter sharing in BERT and ALBERT. On the left is BERT's approach, where each Transformer block has its own unique parameters. On the right is ALBERT's approach, where all Transformer blocks share the same parameters, reducing memory usage. [8]

3.5.3 Sentence-Order Prediction

The Next Sentence Prediction process that was developed at BERT to improve performance proved to be ineffective [36, 23]. It works by taking two consecutive sentences in the same document and predicting that sequence to be true, and taking two random sentences in different documents and predicting that sequence to be false.

ALBERT offers Sentence Order Prediction, which works like this: it will take two consecutive sentences and mark them as true, and reverse them and mark them as false. The process is illustrated in Figure 3.5.

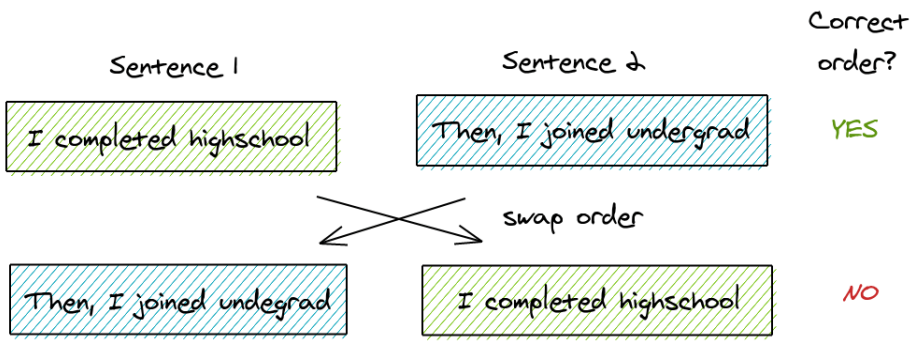

Figure 3.5: Illustration of the Sentence-Order Prediction process in ALBERT. [8]

The ALBERT model can be accessed from Hugging Face's `Transformers`, `PyTorch`, `TensorFlow` libraries or through the ONNX platform.

3.6 DistilBERT

The distilled version of BERT (DistilBERT) [30] is a smaller, faster, and more efficient version of BERT, according to the authors. The authors managed to achieve a reduction in size of 40%, and in terms of inference time, DistilBERT is more than 60% faster and smaller than BERT, while maintaining 97% of BERT's language understanding.

This achievement was made using a technology called knowledge distillation (KD). Knowledge distillation (sometimes also referred to as teacher-student learning) is a compression technique in which a small model is trained to reproduce the behavior of a larger model (or an ensemble of models) [12].

In this process, there is always a so-called teacher, which is a large and computationally hungry model, and a so-called student, which is a lightweight, small, resource-efficient model to which the teacher transfers his knowledge. In addition, there is a division into how models learn, how knowledge is transferred from teacher to student, and what knowledge the teacher represents. This is shown in detail in Figure 3.6.

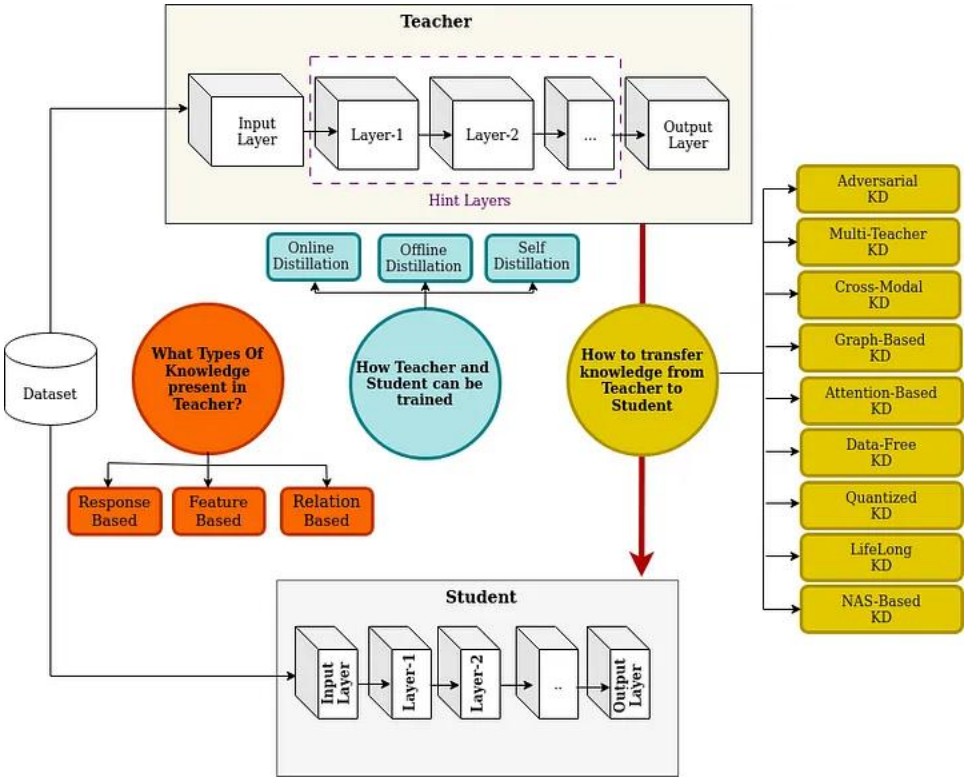

Figure 3.6: Knowledge Distillation Overview. The diagram highlights the types of knowledge in the teacher (response-based, feature-based, relation-based), training methods (online, offline, self-distillation), and various techniques for transferring knowledge (e.g., adversarial, multi-teacher, quantized distillation). [25]

As well as BERT, DistilBERT is available through Hugging Face's `Transformers`[9] library and has fine-tuning and training capabilities through `TensorFlow`[10] or `PyTorch`[11].

3.7 RoBERTa

Robustly optimized BERT approach [23] by increasing the dataset, batch size and training time, removing Next Sentence Prediction, using dynamic masking instead of static masking managed to outperform BERT in various benchmarks.

The increase in the amount of data was due to the addition of three more datasets to the BERT dataset, which the author writes about in more detail, increasing the dataset 10 times, from 16GB to 160GB. Increasing the size of the batch by reducing the training time proved to be more effective, so the authors used this and also extended the training time, so they were left with a batch size of 8 thousand sequences and 500 thousand steps. Those were fairly straightforward things to understand, and Next Sentence Prediction was already covered in the ALBERT section, so all that was left was masking.

Masking is the process of replacing a word in a sentence with a special token [MASK]. BERT uses static masking, i.e., masking occurs before training and the model sees the same mask every time it passes through a dataset (epoch), which can lead to the model memorizing masked tokens instead of understanding patterns. RoBERTa, on the other hand, offers dynamic masking, the idea being that masking occurs between epochs on random words, allowing the model to understand the context of sentences better. As the authors note, this approach is more efficient, and the performance remains at the same level or is better when using dynamic masking.

The official instructions from Hugging Face's `Transformers`[12] for RoBERTa can be followed for custom deployment. It is also possible to access through the already known `ONNX`[13] platform and there is a special `fairseq`[14] library from FacebookAI through which RoBERTa can also be accessed.

3.8 XLNet

XLNet or eXtreme Language Net [36] is an extension of the Transformer-XL [10] model that combines the main characteristics of BERT and Transformer-XL. From Transformer-XL, it takes autoregressivity, which allows it to generate tokens based on past context.

From BERT, it takes bidirectionality but does it in its own way, namely by shuffling the order of tokens in a sentence, thereby acquiring a new context, with the help of which it is possible to find out the context of all shuffles, which expands the knowledge of the model, thereby guaranteeing the bidirectionality of the model. And since XLNet is autoregressive, it does not rely on training on corrupted data, like BERT with its static masking, which uses a special token [MASK] during pre-training, resulting in pretrain-finetune discrepancy. This is a dissonance between pretraining and fine-tuning, because BERT no longer sees

[9] https://huggingface.co/distilbert
[10] https://www.tensorflow.org/
[11] https://www.kaggle.com/code/samson22/distilbert-in-pytorch
[12] https://huggingface.co/docs/transformers/model_doc/roberta
[13] https://github.com/SeldonIO/seldon-models/blob/master/pytorch/moviesentiment_roberta/pytorch-roberta-onnx.ipynb
[14] https://github.com/facebookresearch/fairseq

the special tokens during fine-tuning. The lack of masking makes XLNet more stable and reliable than BERT.

There is a custom library from the authors of `XLNet`[15] through which the model can be accessed, and Hugging Face's `Transformers`[16] library also provides access.

[15] https://github.com/zihangdai/xlnet
[16] https://huggingface.co/docs/transformers/model_doc/xlnet

Chapter 4

Design of the project

This chapter shows the process of designing the thesis recommendation system. It describes how a recommendation system should look like from the user's point of view and what processes take place before, after, and during the user's interaction with the system.

4.1 The Type of Recommendation System

Firstly, it is important to determine what type of recommendation system will be implemented. Given that the main types were discussed in Chapter 2, the ideal solution for this project is a hybrid system based on Content-Based and Collaborative filtering. With Collaborative filtering, a database of students from previous years could be created, containing their grades in subjects, the selected thesis, and the grades they received for it. And when the user wanted to choose a thesis, he would enter his grades for the subjects, and the system would compare him and other students and offer a thesis. But grades for subjects are considered private information and after discussing with the supervisor, this type of filtering would be impossible to create for recommending theses for Faculty of Information Technology students. So I decided to stick with Content-Based Filtering. According to this approach, theses will be analyzed and compared with keywords provided by the user.

4.2 User Interface Design

The system's user interface is a critical aspect, as it directly impacts usability. One of the main components is a keyword input field, where users can enter search terms to find the thesis they want. To guide users, the system may provide hints about suitable inputs, such as topics, programming languages, frameworks, or geographical areas, and warn against invalid inputs, such as random strings of letters or numbers; however, the system does not have to restrict the user in their prompts.

The second key component is the recommendation display. The system will present a list of suggested theses, each with an interactive element that allows users to click and view detailed information about the scope of a specific thesis. Additionally, users can filter recommendations by thesis type (bachelor's, master's, doctoral), ensuring results align with their academic needs.

Another crucial factor is the overall design of the user interface. The elements of the interface will be placed intuitively to help users understand the relationships between the

components and the consequences of their actions. For example, filters should be easily accessible and clearly linked to the recommendations displayed.

In addition, the user interface can provide customization options, such as changing the theme from light to dark or changing the language, so that the user can customize the look of the system to suit their preferences.

Figure 4.1 presents the user interface wireframe, highlighting the structural layout and key components of the application.

Figure 4.1: Wireframe of the user interface, illustrating the input field for text, model selection options, filtering by type of work, and a button to generate results. The output is displayed in a dedicated *Recommendations* section.

4.2.1 Expected Workflow from the User's Perspective

From the user's perspective, the recommendation system is designed to be simple and intuitive. The entire process consists of a few steps:

1. **Accessing the System**
   The user opens the web-based application in a browser. The homepage presents a clean interface with input fields.

2. **Entering Keywords**
   The user types in keywords related to their interests. These can include topics (e.g., *machine learning*), technologies (e.g., *Python*, *React*), or broader areas (e.g., *data science*, *mobile apps*).

3. **Selecting Preferences**
   The user may optionally filter the results by:

   - **Type of thesis** – bachelor's, master's, or doctoral

- **Preferred language** – Czech or English (if multilingual support is enabled)
- **NLP model** – optional, for testing or comparison purposes

4. **Submitting the Request**
   After filling in the desired inputs, the user clicks the *Get Recommendations* button.

5. **Viewing the Recommendations**
   The system processes the input and returns a list of recommended theses. Each recommendation includes:

   - Thesis title
   - Supervisor's name

6. **Exploring the Results**
   The user can click on any thesis to view more detailed information. This helps them better understand what the thesis is about and decide whether it matches their interests.

7. **Revising and Repeating (if needed)**
   If the results are not satisfactory, the user can refine the keywords or filters and request recommendations again. The system responds in real time, allowing for quick iterations.

4.3 The Back-end

This section describes the process of generating specific thesis recommendations from raw data inputs. The complete workflow is illustrated in Figure 4.2. It consists of two main phases: pre-processing theses exported from the informational system (green) and real-time keyword matching based on user input and already computed embeddings (purple).

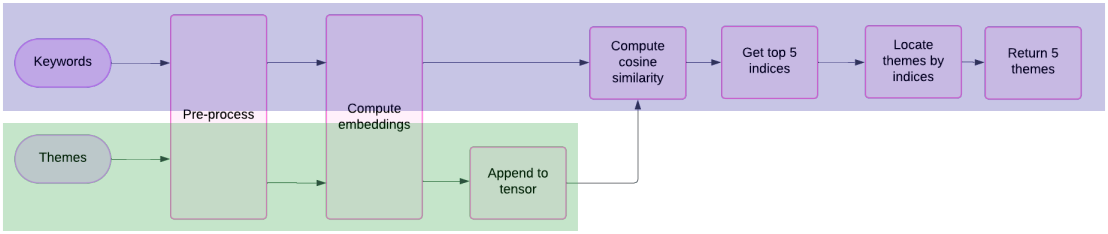

Figure 4.2: Flow diagram of getting recommendations from raw text and keywords. Green: Pre-processing themes. Purple: Real-time keyword matching.

4.3.1 Data Collection

To begin working with the raw data, it is first necessary to extract relevant information from the FIT BUT information system. Each thesis record contains multiple fields, of which the most important are shown in Table 4.1.

Field	Description
Topic	Title of the thesis (available in Czech and English)
Goals	Objectives and plan to be achieved in the thesis (CZ/EN)
Type	Bachelor's, Master's, or Doctoral thesis
Supervisor	Name of the thesis supervisor

Table 4.1: Essential data fields extracted for each thesis

Additionally, less critical fields are also available and optionally included:

- **Year** – The year the thesis was offered or written (not used for filtering)
- **Department** – Department that offers the thesis
- **Company** – Partner company under which the thesis is submitted (if applicable)
- **Focus** – The specific technical or thematic focus of the thesis

4.3.2 Text Pre-processing

Before the raw data are passed to NLP models, it must first be cleaned and normalized. This step is crucial to remove unnecessary elements and to ensure consistency across all texts. The process generally includes the following.

- Removing HTML tags and encoded characters
- Lowercasing all text
- Removing punctuation and stopwords
- Tokenization
- Lemmatization
- ...

Some models require specific types of preprocessing. For example, TF-IDF does not rely on sentence structure or context, so it allows for more aggressive cleaning, such as complete removal of punctuation and even word stemming or lemmatization. On the other hand, transformer-based models such as BERT prefer text to remain close to natural form, requiring only minimal pre-processing.

Example – Raw vs Cleaned Text

- **Original raw input (snippet from thesis goal):**

```
<ol>\r\n<li> Based upon the supervisor's instructions, study
selected topics concerning compiler writing and
WebAssembly.<li>\r\n<li> Based...
```

- **After cleaning and lemmatization (for TF-IDF):**

```
base upon supervisor instruction study select topic concern
compiler writing webassembly base
```

4.3.3 Computation of Embeddings

This process should take place in two parts: the first is the preliminary processing of all topics by the NLP model, which can take a lot of time and computing resources, and the second is the processing of user keywords, which in turn should not take much time since the model needs to process a maximum of several dozen tokens. As a result, high-dimensional vectors, or, in other words, embeddings, are obtained.

4.3.4 Comparison of Inputs

In order to find the most suitable thesis for these keywords, compare the embedding of the keywords with the embedding of each of the theses. The comparison is made using different Text Similarity approaches [14], but in the case of embeddings, Cosine similarity is appropriate since it calculates the angle between vectors, and the smaller this angle is, the more similar the texts are.

4.3.5 Filtering and Outputting

Since there is a choice between the types of theses, it is necessary to somehow filter by these types (bachelor's, master's, doctoral).

So after all these simple manipulations, a filtered and sorted list of theses is obtained, all that remains is to take a certain number from the beginning of this list. The number 5 was chosen as a balance—it provides more variety than 3 while avoiding excessive output compared to 10.

Chapter 5

Implementation

This chapter describes the development process of the thesis recommendation system, including the challenges encountered and the solutions implemented

5.1 Development Tools and Technologies

I chose Python as a programming language because it is ideal for working with text and machine learning, as it has a large number of libraries for this purpose.

Initially, I utilized my personal computer for testing various models and libraries. However, as I continued to work on my experiments, I encountered storage limitations due to the growing volume of downloaded files. To avoid the inconvenience of manually managing unnecessary files, I decided to use Google Colaboratory, a platform specifically designed for machine learning, which offers pre-installed libraries such as Numpy, TensorFlow, PyTorch, and others, to avoid the hassle of managing unnecessary files. But also installing missing libraries is not a problem, because it is all done in one separate `%pip install` command.

The main libraries required for the project are:

- **Pandas**[1] – Data manipulation and analysis; Required for interaction with JSON files.
- **PyTorch**[2] – Machine vision and inference, Natural language processing; Needed to manipulate word embeddings.
- **NumPy**[3] – Support for large multidimensional arrays and matrices, along with a large library of high-level mathematical functions for operations with these arrays; Required to sort the list of word embeddings.
- **Flask**[4] – A lightweight framework for web applications; Needed to establish communication between the front end and back end.
- **Scikit-learn**[5] – Machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction; Used to implement the TF-IDF model and the cosine similarity method.

[1] https://pandas.pydata.org/
[2] https://pytorch.org/
[3] https://numpy.org/
[4] https://flask.palletsprojects.com/en/stable/
[5] https://scikit-learn.org/stable/

- **Transformers**[6] – A powerful machine learning tool that enables natural language processing, computer vision, speech recognition, and many other tasks; It is required to load most of the models and work with them.
- **Sentence Transformers** [7] – Also known as SBERT, it is a library that contains various models with a collection of approximately 5000 models; It is used to load SBERT.

Other libraries used in the project will be described below, as they are optional and some of them can be replaced by other libraries or tools.

5.2 Data Gathering

Since this system is intended specifically for students of FIT BUT, it is essential that the thesis topics are sourced directly from the faculty's information system. Initially, the plan was to utilize an existing API to retrieve the relevant data. However, after consulting with my supervisor and exploring the available options, it became clear that although an API technically exists, it is not suitable for this use case in its current form.

Although the API allows access to a basic list of available theses, it lacks an endpoint that would provide the full details of each topic of the thesis. Additionally, accessing this data through the API requires specific permissions that are typically not granted to students. Therefore, implementing full integration with the information system would require not only the deployment of this application, but also administrative changes within the IS infrastructure, including modifications to the API and access rights.

Instead, my supervisor gave me access to the teacher's page, listing all current topics. As a workaround, I developed a simple `Selenium`[8] script that more or less does the job.

The script prompts the user for their log-in credentials (they must have access to the teacher's page in the BUT information system). Upon entering the credentials, the web driver is launched, navigating to the thesis topics page. Since authentication is required, the website redirects to the login page, where the script enters the credentials and submits the form.

Once authenticated, the page displaying the list of thesis topics is loaded. By default, only 25 topics are shown per page. Initially, I considered setting the maximum display limit (2500), but realized that the website dynamically loads content as the user scrolls. Attempting to force the maximum number caused pages to be unloaded once new content appeared, which is likely an optimization feature. Then, I attempted automatic scrolling, but it was too fast for the script to load content properly. I decided to keep the default setting of 25 topics per page, as it ensured that previously loaded topics remained visible.

Once the topics are fully loaded, the script extracts all links to individual thesis descriptions. For each link, it:

1. Opens the corresponding page in a new window.
2. Waits for it to load.
3. Extracts relevant details such as the thesis name, type, supervisor, and objectives.

[6] https://huggingface.co/docs/transformers/index
[7] https://sbert.net/
[8] https://www.selenium.dev/

4. Stores this information in a data list.

5. Closes the window and returns to the main list.

After processing all topics on a page, the script clicks the 'Next Page' button and repeats the process until no more pages are available or an error occurs. When the script completes execution, the collected data in data are saved to `themes.json` (see Listing 5.1).

```
{
    "Name_CZ": "Thesis theme in Czech.",
    "Name_EN": "Thesis theme in English.",
    "Type of Work": "bachelor",
    "Supervisor": "John Doe",
    "Targets_CZ": "Targets of thesis in Czech.",
    "Targets_EN": "Targets of thesis in English."
}
```

Listing 5.1: Example JSON Structure for each thesis in the 'themes.json' file.

Although this approach should theoretically work, in practice, the script ran into issues. After about five minutes, the website stopped responding and the script encountered an error when attempting to retrieve data. I suspect that the server began blocking my requests due to their frequency. I tried increasing the interval between requests, which may have helped slightly, but the issue persisted, leading to eventual blocking.

Before the server started rejecting requests, the script successfully downloaded around 20% of the thesis topics. To work around this, I implemented a feature that allows users to specify a starting page for data extraction. Using this method, I was able to resume the process manually and likely retrieve all available topics as of September 2024. The entire process took about a week, though further tweaking could likely optimize it. Of course, the ideal solution would be an official API.

5.3 Data Preparation

In order to start working with the data, it must first be converted into a valid form for each individual model. But before that, the data produced by the previous script needs to be opened in the main program because further manipulations will be carried out in it. This is handled using the `pandas` library, which efficiently processes JSON files. The thesis names and objectives are then extracted and merged for both the Czech and English versions.

During this merging process, text cleaning is necessary due to the presence of HTML tags. These tags improve visual presentation for users, but introduce noise that could negatively impact word embedding calculations. To address this, a function was implemented using the `re` (regular expressions) library to remove HTML tags based on their predictable pattern.

Another challenge is that some of theses lack topic names or objectives, mainly in the English section, although there are cases in Czech as well. Since there is no better alternative, these gaps are filled with a space.

No further text pre-processing is required because:

- SBERT and TF-IDF models already incorporate their own text processing.
- Models from the `Transformers` library handle tokenization and other text preprocessing via the `PretrainedTokenizer` class, which specific model tokenizers inherit from.

The output of the tokenizers is illustrated in Listing 5.2, using the BERT tokenizer as an example. Although the output may vary slightly across different models, the general principle of operation remains the same. These models are capable of processing tokenized input in this format.

```
['[CLS]', 'A', 'New', 'Comp', '##iler', 'that',
'produ', '##ces', 'a', 'Web', '##A', '##sse', '##mb',
'##ly', 'Tar', '##get', 'Co', '##de' ... '.', '[SEP]']
```

Listing 5.2: Example of BERT-tokenized text. Special tokens (e.g., `[CLS]`, `[SEP]`) and subword units (indicated by `##`) are included as part of the tokenization process.

5.4 Precomputation of Embeddings

This process is called *precomputation* because it occurs before user interaction with the system. The first step is loading the models, handled in the `load_[model name]` functions. Along with model loading, required tokenizers are initialized as needed. At this stage, the processed text, initialized tokenizers, and loaded models are ready; only embedding generation remains.

This is done in the `get_[model name]_embedding` functions, as follows:

1. The input text is tokenized.
2. The tokenized output is passed to the model.
3. The *last hidden state* is extracted, a tensor containing embeddings for all tokens in the sequence.

Since different models produce embeddings of varying sizes, they must be converted into a fixed-size representation. This is achieved using `mean(dim=1)`, which averages across the dimension of the length of the sequence, collapsing token embeddings into a single sentence embedding. The batch dimension (equal to 1 since the model processes only one thesis at a time) is then removed; it must be removed to get a standard sentence vector that can be used to compare text similarity. And the `PyTorch` tensor is converted into a `NumPy` array using `.numpy()`, making it compatible with direct comparison.

This approach to processing was used in 5 five models, namely BERT, ALBERT, RoBERTa, DistilBERT, and XLNet, which all come from the `Transformers` library. The remaining models were used as follows:

- **SBERT**: Designed specifically for semantic similarity tasks, simplifying the process with a single `SentenceTransformer.encode(text)` call.
- **TF-IDF**: Uses a completely different but similarly straightforward approach as SBERT using `TfidfVectorizer.transform()`.
- **GPT**: Not used in this project due to cost considerations and its primary purpose being text generation rather than natural language understanding.

5.4.1 Batch Processing and Optimization

Initially, the embeddings were calculated on each program run when working with a small dataset (~15 topics, manually copied). But when I got a real dataset, this process took

several hours, and after a few such waits, I realized that it was taking a lot of time, and I figured out how to save time. To optimize performance, a caching mechanism was implemented:

- After computing the embeddings, they are stored on disk.
- A switch allows toggling between computing, saving embeddings, and loading pre-computed ones.
- **Saving:** Done using `torch.save()`, and **loading:** Done with `torch.load()`.

Since the next stage of the project also requires these models, they must be loaded again when loading stored embeddings, as they are not utilized during this process. At the same time, a dictionary for embeddings is created, where a separate model is assigned its own tensor with embeddings.

5.5 Real-Time Recommendation Inference

The real-time part of the project runs when a user interacts with the system. This is managed by a `Flask` server, which also pulls up the front-end, but more on that later.

There are two endpoints that are necessary for the system to work properly. The first one is loading the main page (the '/' root), and the second one (/recommendations) is processing the data provided by the user using this system after he clicks the 'Get Recommendations' button.

The first endpoint is quite simple; it just renders the 'index.html' page. The second one is more complicated; at the beginning, data are extracted from the request, namely keywords, approach (or model) and type of work.

Next, the system retrieves the relevant tensor from the dictionary. If the tensor is missing or no keywords are provided, an error message is returned to the user. If everything is fine, then the model calculates embeddings for keywords from the user; the process is the same as in the previous section.

Next, the embeddings of the theses and the user's words are compared using the `cosine_similarity()` function. This function returns a similarity score between the user's embedding and each thesis embedding, indicating how well they match. Then the best matches are determined: `argsort()` sorts the indexes in ascending order of similarity, and `[0][::-1]` inverts the order to get the most similar topics at the top of the list.

However, this is not a list of theses, but of the indexes of these theses, and to make a list, it is necessary to assign the title, purpose, and supervisor of the thesis to each index. This is done with the help of function `.iloc[]`[9].

To provide a better understanding of the overall structure, the architecture of the system is illustrated in Figure 5.1. It outlines the interaction between the user, the web interface, the main processing script, and the stored thesis embeddings.

During the trial runs, I observed that some theses appeared multiple times. Further investigation revealed that supervisors often list identical topics for different students due to broad themes or high demand. The system maintains a set (`unique_themes`) to track already included topics to ensure that no duplicate thesis topics appear in the recommendations. Additionally, it verifies that only theses that match the user-specified type of work are considered.

[9] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html

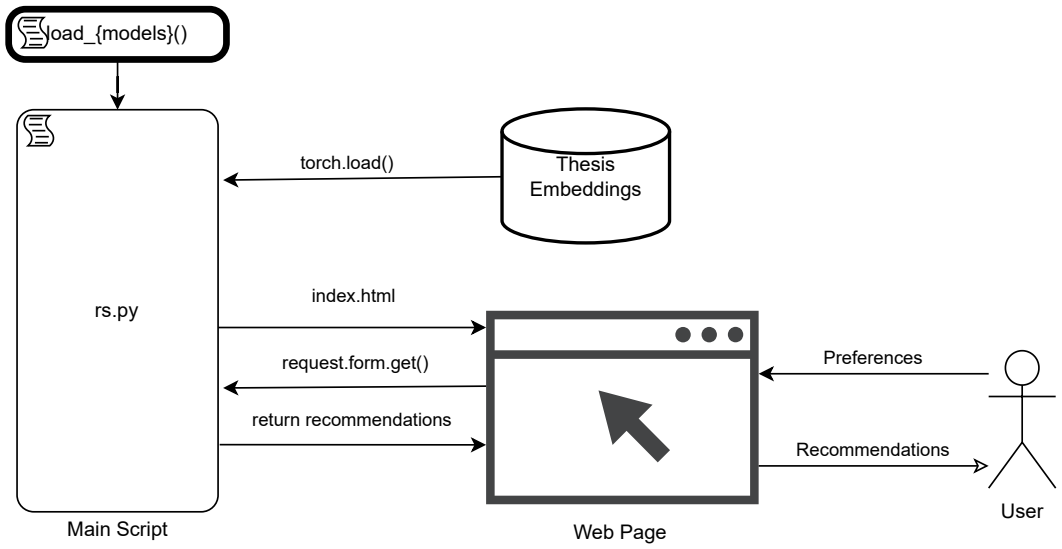

Figure 5.1: System architecture of the thesis recommendation application. The user interacts with a web page to input their preferences. The main script (‘rs.py‘) loads precomputed thesis embeddings using `torch.load()`, loads models using `load_{models}()` where `{models}` is list of 7 models that can be used to compute embeddings, processes the input, and returns the most relevant recommendations. The figure illustrates the flow of data and the interaction between the user interface, back-end script, and the database of thesis embeddings.

To start the server itself, I used the `.run()` function on port 5000, but this starts the server only on the local network, which makes it impossible for the user to access the system. So I used the `ngrok`[10] library, which provides a public URL for free. One limitation is that `ngrok` displays a warning about the free version, but this does not affect functionality and is acceptable for academic use.

5.6 Front-end

The front-end was built using standard `HTML`, `CSS`, and `JavaScript` with `jQuery`[11], without additional frameworks, as the complexity did not warrant them. However, `Bootstrap`[12] was used to decorate the page and give it a more modern look to make it more pleasant for the user to use the system.

The project follows a standard Flask structure with two key directories:

- `static/` – Stores `JavaScript`, `CSS`, and graphical assets.
- `templates/` – Contains `HTML` files.

[10] https://ngrok.com/
[11] https://jquery.com/
[12] https://getbootstrap.com/

To preserve the general appearance of all possible pages in this project, `base.html` was created, which serves as a template for all pages, ensuring a consistent layout by defining shared elements such as stylesheets, the favicon, and the general structure of the page.

Next is the `error.html` file, which is needed to handle unexpected situations, such as incorrect URLs. With the help of this file, the user will know that he/she has reached the wrong place and will be offered to return to the main page.

And finally, the main page `index.html`, which contains a form for the user to enter keywords, choose an approach from seven models, filter by type of work, and submit the form using the 'Get recommendations' button.

The logo of the University of Technology in Brno was used as a favicon.

The design was enhanced with:

- A linear gradient background for a modern aesthetic.
- Containers to highlight key sections.
- Rounded corners and shadows for a polished look.
- Interactive feedback for buttons and list items.

As a result, the user interface looks like the one shown in Figure 5.2.

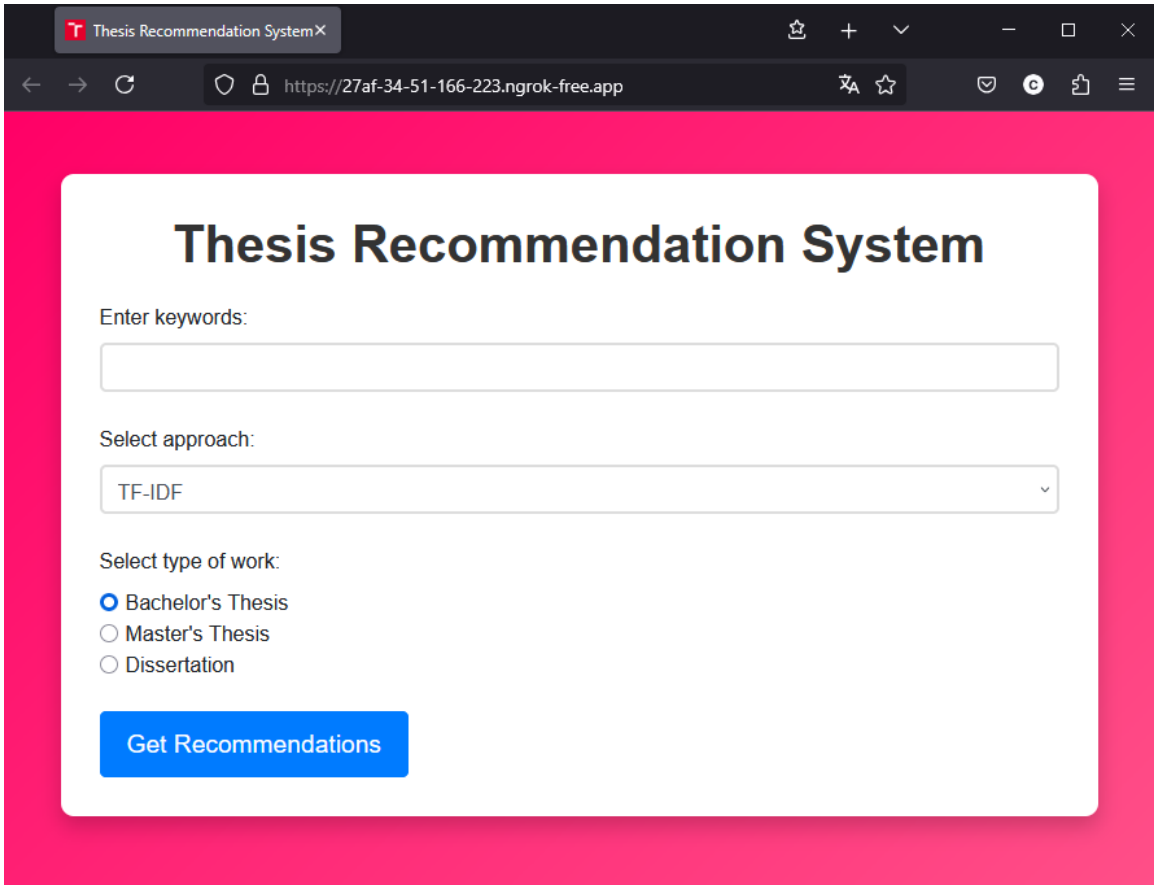

Figure 5.2: The page's appearance when the user first visits it.

`fetchRecommendations()` is the core front-end function responsible for sending requests, handling server responses, and displaying the results. It is activated immediately after

clicking on the "Get Recommendations" button and the function first disables the button, making it impossible to spam with requests. Also, the recommendations section is cleared of possible past recommendations. The form data is serialized and sent to `/recommendations` via an `AJAX` POST request. `AJAX` or Asynchronous JavaScript And XML, as the name suggests, allows asynchronous communication with the server and dynamic page updates.

As soon as the script receives a response from the server, it checks if there is an error, and if an error occurs, instead of recommending it, the script displays a red error message instead of recommendations. Otherwise, the script will add to the list of recommendations those recommendations that were provided in the response, namely the topic of the thesis, thesis supervisor, and goals.

These recommendations are then displayed on the client side in five separate rectangles with the topic name and supervisor data, and the previously blocked button to search for recommendations will be unlocked. Each recommendation can be interacted with; by clicking on one of them, it expands and the thesis objectives appear next to the theme and supervisor. This was done to reduce the text load on the page, as the thesis targets can be large, so the five recommendations will not fit on one screen.

The appearance of the recommendation elements looks like that shown in Figure 5.3.

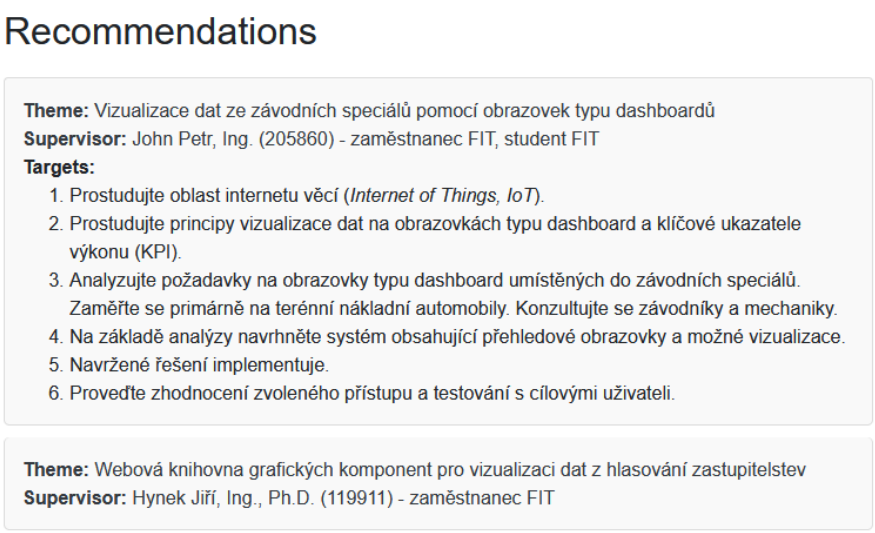

Figure 5.3: The look of two recommendations: the first is opened, the second is closed.

5.7 Extensions

Since I gained access to the dataset later than expected, I had extra time to enhance the system by implementing two key extensions: language support and pagination.

5.7.1 Language support

Initially, I planned to implement dynamic language switching only at the interface level. The supported languages are Czech, English, and Ukrainian, languages I am familiar with. The default language was English, but since Czech is the primary language of the university, its inclusion was necessary. Ukrainian was added to support my fellow citizens.

To achieve this, I added two new folders to the `static` directory:

- `langs` – Contains `.json` files for each language. Each file maps the text elements of the interface to their translations.
- `flags` – Stores flag images representing each language.

The flags were integrated into base.html, appearing as three interactive rectangles in the bottom-right corner (Figure 5.4).

Figure 5.4: Language selection flags in the user interface.

Clicking a flag updates the selectedLanguage value in local storage and instantly changes the page language by calling loadLanguage(selectedLanguage). This function retrieves the corresponding `JSON` file via an `AJAX` request `($.getJSON)` and applies translations to all relevant UI elements.

Since recommendations are dynamically loaded, their labels must also update when switching languages. The updateRecommendationsLanguage() function ensures consistency between static UI elements and dynamically generated content.

Additionally, since the BUT system maintains the Czech and English versions of each thesis, I extended the selection of languages to the model level. I implemented a language check during model loading to determine whether a Czech model is available. Some models lack Czech versions, so for those, I defaulted to English. If Czech versions become available in the future, they can be easily added.

The same embedding processing workflow, loading, computing, and saving, was applied to both Czech and English versions. The selected language is included in server requests for interactive use, allowing the system to deliver results accordingly.

Ukrainian posed a special case since the BUT system does not officially support it. To address this, I mapped Ukrainian queries to English models and embeddings, making it functionally identical to the English version.

Unfortunately, after receiving the actual dataset, I saw that about 90 percent of English theses did not have a topic or goal. As a result, this model-based language extension could not be used to its full potential in practice. So, some parts of the code related to this have been commented out to avoid wasting calculation resources once again.

5.7.2 Pagination

The second extension is the addition of the paging of theses. That is, the user can look not only at the first 5 proposed theses but also scroll further. For this purpose, forward and back buttons were added to the interface, which adds or subtracts offset from the current cursor position in the list of topics, which is written in two functions along with others in the script file for the front-end. To load a new page, these functions also call `fetchRecommendations()`. To communicate with the back-end, this offset is transmitted along with the other data in the request described above in Section 5.5.

These buttons (Figure 5.5) have been added to the `index.html` file and are visually located on the bottom left below the proposed recommendations.

On the back-end, this is implemented quite simply since, again, this is just a sorted list of recommendations, and in Python, moving through the list is easy, namely with the following expression:

Figure 5.5: Newly added buttons for pagination.

```
recommendations[offset:offset+5]
```

It slices the `recommendations` list, returning items from the index `offset` up to `offset+5`. Where `offset` is the data received from the front-end request and `recommendations` is the list of sorted recommendations.

After all these improvements, the interface that the user will be dealing with is as shown in Figure 5.6.

5.8 Hosting

To make this system accessible to users, it needs to be hosted on an external platform. Since I had some previous experience with Heroku from a school project, I initially chose it because it offers free hosting to students for two years. Although this is not a permanent solution, it provides enough time to explore alternatives. However, after trial and error, I realized that Heroku's resource limitations, specifically its 1 GB RAM cap, were insufficient. When the system attempted to load the models into memory, it exceeded 1.7 GB of usage, causing Heroku to crash due to excessive memory consumption.

As a result, I needed to find an alternative hosting solution. Initially, I considered deploying on the faculty servers, but after further research, I found that only PHP-based applications could be hosted there, indicating that it is unsuitable for this system.

Then I explored Oracle Cloud Infrastructure, which offers free hosting resources upon application approval. Despite submitting an application, I did not receive a response. While awaiting a reply, I investigated other options and discovered Google Cloud, which offers new users $300 in credits for 90 days. This trial period presented an ideal opportunity to explore the capabilities of the platform and test existing solutions.

For testing, I opted to run the system on a virtual machine. Full deployment requires additional steps, but since I was unfamiliar with the platform's interface and only needed a basic setup, a virtual machine was sufficient. The system runs entirely in the console, which is all that is required. While Google Cloud provides a built-in file transfer tool, I encountered persistent errors when using it. As a workaround, I transferred files through GitHub. Some of the files were larger than 100 megabytes, so I discovered Git Large File Storage, which helps to transfer large files. After downloading the files, all that remains is to install the libraries, but this cannot be done directly in the virtual machine, so it is necessary to activate the Python virtual environment through which the necessary libraries can be installed. But before launching the system, it is worth mentioning that all these manipulations take place in a session open on this computer, so when the computer is turned off, this session will close and, therefore, all programs in it. To prevent the session from closing, it must be separated from this computer. This is done using `tmux`[13] or other similar programs such as `screen` or `systemd service`[14]. `tmux` is a more temporary solution, and in the case of deployment on a virtual machine, `systemd service` is longer-term because

[13] https://github.com/tmux/tmux/wiki
[14] https://docs.fedoraproject.org/en-US/quick-docs/systemd-understanding-and-administering/

it is closer to the operating system and has more tools for work. So when nothing is in the way and all the problems are solved, this system can be launched.

This virtual machine costs approximately $25–30 per month, which is relatively expensive. A more affordable hosting provider should be considered for a long-term solution, as I found alternatives offering virtual machines for around $5 per month. However, for a full deployment, simply hosting on a virtual machine will not be sufficient; it would need a service that also provides a public URL.

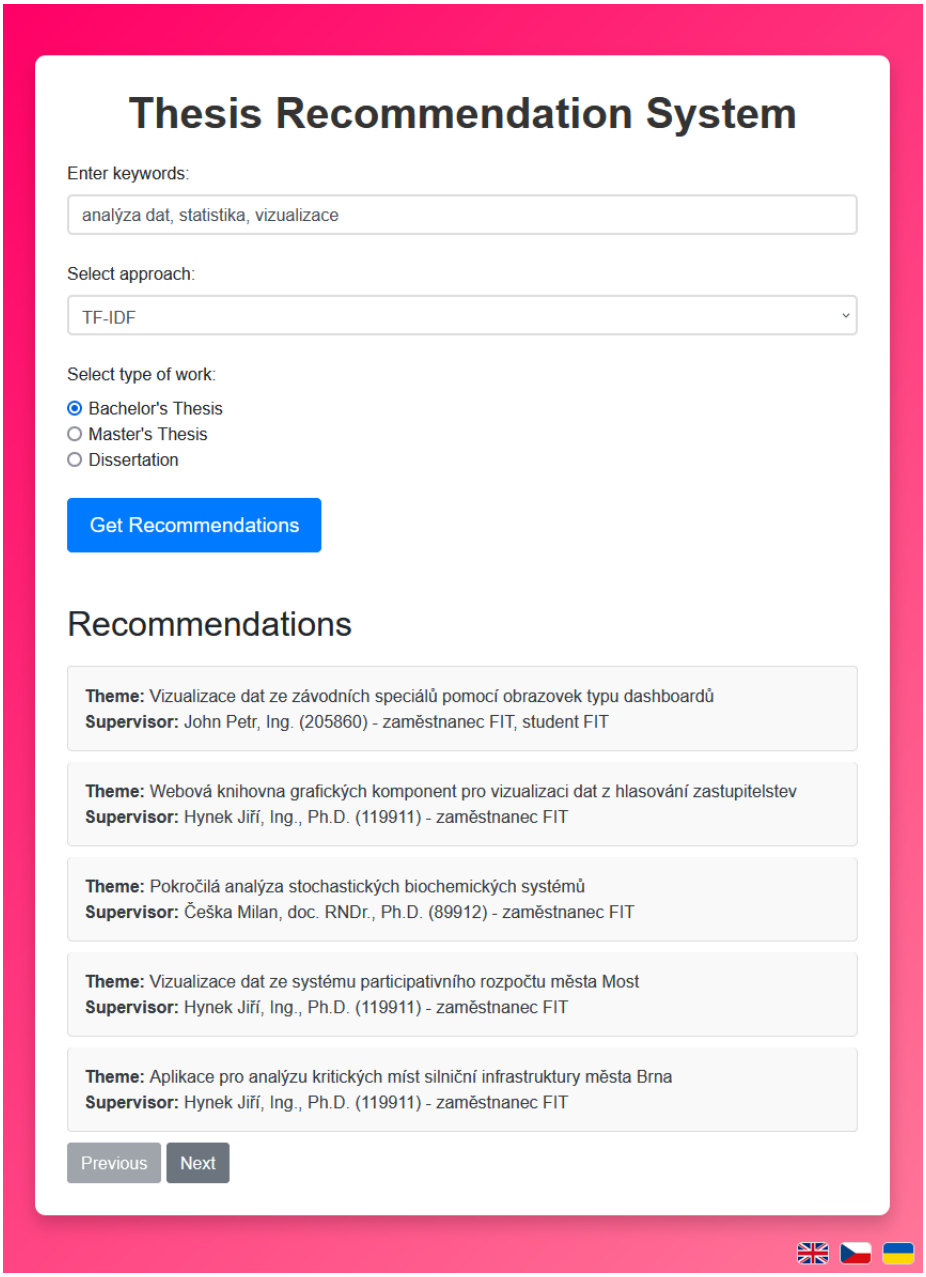

Figure 5.6: Final look of application with use-case.

Chapter 6

Testing

This chapter describes the process of testing the functionality of the recommendation system for theses, namely the choice of the testing approach, the testing protocol, and the results achieved.

6.1 Approach

At first, I thought of simply comparing the models' responses to specific keywords with each other and even wrote a sketch that did this. Still, after a while, I realized that this approach did not really test the system's functionality. So, I came up with the idea of taking a random thesis and searching for all the words in it and then removing some words from it and seeing where the thesis would rank. However, during the consultation at the defense of the semester part of the project, I was advised to test it directly on students because they will eventually use the system. It was also suggested that the system should be tested by students who have already chosen their thesis for the 24/25 academic year, because they should know what words best describe their thesis.

6.2 Protocol

A group of twenty students should be sufficient to evaluate the system; however, gathering that many specific participants presents a challenge. To efficiently distribute the necessary testing information, I used a student server on Discord, a platform widely used among FIT BUT students, including those suitable for testing.

The information provided to participants includes several key points. First, each participant must receive a brief explanation of the system, including its purpose, functionality, and their role in the evaluation. The participant's role is to describe their thesis in a sentence or a set of keywords, select an approach and filter, review the generated recommendations, and evaluate the results. Additionally, each student must be given access to the system (i.e., the URL) along with a questionnaire to complete based on the system's output. A step-by-step protocol is included within the questionnaire description to ensure consistency in the evaluation process.

Following discussions with my supervisor, the final protocol for testers is as follows:

1. Identify at least three keywords that best describe your thesis.
2. Enter these keywords in Czech in the designated field.

3. Select an approach from the available options.
4. Choose the type of thesis.
5. Click the **'Get Recommendations'** button to generate recommendations.
6. Search for your thesis in the provided recommendations and complete the questionnaire, repeating the process for each approach.

The questionnaire consists of a few but essential questions:

- **Have you found your thesis?** Yes/No
- **At what position was your thesis listed?** Options: Positions one to five, or 'Didn't found' for each approach
- **Would you use this system or recommend it to future students?** Yes/No
- Additionally, a section is provided for testers to submit comments, suggestions, or recommendations for improving the system.

6.3 Results

At the end of the testing, I received 21 responses, which presented quite good statistics.

According to the respondents, the system found the correct thesis seventeen times, as shown in Figure 6.1, but as I understood from further comments, one of the participants could not find his topic, which he failed last year but took this year. After further searching, I realized that this topic was not in the dataset of all the papers of the 24/25 academic year, because it cannot be found in the BUT system itself as another regular thesis. So, the overall accuracy of the system is 85%.

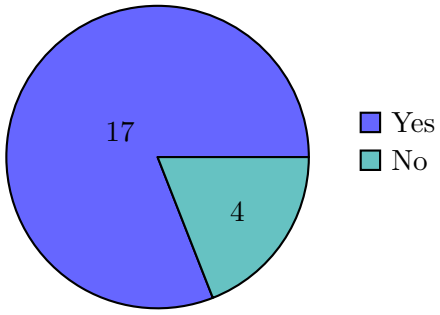

Figure 6.1: Responses to the question: 'Have you found your thesis?'

However, the statistics are different for each model (Figure 6.2): TF-IDF is the first with more than half of the results, and SBERT is the second most effective and the best of the remaining semantic models. The XLNet and ALBERT models showed the worst results, not providing any recommendations in any of the use cases.

Regarding the student's own opinion of the system, 16 respondents would like to use it or would recommend it to other students, shown in Figure 6.3. That is, in 76.2% of cases, this system would be used.

About half of the respondents wrote comments, some of which are quite informative and require further discussion, and they will be discussed in the next chapter, as well as an additional analysis of the test results.

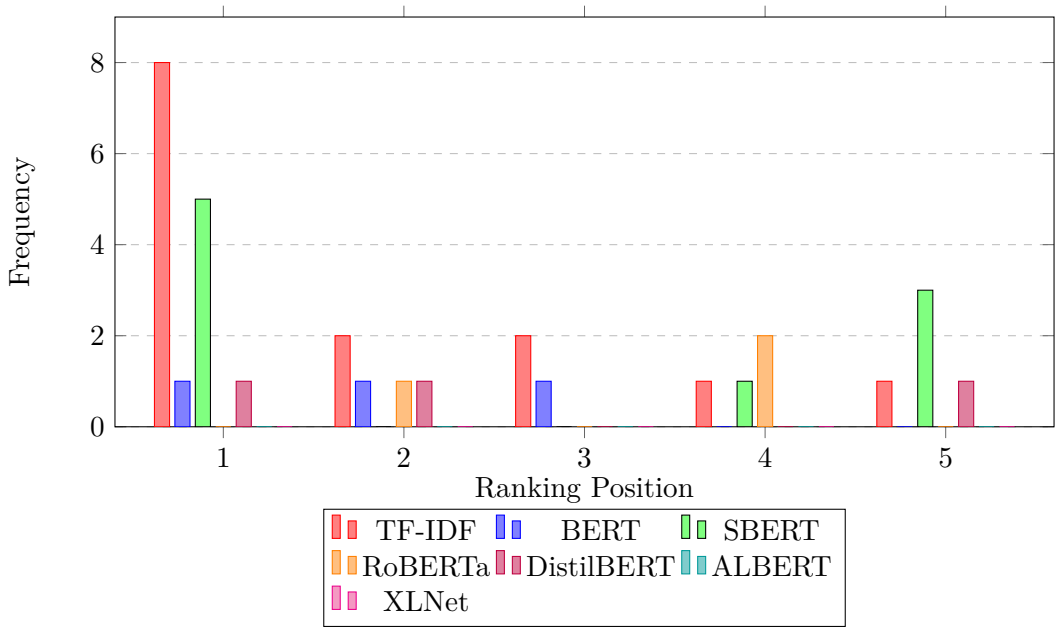

Figure 6.2: Responses to the question: 'On which place was your thesis?' Frequency means how many respondents found the thesis. Ranking Position means where the thesis was found on the first page of recommendations. Each color corresponds to a different model, as described in the legend.

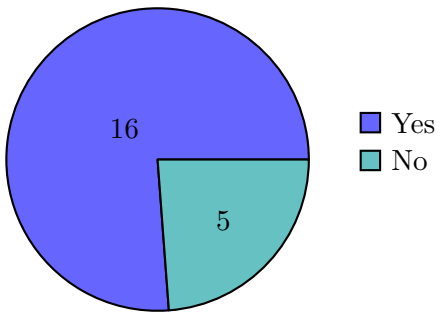

Figure 6.3: Responses to the question: 'Would you like to use this system, or would you recommend it to future students?'

Chapter 7

Discussion

This chapter describes the interpretation of the test results and the subsequent discussion of the system and various assessments of its functionality, namely its usability, accuracy, and hardware requirements.

7.1 Accuracy

Although the system achieved an overall accuracy of 85%, suggesting strong performance, a deeper analysis of the individual models reveals more nuanced insights.

TF-IDF was the most successful in retrieving these, probably because the testers used keywords directly from their topics. As a result, many theses appeared as the top recommendation. However, for context-aware models, exact keyword matching may not be sufficient. Some respondents noted that when their thesis was not found immediately, the system still suggested highly relevant alternatives or placed the correct thesis on the second or third page, which was not explicitly accounted for in the evaluation.

ALBERT and XLNet performed poorly, primarily because there are no Czech-language versions of these models. Given that the dataset contains very few English theses, their application was impractical.

Another contributing factor to inaccurate recommendations is the imperfections in the dataset. Many topics are incomplete, with missing or placeholder content, such as „TBD“ or „TODO.“ This particularly affects semantic models, which rely on contextual understanding, whereas TF-IDF remains unaffected due to its reliance on exact keyword matching.

Although TF-IDF outperformed semantic models in many cases, there were instances where TF-IDF failed, but context-aware models successfully retrieved the correct thesis.

An experienced respondent expressed surprise at the limitations of the models and suggested potential improvements. A key recommendation was to explore ModernBERT[1], a recently released model designed to address the weaknesses of earlier transformers. Furthermore, the respondent proposed a hybrid approach that combined TF-IDF with a semantic model, which could significantly improve accuracy but would require further research.

Among the tested semantic models, SBERT demonstrated the best performance. Specifically, MPNet-based SBERT[2] outperformed other variations. However, different SBERT variants may yield better results in specific applications, suggesting that further model selection and optimization could be beneficial.

[1] https://huggingface.co/blog/modernbert

[2] https://huggingface.co/docs/transformers/model_doc/mpnet

A promising avenue for further improvement is fine-tuning models using domain-specific datasets based on available theses. This would require additional research and experimentation to determine the most effective approach.

7.2 Usability

Based on the test results, 76.2% of users expressed interest in seeing this system implemented. While overall feedback was positive, users provided several suggestions and critiques regarding usability.

One issue raised was the truncation of model names in the list of approaches. Upon further investigation, it was found that this problem occurs specifically in Google Chrome, whereas it remained undetected during development in Mozilla Firefox.

Another common suggestion was to include page information, such as a visual indicator of the current page number and the total number of pages, separated by a slash (e.g., Page 1/5).

Regarding language selection, users noted that keywords should ideally match the language of the interface. However, as this is not currently feasible, a clear note should be added specifying the required language for keyword entry.

Some users also found that it was unclear what input was expected in certain fields. To address this, they suggested adding tooltips or hints next to required fields to improve clarity and usability.

7.3 Hardware requirements

In terms of hardware requirements, this system does not require much for its existence, but there are nuances. Thesis embeddings are computed on Google Colab using only the CPU (Intel(R) Xeon(R) CPU @ 2.20GHz). Given that the models are relatively small and the dataset contains approximately 1,500 theses, GPU acceleration is unnecessary.

Measurements of time and memory usage were obtained using the `time`[3] and `tracemalloc`[4] libraries. The results are presented in Table 7.1.

Model	Time Taken (s)	Peak Memory Usage (MB)
TF-IDF	14.15	6.10
BERT	937.75	59.10
SBERT	1539.51	83.29
RoBERTa	1069.70	107.68
DistilBERT	671.84	117.35
ALBERT	1578.46	117.35
XLNet	2695.54	117.35

Table 7.1: Processing Time and Peak Memory Usage for Each Model

It is also worth noting that TF-IDF embeddings take up much more disk space than the other models, namely 126.9 MB versus 4.1 MB. The system itself takes up about 7GB

[3] https://docs.python.org/3/library/time.html
[4] https://docs.python.org/3/library/tracemalloc.html

of disk space after installing the necessary libraries, but this can be optimized by using the `PyTorch` library designed for CPU use only.

The system was hosted on an e2-medium virtual machine on Google Cloud, which provides 1-2 vCPUs (1 shared core) and 4 GB of RAM. The machine runs on the AMD Rome CPU platform. After downloading the models, the RAM usage was 48% (1.92 GB). However, I observed a gradual increase in memory consumption over two months, reaching 56%, probably due to memory leaks in the Flask development server, which is not intended for production deployment. CPU usage remained around 5%, even under active use.

Each semantic model requires approximately 250 MB of RAM after downloading. By selecting only a subset of models and applying necessary optimizations, it may be possible to host this system on certain free hosting services. There may be other platforms offering 2–2.5 GB of free RAM, but I have not found any suitable options so far.

Chapter 8

Conclusion

The primary goal of this thesis was to develop a thesis recommendation system for students at the Faculty of Information Technology, Brno University of Technology. This system aims to help students select their thesis topics more efficiently.

To achieve this, it was necessary to explore Recommendation Systems and Natural Language Processing (NLP), particularly various NLP models. This theoretical foundation was applied in the design and implementation of the system.

The system features a web-based interactive interface that allows users to input text, which is analyzed using TF-IDF, BERT, SBERT, ALBERT, DistilBERT, RoBERTa, and XLNet. Although individual models produced varied results, their combined effectiveness succeeded in finding the thesis that the testers were looking for in 85% of the cases.

Testing with real users confirmed the system's usefulness, with most participants finding it helpful. A discussion was also held on the test results and the system's general characteristics, highlighting areas for improvement. Since the machine learning industry is constantly evolving and new opportunities are opening up, a new ModernBERT model was developed after the system was developed, which, according to the authors, is free from the shortcomings of previous versions and can also be used in this system. It is also worth trying and evaluating the results of fine-tuning for each model to improve the accuracy of recommendations. As for the interface, feedback revealed areas for improvement. It needs to be made more user-friendly because some users did not fully understand what they needed to do. So, the proposed enhancement is to add hints to some interface elements or explanations of how it works.

As for further system modernization, I came up with some interesting ideas. The first is integration with the university's information system, because now theses are collected using my script, which is not very stable, and this integration with the system would solve the problem of hosting this system. The second is the implementation of a kind of wizard assistant that is trained on available theses so that it can help the user to come up with a new thesis that would be on par with other theses and have similarities in the general form of existing theses.

Bibliography

[1] Adomavicius, G.; Mobasher, B.; Ricci, F. and Tuzhilin, A. Context-aware recommender systems. *AI Magazine*, 2011, vol. 32, no. 3, p. 67 – 80. Available at: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84255199841&doi=10.1609%2faimag.v32i3.2364&partnerID=40&md5=220e7f77a223bc44abdef45ca649aeec. Cited by: 351; All Open Access, Bronze Open Access.

[2] Adomavicius, G. and Tuzhilin, A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*. IEEE, 2005, vol. 17, no. 6, p. 734–749.

[3] Aïmeur, E.; Brassard, G.; Fernandez, J. M. and Onana, F. S. M. Privacy-preserving demographic filtering. In:. New York, NY, USA: Association for Computing Machinery, 2006, p. 872–878. SAC '06. ISBN 1595931082. Available at: https://doi.org/10.1145/1141277.1141479.

[4] Al Obaydy, W. I.; Hashim, H. A.; Najm, Y. and Jalal, A. A. Document classification using term frequency-inverse document frequency and K-means clustering. *Indonesian Journal of Electrical Engineering and Computer Science*, 2022, vol. 27, no. 3, p. 1517–1524.

[5] Bobadilla, J.; Ortega, F.; Hernando, A. and Gutiérrez, A. Recommender systems survey. *Knowledge-Based Systems*, 2013, vol. 46, p. 109–132. ISSN 0950-7051. Available at: https://www.sciencedirect.com/science/article/pii/S0950705113001044.

[6] BotPenguin. *What is Content-Based Filtering & its importance?* https://botpenguin.com/glossary/content-based-filtering. 2024.

[7] Burke, R. Hybrid web recommender systems. *The adaptive web: methods and strategies of web personalization*. Springer, 2007, p. 377–408.

[8] Chaudhary, A. *ALBERT: A Visual Summary*. 2020. Available at: https://amitness.com/posts/albert-visual-summary.

[9] Christian, H.; Agus, M. P. and Suhartono, D. Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF). *ComTech: Computer, Mathematics and Engineering Applications*, 2016, vol. 7, no. 4, p. 285–294.

[10] Dai, Z. Transformer-xl: Attentive language models beyond a fixed-length context. *ArXiv preprint arXiv:1901.02860*, 2019.

[11] Devlin, J.; Chang, M.-W.; Lee, K. and Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. Available at: https://arxiv.org/abs/1810.04805.

[12] Face, H. DistilBERT: Smaller, Faster, Cheaper, and Lighter BERT. *Medium*, 2020. Available at: https://medium.com/huggingface/distilbert-8cf3380435b5.

[13] Friedman, A.; Knijnenburg, B.; Vanhecke, K.; Martens, L. and Berkovsky, S. Privacy Aspects of Recommender Systems. In:. Springer, January 2015, p. 649–688. ISBN 978-1-4899-7636-9.

[14] Gomaa, W. and Fahmy, A. A Survey of Text Similarity Approaches. *International journal of Computer Applications*, april 2013, vol. 68.

[15] Isinkaye, F.; Folajimi, Y. and Ojokoh, B. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 2015, vol. 16, no. 3, p. 261–273. ISSN 1110-8665. Available at: https://www.sciencedirect.com/science/article/pii/S1110866515000341.

[16] Jurafsky, D. *Speech and language processing*. Prentice-Hall, 2000.

[17] Kalyan, K. S. A survey of GPT-3 family large language models including ChatGPT and GPT-4. *Natural Language Processing Journal*, 2024, vol. 6, p. 100048. ISSN 2949-7191. Available at: https://www.sciencedirect.com/science/article/pii/S2949719123000456.

[18] Khanal, S.; P.W.C, P.; Alsadoon, A. and Maag, A. A systematic review: machine learning based recommendation systems for e-learning. *Education and Information Technologies*, july 2020, vol. 25, p. 1–30.

[19] Kumar, A. *Recommender Systems in Machine Learning: Examples*. 2024. Available at: https://vitalflux.com/recommender-systems-in-machine-learning-examples/. Accessed: 2024-12-05.

[20] Lan, Z. Albert: A lite bert for self-supervised learning of language representations. *ArXiv preprint arXiv:1909.11942*, 2019.

[21] Lee, J. and Song, M. Recommender Systems in Healthcare: An Overview of State-of-the-Art Approaches, Challenges, and Future Directions. *Journal of Healthcare Informatics Research*. Springer Nature, 2022, vol. 6, no. 4, p. 425–447.

[22] Liddy, E. D. Natural language processing, 2001.

[23] Liu, Y. Roberta: A robustly optimized bert pretraining approach. *ArXiv preprint arXiv:1907.11692*, 2019, vol. 364.

[24] Pazzani, M. J. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*, 1999, vol. 13, no. 5, p. 393–408. Available at: https://doi.org/10.1023/A:1006544522159.

[25] Prakash, S. Understanding Knowledge Distillation in Simple Steps. *Medium*, 2023. Available at: https://medium.com/@satya15july_11937/understanding-knowledge-distillation-in-simple-steps-3310ef01f5e0. Accessed: 2024-12-21.

[26] Ravichandiran, S. *Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT*. Packt Publishing Ltd, 2021.

[27] Reimers, N. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *ArXiv preprint arXiv:1908.10084*, 2019.

[28] Riahi, F. and De Freitas, S. Recommender Systems for Education: State-of-the-Art, Challenges, and Research Opportunities. *Electronics*. MDPI, 2021, vol. 10, no. 14, p. 1611. Available at: https://www.mdpi.com/2079-9292/10/14/1611.

[29] Roy, D. and Dutta, M. A systematic review and research perspective on recommender systems. *Journal of Big Data*, 2022, vol. 9, no. 1, p. 59. ISSN 2196-1115. Available at: https://doi.org/10.1186/s40537-022-00592-5.

[30] Sanh, V. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv preprint arXiv:1910.01108*, 2019.

[31] Schafer, J. B.; Frankowski, D.; Herlocker, J. and Sen, S. Collaborative Filtering Recommender Systems. In: Brusilovsky, P.; Kobsa, A. and Nejdl, W., ed. *The Adaptive Web: Methods and Strategies of Web Personalization*. Berlin, Heidelberg: Springer, 2007, p. 291–324. Available at: https://link.springer.com/content/pdf/10.1007/978-3-540-72079-9_9.

[32] Schafer, J. B.; Konstan, J. A. and Riedl, J. Recommender Systems in E-Commerce. In: *Proceedings of the 1st ACM Conference on Electronic Commerce (EC'99)*. ACM, 1999, p. 158–166. Available at: https://www.researchgate.net/publication/2507550_Recommender_Systems_in_E-Commerce.

[33] Son, J. and Kim, S. B. Content-based filtering for recommendation systems using multiattribute networks. *Expert Systems with Applications*. Elsevier, 2017, vol. 89, p. 404–412.

[34] Tarus, J. K.; Niu, Z. and Mustafa, G. Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning. *Artificial Intelligence Review*, 2018, vol. 50, no. 1, p. 21–48. ISSN 1573-7462. Available at: https://doi.org/10.1007/s10462-017-9539-5.

[35] Vozalis, M. and Margaritis, K. G. Collaborative filtering enhanced by demographic correlation. In: *AIAI symposium on professional practice in AI, of the 18th world computer congress*. 2004.

[36] Yang, Z. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *ArXiv preprint arXiv:1906.08237*, 2019.

[37] Yenduri, G.; Ramalingam, M.; Selvi, G. C.; Supriya, Y.; Srivastava, G. et al. Gpt (generative pre-trained transformer)–a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *IEEE Access*. IEEE, 2024.