Utrecht University

Master Thesis Business Informatics

Beyond Keywords: Intent-Driven Semantic Code Search in Software Ecosystems

Author:
Chris Pfaff

1st Supervisor:
dr. Siamak Farshidi
2nd Supervisor:
dr. Slinger Jansen

SearchSeco Research Group
Department of Computer Science

June 21, 2023

"Seek, and ye shall find" - Matthew 7:7

Abstract

The exponential growth of code repositories has posed significant challenges for developers in efficiently and effectively searching for relevant code snippets. Traditional keyword-based code search engines often struggle to provide accurate results due to the ambiguity inherent in programming language keywords and the semantic gap between the developer's search intent and the code syntax.

To address this challenge, this study proposes a novel approach—an advanced semantic code search engine—that harnesses intent modelling and vector embedding techniques to enhance the relevance of search results. Our methodology utilizes machine learning models to extract the developer's search intent from their query, thereby capturing the underlying meaning of their search. Furthermore, the code snippets are represented using vector embeddings, which capture the semantic context and relationships between different pieces of code. This allows for a more nuanced understanding of the code snippets' meanings and functionalities.

The proposed system ranks the code snippets based on their semantic similarity with the user's search intent. This ranking approach facilitates the delivery of more accurate and relevant search results, providing developers with a more targeted and practical search experience. Moreover, the improved relevance of the search results guides users in the right direction for future searches, fostering an iterative and progressive learning process.

The findings of this research demonstrate the effectiveness of leveraging intent modelling and vector embedding techniques in enhancing the search capabilities of code repositories. By bridging the gap between the developer's search intent and the code syntax, the proposed semantic code search engine offers a valuable tool for developers to locate and utilize relevant code snippets effectively.

Keywords: Code search, semantic search, intent modelling, vector embeddings, search intent, machine learning, iterative learning

Acknowledgements

This past year has been a challenging journey where my supervisor and I consistently challenged me to improve. The result is, in my opinion, one of the most extended, best and most challenging projects I have worked on as a student. With my little knowledge of NLP and Machine Learning, I started this project with a growth mindset, and in the end, I am thrilled it paid off. I want to thank my supervisor Siamak for the weekly meetings, challenging me repeatedly to improve and create the best version of myself. I would also like to thank everyone who has mentally supported me over the past year, including my family, colleagues, and friends. The thesis you have in front of you now resembles over a year of discovery, attending scientific talks, programming, and evaluating. It challenged me to combine all R, Python, Data Science, and Machine Learning skills I attained through my Master's in Business Informatics. I sincerely hope you will enjoy reading it.

Contents

Chapter 1

Introduction

The primary objective of this study is to explore the application of user intent modelling to enhance the performance and effectiveness of semantic code search engines. This initial chapter will present an extensive examination of relevant background information, followed by a comprehensive drawing of the problem statement. Furthermore, the study aims to establish clearly defined study objectives and research questions, ensuring a systematic and focused approach to investigating the subject matter. Different research methods will be employed to accomplish these objectives, which will be elaborated upon in detail. Lastly, the scientific significance of this study will be illustrated, elucidating its potential contributions to the existing knowledge in the semantic code search engines field.

1.1 Background

In the current landscape of our modern world, the significance of software has witnessed a steady rise, serving as an instrumental solution to a diverse range of everyday challenges and predicaments encountered by individuals [81]. It greatly impacts various facets of our lives, finding integration in an assortment of contemporary household appliances such as washing machines, climate control systems, and security frameworks. Consequently, all software is inherently driven by programming code, which resides within designated software repositories, serving as repositories for these essential code snippets [40]. These online software repositories harbour extensive code lines, encapsulating the intricacies of countless software programs [12]. As the size of these repositories continues to expand exponentially, locating specific code segments that software developers seek becomes increasingly challenging. Consequently, developers frequently rely on search engines to find reusable code snippets or use them as reference points for enhancing their existing code [124]. These codes, created and shared by developers from diverse backgrounds, are subject to licensing regulations that dictate how much they can be reused, thereby underscoring the importance of proper code attribution. However, with recent advancements in code generation techniques [116], the attribution of code ownership has somewhat been neglected, prioritizing rapid progress within the field. Traditional code search engines address this concern by providing information regarding the source and license of the methods, thereby instilling a sense of trust in users regarding the origin of the code snippet [60].

Nonetheless, conventional search engines such as Google have failed to adequately meet the distinct requirements of developers, prompting the emergence of specialized code search engines like Koders [115]. The rationale behind this shift was driven by the realization that ordinary web search engines did not yield optimal results, necessitating search engines tailored explicitly to cater to the needs of developers. However, despite the intent to fulfil this demand, specialized code search engines have not been

widely embraced and have exhibited a propensity only to provide valuable outcomes for queries with high specificity [8]. It has been observed that many developers still prefer to rely on regular web search engines instead of utilizing the aforementioned specialized code search engines [105]. Nevertheless, the advent of advanced Natural Language Processing (NLP) and Machine Learning (ML) techniques has ushered in a new era of code search known as Semantic Code Search, which has gained considerable traction in recent times [96, 17]. Leveraging the potential of these cutting-edge technologies, specialized code search engines now have an unprecedented opportunity to enhance their functionalities and align more effectively with users' ever-evolving needs and expectations.

1.2 Problem statement

In the contemporary landscape, an array of semantic code snippet search engines have emerged, showcasing notable contenders such as CodeSearchNet, Scotch, and CodeMatcher [56, 25, 76]. Recent studies in code search primarily revolve around utilising novel machine learning models [56, 11]. State-of-the-art applications actively engage in competitions such as CodeSearchNet and benchmark their models against others, thereby evaluating their performance using benchmark sets like CodexGlue [56, 79]. Through these endeavours, researchers strive to refine existing algorithms and enhance the ranking results for end users. Nevertheless, despite these advancements, current semantic search engines have not yet achieved flawless modelling between input queries and the subsequent code ranking they provide [18, 147, 19]. Consequently, developers often find themselves repeatedly reformulating their questions [19], instead of actually finding the code that matches their intent. This leads us to the following concise problem statement:

The current problem in semantic code search is the lack of accurate intent understanding, resulting in suboptimal code snippet rankings and a time-consuming search process for developers, highlighting the need for implementing intent modelling to enhance the overall search experience and address the significant challenge of efficiently retrieving relevant code.

1.3 Objectives and research questions

To address the limitations presented in the problem statement, a recent study by Al-Hossami [52] proposed adopting a dialogue system to improve ranking outcomes. By effectively modelling user intent within the query, it is envisaged that excessive query reformulation may no longer be necessary to optimize ranking results [19]. The core objective of this study is to enhance the user experience in code search by implementing a dialogue system that effectively captures and models user search intent instead of focusing solely on refining machine learning algorithms as pursued in the existing literature. By implementing various Natural Language Processing (NLP) and Machine Learning (ML) techniques described in subsequent chapters, we aim to integrate them harmoniously within a code search engine, laying the groundwork for future research endeavours within this domain.

The primary objective of this study is to delve into the inner workings of semantic code search engines and explore avenues for enhancing the end-user experience through user intent modelling. To achieve this, an application will be designed and developed, which, based on user input, will present a ranked list of software snippets. This user input can take the form of a "conversation" with a conversational AI system,

as described in Al-Hossami et al. [52]. Subsequently, the user input will be mapped to determine the code query employed for searching the code database.

The scientific contribution of this research encompasses the outcomes derived from a systematic literature study on the fundamental components of a code search engine, in conjunction with the design and prototyping of a code search engine incorporating user intent modelling. This code search engine can be evaluated continuously and improved by applying reinforcement learning techniques, explicitly employing human feedback [92]. To simulate this process, an experiment will be detailed in Chapter 3. By integrating a dialogue system within the design framework, this study will pave the way for exploring the amalgamation of semantic code search with user intent and entity modelling.

To address the issues outlined in Section 1.2 and accomplish the study's objectives, the following research questions have been formulated: The main research question (MRQ) is 'How can code search engines understand software developers' intentions?' The MRQ is divided into four research questions to structure the thesis. These four research questions are as follows:

(RQ1) What are the essential components for a code search engine?

(RQ2) How can software developers' search intent be modelled to support them with the code search process?

(RQ3) Which components should be employed in a code search engine?

(RQ4) How can the code search engine be evaluated?

The study employs a range of methods to address each of the research questions posed. Table 1.1 provides an overview of the methods chosen to answer each research question. A brief overview of these methods will be presented in the subsequent section, with comprehensive details provided in Chapter 3.

TABLE 1.1: Research questions and their proposed methods. Systematic Literature Review (SLR), Design Science (DS), Technology Acceptance Model 2 (TAM2)

Research Questions	**SLR**	**DS**	**Experiment/TAM2**
RQ1 What are the essential components for a code search engine?	x		
RQ2 How can software developers' search intent be modelled to support them with the code search process?	x		
RQ3 Which components should be employed in a code search engine?	x	x	
RQ4 How can the code search engine be evaluated?		x	x

1.4 Research Methods

The methodology employed in this study is detailed extensively in Chapter 3. This section provides a summary of the scientific methods used.

To address the first, second, and third research questions, a Systematic Literature Review (SLR) was conducted. The SLR is comprehensively described in Chapter 2 and serves as a foundational method to investigate and synthesize existing knowledge in the field.

The Design Science approach addressed the fourth and final research question. This involved creating a software artifact that serves as a solution to the research problem. The design process, illustrated in Figure 1.1, integrates insights from the literature study, providing the scientific foundation for the software artifact's conceptual framework. The subsequent steps involve conducting an experiment and administering a TAM2 (Technology Acceptance Model 2) questionnaire to evaluate the effectiveness and usability of the proposed software artifact.

Combining these methods (SLR, Design Science, experiment, and TAM2 questionnaire) collectively addresses the research questions and establishes a framework for answering the main research question and achieving the study's objectives.

1.4.1 Development Process

Additionally, guide the development process of the software artifact in design science research; a framework was adopted from the work of Farshidi [33, 9, 34] to show the iterative refinement based on literature and artifact evaluation on the software artifact. This approach aids in making informed decisions throughout the design process of the intent-enhanced code search engine. The software artifact's capabilities and areas for improvement are validated through the experiment and TAM2 questionnaire, ensuring that the proposed solution effectively fulfils its intended purpose. Chapter 2 will explain how the SLR forms the basis of the knowledge acquisition used to create design decisions that improve the software artifact. It will also act as the Description and Explanation for all of the concepts used in the software artifact and where they came from. Chapter 4 will show the results of the experiment and evaluation of the software artifact in context to fulfil the validation of the Intent-Enhanced Code Search Engine.

1.5 Significance

The significance of this research project lies in its potential to contribute to the advancement of more efficient and effective code search engines. Furthermore, by providing a comprehensive guide and prototype for creating a semantic code search engine, along with guidelines for enhancements in various areas and showcasing evaluation methods, this research project empowers other scientists in the community to further explore and expand upon the field of semantic code search.

Through the application of intent modelling techniques, this project aims to demonstrate the efficacy and feasibility of innovating with different strategies in semantic code search. Additionally, it highlights the potential for further improvement in capturing the intent of software developers, thus enhancing the precision and relevance of search results in code search engines.

The practices elucidated in this thesis also hold potential for broader applications, extending beyond code search engines. They can be leveraged in generative AI systems and the development of enhanced machine learning models, paving the way for

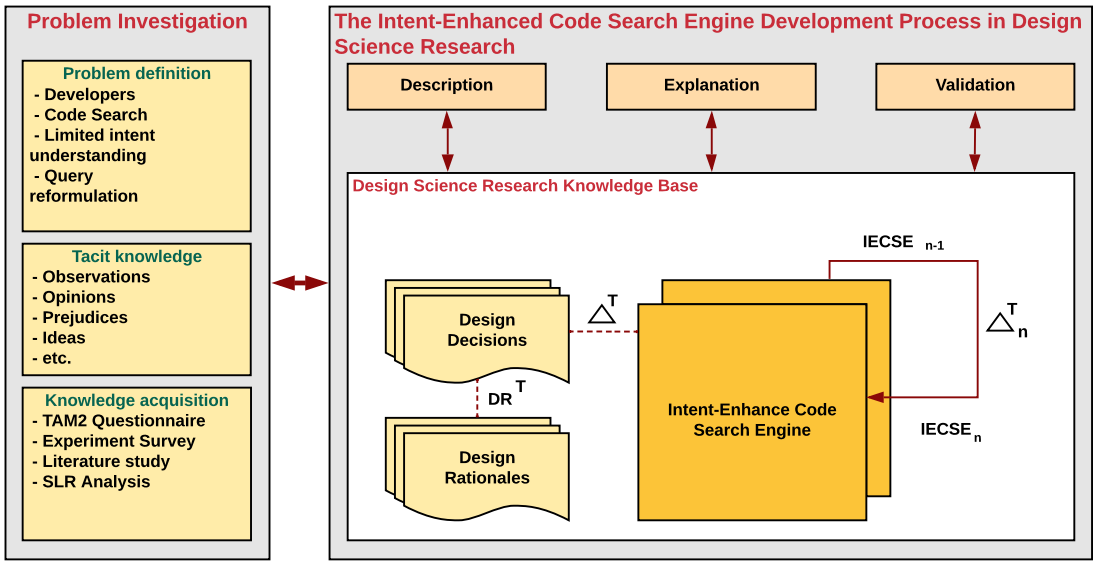


FIGURE 1.1: The design process of the intent-enhanced code search engine. Framework adopted from [33]

advancements in various domains. This research project aims to foster continuous progress and innovation in semantic code search and its related disciplines by sharing these findings.

1.6 Concept Explanations

This section defines concepts used to answer the main research question and the research questions used to structure it. It explains each concept in an easy-to-understand manner for clarity.

- **Search Engine**: The original Google paper [13] defined the anatomy of a search engine, including components like a crawler, indexer, searcher, sorter, and PageRank, enabling users to search the World Wide Web.
- **Code Search Engine**: Code search engines return relevant code snippets based on user queries [84].
- **User Intent Modelling**: User intent can vary even with similar queries. Large-scale log analysis of user behaviour patterns is one technique to model user intent [123]. Other techniques will be discussed in Chapter 3.
- **Indexing**: Indexing creates an index that points to the actual data storage where the code snippets are stored.
- **Code Input Encoder**: This method encodes a code snippet into a vector or another representation, making it easier to handle by computing engines or algorithms. Indexing code snippets in vector space provides various benefits, as explained in Chapter 3.

- **Code Input Decoder**: An encoded code snippet in vector space is not intuitively readable for humans, so it needs to be decoded back to its original form for human understanding.
- **Abstract Syntax Tree (AST)**: An AST is a tree representation of code snippets' abstract syntactical text features. It shows the abstract syntax of a code snippet while abstracting away specific implementation details [67, 89].
- **Control Flow Graph (CFG)**: A CFG is a graph-based representation of all paths that might be traversed through a program or method. It provides structural and semantic information about the code [131].
- **Call Graph**: A call graph represents code functions and their calls to one another.
- **Querying**: Querying is the process of asking a question or sending a command to the search engine to retrieve ranked results. Queries can be in the form of text input strings or even code snippets [64].
- **Conversational AI**: Conversational AI refers to artificial intelligence programmed to engage in conversations and achieve specific goals.
- **Replacement Queries**: Replacement queries are used in conversational AI and involve replacing specific parts of the textual user query to improve result matching. For example, "Singleton Java" may be reformulated as "Singleton Design Pattern Java" to improve ranking results [128, 1].
- **Convolutional Neural Network (CNN)**: A CNN is a deep learning algorithm that uses convolutional neural layers [68]. It is commonly used in image recognition tasks but can also be applied to code search [129]. Attention mechanisms are often added to CNN models to improve performance [87, 4]. In code search, it trains on natural language code snippets and input queries.
- **Recurrent Neural Network (RNN)**: RNNs differ from traditional feed-forward neural networks by having feedback loops [46]. This allows the modelling of interactions between words in input queries or code snippets using techniques like Bi-Directional Long Short-term Memory networks [51, 144].
- **Graph Neural Network (GNN)**: GNNs are neural networks designed for graph data structures like ASTs, CFGs, and call graphs [135, 157]. They capture dependencies and relationships between code elements. Different categories of GNNs include convolutional GNNs, recurrent GNNs, graph autoencoders, and spatial-temporal GNNs [135, 157].
- **Feed-Forward Neural Network (FFNN)**: FFNNs are simpler forms of neural networks compared to RNNs [86]. They are based on the structure of a real brain using neurons. FFNNs are supervised learning techniques that train the network based on desired outputs from user queries.
- **Reinforcement Learning**: RL is a machine learning paradigm where an agent learns behaviour through trial-and-error interactions with an environment [62]. In a code search engine context, RL maps to a developer interacting with the engine to retrieve a code snippet matching their intentions.
- **Ranking**: Ranking involves structuring documents that match the user query to present the most relevant ones [13].

- **Multi-Feature Ranking**: Multi-Feature Ranking utilizes multiple searching, indexing, and querying features to rank results. Techniques like CRaDLe, PSCS, CommitBert, and CodeMatcher incorporate syntax and semantic information to improve ranking [43, 117, 61, 76].

- **Best Match 25 (BM25)**: BM25 is an algorithm that calculates the relevance of documents to a given search query [107]. Equation 1.1 shows the BM25 scoring function.

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \tag{1.1}$$

- **Cosine Similarity**: Cosine similarity measures the similarity or difference between two vector embeddings, such as query and code snippet embeddings [17, 90]. Equation 1.2 shows the cosine similarity calculation between two vectors.

$$\text{cosine\_similarity}(A,B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1.2}$$

These definitions and concepts will be used throughout the study to analyze and develop techniques for improving code search engines. In addition, they provide a foundation for understanding the components, algorithms, and methodologies involved in code search and retrieval.

1.7 Conceptual Framework

Based on earlier studies and a publication [95], a conceptual framework for a code search engine was created. Figure 1.2 shows the proposed conceptual framework which formed the basis for the rest of the study.

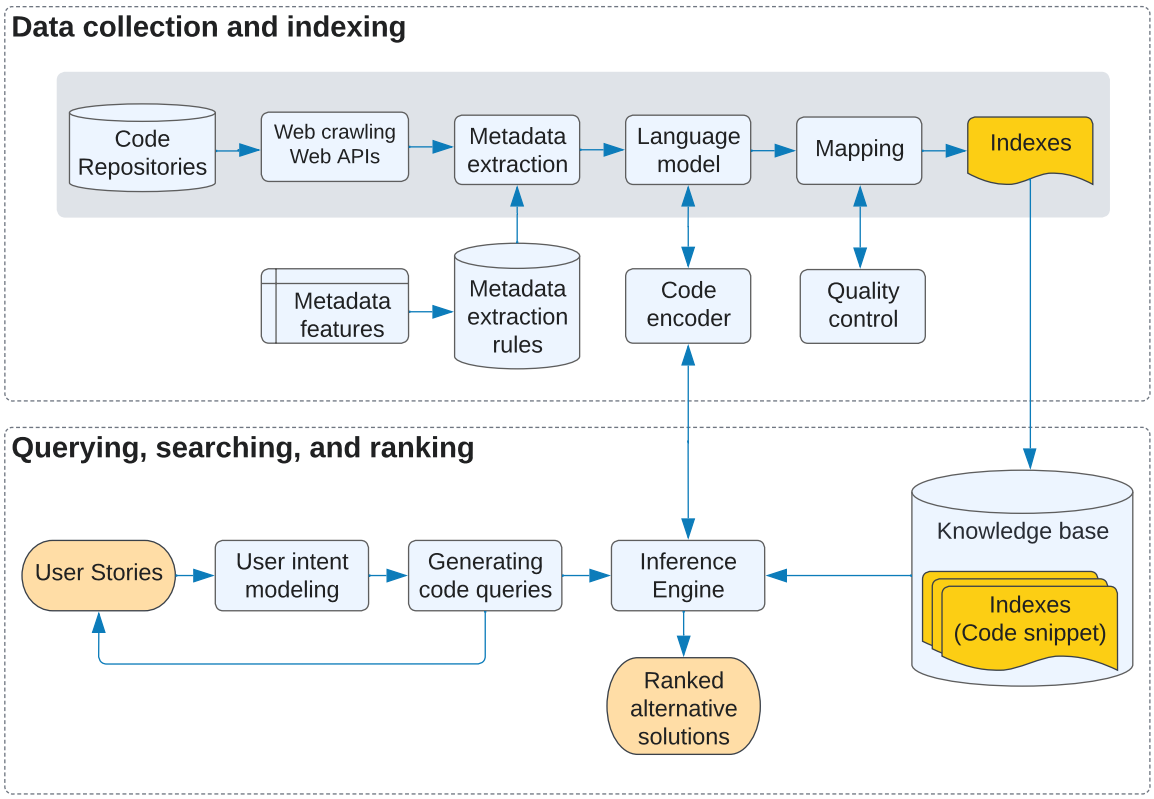


FIGURE 1.2: Conceptual framework of the code search engine

Chapter 2

Literature Review

This section presents the findings obtained from the systematic literature review (SLR) conducted as an integral part of this research. Following the SLR framework proposed by Kitchenham (2007) [66], a comprehensive collection of relevant scholarly works was gathered, forming an extensive framework that encompasses existing knowledge in the area of semantic code search. This framework is essential for understanding the scientific landscape and informing the design decisions discussed in Chapter 3.

2.1 Methodology for Conducting the Systematic Literature Review

This section outlines the methodology employed for the SLR, explaining the rationale behind the different phases selected based on Kitchenham's literature review process in 2004 [65]. The SLR aimed to answer the following research questions:

(RQ1) What are the essential components for a code search engine?

(RQ2) How can software developers' search intent be modelled to support them with the code search process?

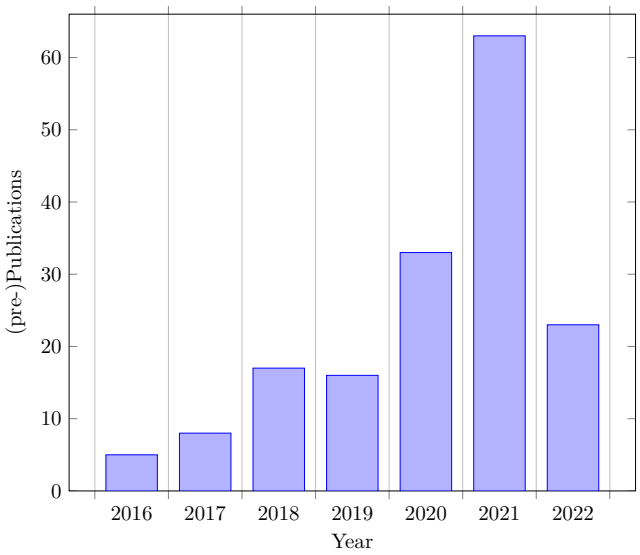
(RQ3) Which components should be employed in a code search engine?

To address these research questions, it was necessary to establish a comprehensive knowledge base, which motivated the execution of the literature study. Each subsection below describes the reproducible steps taken during a specific phase of the literature study. The search phase details the search terms used and the databases considered. The screening phase presents the criteria and procedures used for selecting relevant papers. The component analysis involves extracting data from the screened studies to acquire the knowledge required to answer the research questions. Finally, the extracted data is synthesized by organizing it into components and selecting the final set of studies that form the knowledge base for this thesis. The table below illustrates the alignment between the features of an SLR as described by Kitchenham (2004) [65] and the corresponding phases performed in this SLR.

Table 2.1: SLR Mapping

Feature	**SLR Phase**
Identification of research	Searching
Selection of primary studies	Screening
Study quality assessment	Screening
Data extraction and Monitoring	Component Analysis
Data Synthesis, Configuration and Final Studies	

FIGURE 2.1: Analyzed literature per year of publication



2.2 SLR Results

2.2.1 Searching

The searching phase represents the initial stage of the SLR, where various repositories were explored to identify relevant literature.

Figure 2.1 illustrates the distribution of analyzed literature based on the year of publication.

The majority of the papers identified were published after 2017, as the field of semantic code search has witnessed significant growth in recent years. Papers published earlier than 2017 were obtained through snowballing, where they were referenced in more recent publications, serving as a foundation for the current literature.

The following table provides documentation of the search process and the various data sources utilized:

With few exceptions, the majority of papers proceeded to the screening phase. The papers excluded from screening primarily focused on automatic code generation and the creation of code snippets from scratch. Although these papers mentioned code snippets, their main emphasis was not aligned with the research objective. The resulting papers from the searching phase are accessible on Mendeley Data, as indicated in Appendix A.

2.2.2 Screening

The screening phase encompassed the evaluation of the majority of papers identified during the searching phase. The primary objective of this phase was to gain a comprehensive understanding of the research field's development and assess the potential for addressing the research questions through the available literature. A significant portion of the papers, particularly recent ones, focused on the application of machine learning techniques to code search. For instance, one notable example is Trans3 [132], which leverages transformer algorithms to enhance indexing and code search. Another example is Cosea [129], which employs layer-wise attention mechanisms in convolutional neural networks to improve search outcomes.

TABLE 2.2: Search process documentation

Data Source	**Documentation**
Google Scholar	Google Scholar was utilized to search for high-quality papers and distinguish them from grey literature. The search period spanned from 2017 to 2022, and keywords such as Code Search, Semantic Code Search, User Intent Search, Code Indexing, Code Ranking, and Code Querying were employed. Snowballing was performed to identify relevant foundational papers. The consultation of this data source took place between April 2022 and November 2022.
Scopus	Scopus was consulted to identify additional high-quality published papers based on the data collected from search engine queries. The search period ranged from 2015 to 2022, and the same keywords used in Google Scholar were reused. The data source was consulted between April 2022 and November 2022.
ACM Digital Library	ACM Digital Library was consulted to discover technical papers with concrete implementations in the field of computer science. The search period covered 2015 to 2022, and the keywords from Scopus were reused. However, the search also incorporated specific terms related to neural networking and machine learning techniques to obtain more targeted results. The data source was consulted between April 2022 and November 2022.

Certain papers did not proceed to the component analysis phase. The inclusion criteria for this phase required that the papers primarily revolve around code search and provide a thorough analysis of both the positive and negative aspects of the specific algorithm, along with potential future research directions. Given the recent publication of many papers and their extension of earlier studies, the quality assessment of papers did not heavily rely on citations as a metric.

During the screening phase, some papers were excluded from the analysis. These papers often presented code implementations in domains such as science or chemistry, which deviated from the focus on code search. Additionally, literature proposing datasets, machine learning corpora, or other methods that could be beneficial for a code search engine but did not incorporate component features were left unconsidered during the component analysis. For instance, Codexglue [79] introduced a benchmark for evaluation purposes but did not address code snippet ranking or indexing. While these papers passed the screening phase due to their potential contribution to the engineering cycle and software artifact development, they were left out during the component analysis.

Furthermore, this phase laid the foundation for the qualitative analysis of different components and the papers themselves. Abstract takeaways were generated for all the screened papers, with some extracted directly from the paper abstracts and others developed during the screening process. These abstracts, combined with a thorough reading of the final selected studies, will facilitate a qualitative analysis of the configured code search engine examples. Further details on this analysis can be found in Section 2.2.4.

2.2.3 Component analysis

The component analysis is the major deliverable of the SLR and contains the setup for the engineering and design cycle. The Artifact Design will be based on the requirement analysis from the pipeline. What techniques are used in state-of-the-art

algorithms and methods to determine the direction in which this research will contribute? This way, the resulting list of components and the configurations created from it provide a solid quantitative knowledge base to answer the first three research questions. Figure 2.2 shows a table with the results of this component analysis. It shows the frequency of components and different potential combinations of existing techniques used in the literature.

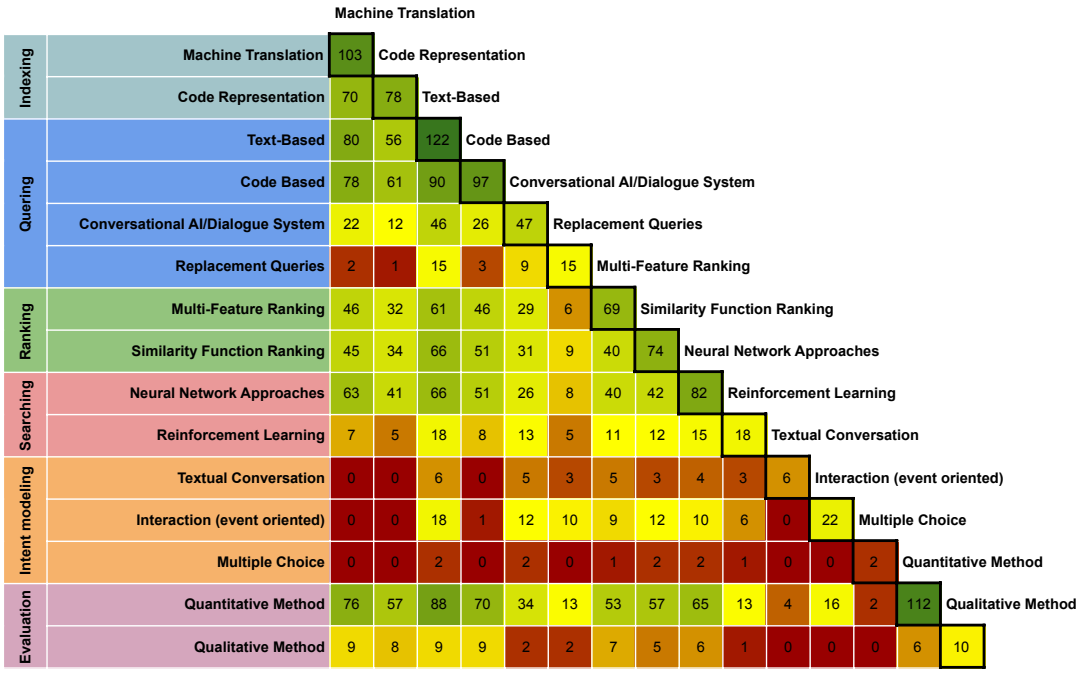


FIGURE 2.2: shows SLR results in a table abstraction

The list of components was chosen by an initial analysis and read of high-quality papers like CodeSearchNet [56] and When deep learning met code corpus[17]. These papers depict a state of semantic code search engines and their most-used components. The latter study also cites some recent publications which use techniques and set the standard for semantic code search like NCS [111], CODEnn [58], and SCS [55].

Figure 2.2 shows many clusters of text and code-based querying in combination with machine translation, neural network approaches, and multi-feature ranking. These clusters may be used in the decision-making progress on the architecture of a code search engine. An example and the analysis for this paper will be shown in section 2.2.5.

The following tables show the mapping of components depicted in Figure 2.2 to the different techniques used in different papers. This analysis is part of the component analysis as it shows what different techniques are used in what papers, which can later be used for reference. Furthermore, it shows the qualitative (dis-)advantages of using the said technique in an actual research implementation

TABLE 2.3: Techniques and their (dis-)advantages

Technique(s)	Description	(Dis-)advantages	Sources
Indexing			
		Continued on next page	

Table 2.3 – continued from previous page

Technique(s)	Description	(Dis-)advantages	Sources
Code Input encoder/decoder	Many authors [138] [2] [129] use some kind of code input encoder/decoder mechanism to store code snippets in vector space. This allows for efficient searching using the different techniques mentioned below. Many different data stores that index worldwide code snippets from sources like GitHub or StackOverflow exist [137] [19] [146] these use a combination of NLP techniques that extract semantic meaning from the code snippets and stores this.	**Advantages:** Because source code is so widely available the NLP algorithms can be trained on a very large dataset to improve results in storing the semantic meaning [146], [19], [132]. Furthermore, using these techniques allow for other relevant features to be added like code clone detection [131] and bug localization [45]. **Disadvantages:** Using an encoder to index the code snippets is only useful if the data it is trained on is useful. Garbage in means garbage out, this also applies to the query used to match the vector space, if it is of low quality then the results will not be of high quality. ([146], [2], [143])	[138] [2] [129] [137] [19] [146] [131] [132] [45]
Abstract Syntax Tree	Many studies propose the use of an Abstract Syntax Tree along with code representation learning in their deep learning models [22] [133]. Other highly cited search engines like Facoy [64] generate ASTs and use their nodes to create an index pointing at the original code snippet based on the search terms used.	**Advantages:** Broadly used method of depicting syntax of code snippets as it is used in over a third of the analyzed literature. Furthermore, it is being extended in recent studies also to add semantic value to the indexing. [22] [131] **Disadvantages:** Without AST-position and without AST-Mask-Self-attention, the performance of pre-trained indexing may decline [73]. Furthermore, AST-based approaches can not fully leverage structural-semantic information of code snippets if used in its basic form or without semantic upgrades. [131]	[22] [133] [64] [131] [73]
Control/Data Flow Graph	Control and Data Flow Graphs are a graph representation of a system that may be incorporated into a semantic matrix [153] to store the semantic and syntactic meaning of code. Control flow captures the dependence between code blocks, and data flow captures the flow of data along program operations. [153] Flow graphs may be used in conjunction with AST [131] and, because of its graph nature, lends itself well to a graph neural network approach. [148]	**Advantages:** The Flow Graph techniques lend themselves well to depicting code semantics when compared to AST. They also combine very well with graph algorithms and deep learning techniques. [148] **Disadvantages:** In their base form, control and data flow graph need to be used together, or higher-level inter-procedural control flow graphs may have to be implemented to improve baseline performance [97].	[153] [131] [148] [97]

TABLE 2.4: Querying techniques and their (dis-)advantages

Technique(s)	Description	(Dis-)advantages	Sources
Querying			
Text Input Query	Text input query is a basic query box where a user types a string to retrieve code snippets that syntactically and semantically match the textual input string. [80] [96] [129]. On top of this basic textual implementation, many studies implement things like categories, programming languages, API and other improvements to a baseline textual input string. [31] [130] [149]	**Advantages:** Text input query can be extended upon with topics and additional filters [149]. Furthermore, NLP techniques can be used to extract entities and encode the search query to find the nearest neighbours in vector space. [71] Queries can also be reformulated to provide better results in the code ranking [19]. **Disadvantages:** Textual input query is highly dependent on the user's way of typing strings, which is often not fine-tuned to work with the indexed code snippets. [71] Query reformulation [19] and conversational AI [52] may complement text input query in this manner.	[80] [96] [129] [31] [130] [149] [19] [52] [71]
Code Input Query	Code input query uses literal code snippets to search for other code snippets [64]. Implementations for this technique could be a code-to-code search engine like Facoy [64] or translating a user query into a snippet which is then encoded to look for code clones or similarities. [80] [6] [57]	**Advantages:** Code input query is more accurate at finding clones or similarities because the user is using a code snippet of its own. Therefore the query matches the indexed database better, regardless of the language used [83]. **Disadvantages:** A user could not be native in writing code and wants to search for code based on natural language. Therefore NLP transformers to code snippets may have to exist to bridge the gap between the NLP user query and the to-be-encoded code snippet. [6] [57] [11].	[6] [57] [11] [83] [80] [64]
Conversational AI	To bridge the gap between user textual input query and the ranking, we believe a conversational agent may provide valuable results. Using interaction-based user intent modelling the actual user intent may be considered when querying the codebase index [52] [1] [19]. These concepts will be discussed in the User Intent section.	**Advantages:** As there exists a semantic gap between a user's query intention and input query [19] a conversational AI may be able to close that gap by searching for the user intent in different ways.**Disadvantages:** Interacting with a conversational AI to achieve personal search may not be beneficial to the user experience of the system. Furthermore, many additional techniques may have to be utilized to improve intent modelling performance, like, for example, knowledge injection, entity linking, sentiment detection and sentence completion [103]	[52] [1] [19] [103]
Replacement queries	A technique that works in conjunction with a conversational AI and text input query is the use of replacement queries [128] [1]. This technique replaces specific parts of the textual user query to match the results better. A user may query: 'Singleton Java', which may then be reformulated to 'Singleton Design Pattern Java' to improve ranking results possibly.	**Advantages:** Improves query results [128] while not interfering with the user as much as a conversational AI. **Disadvantages:** Is still heavily reliant on the original input query by the user; if there is nothing there to match, then the system can not provide a good replacement. [21]	[128] [1] [21]

TABLE 2.5: Ranking techniques and their (dis-)advantages

Technique(s)	Description	(Dis-)advantages	Sources
Ranking			
Multi-Feature Ranking	Multi-Feature Ranking is ranking based on multiple searching/indexing/querying features. One new approach is CRaDLe [43], which highlights dependency relations and learns a unified vector representation for a code and description pair. In this way, a code snippet's syntax and semantics can be used for ranking. Other examples are PSCS [117], CommitBert [61] and CodeMatcher [76].	**Advantages:** Combining multiple code features when ranking may greatly improve the state-of-the-art in terms of score [43]. Furthermore, it is very interesting for using in a user/query intent-based code search engine [50] **Disadvantages:** To use multi-feature ranking, the multiple features must be widely available and of high quality in the indexed dataset. This is an extensive amount of work and may require additional techniques to create on a large scale [114]	[43] [117] [61] [76] [50] [114]
BM25	An algorithm to calculate the relevance of documents to a given search query. It is short for Okapi BM25, where BM means Best Matching. [107]. It has been extended with multiple new features to improve its workings [120] [107]. It is still used in modern code searches. Still, it is often improved upon with additional techniques [77] [52]	**Advantages:** BM25 is a well-established method of determining a ranking of documents, in this regard, code snippets [107]. **Disadvantages:** To reach state-of-the-art performance or compete with neural similarity functions, it needs to be extended [77] [52].	[107] [107] [77] [52]
Cosine Similarity	Cosine similarity is a function based on the cosine in math. It measures the similarity or difference between two vector embeddings - one of the query and one of the code snippet -. [17] [90]	**Advantages:** Cosine similarity is a relatively easy similarity metric and works in collaboration with many algorithms. **Disadvantages:** Cosine similarity calculation is heavily dependent on both the encoding algorithms [17] as well as the user input query [90].	[17] [90]
Custom Similarity Function	Many custom search engines use their own custom similarity function based on for example Jaccard similarity or Mean Reciprocal Rank. [140] [111]	**Advantages:** A custom similarity function is often tailored very well to its implementation purpose and is used in conjunction with advanced neural code search [111]. They are also useful in a user intent model with annotated code snippets to measure the similarity between code and query intent [50]. **Disadvantages:** As there are no possibilities of annotating each query-snippet pair with intent there are additional techniques necessary to attain useful results [114].	[140] [111] [50] [50]

TABLE 2.6: Searching techniques and their (dis-)advantages

Technique(s)	Description	(Dis-)advantages	Sources
Searching			
CNN	A convolutional neural network is a deep learning algorithm that uses convolutional neural layers [68]. The name convolutional comes from a linear mathematical operation between matrixes called convolution [3]. Besides these convolutional layers, it has a non-linearity layer, a pooling layer, and a fully-connected layer [3]. It is mainly used in image recognition [68] but can also be implemented in different areas such as code search [129]. As an addition to the base convolutional neural network, attention mechanisms were introduced to reduce computation time and improve performance [87]. These attention mechanisms add a so-called attention layer to the model, which allows it to find important features more easily [4]. In code search, it is used to train on natural language code snippets and input queries [129].	**Advantages:** Convolutional models are a natural choice for learning translation-invariant features with a small number of parameters [4]. Furthermore, they can be used for sentence classification [152] to model user intent and combine that with the existing search for code snippets. **Disadvantages:** A standalone convolutional architecture is computationally expensive and requires attention layers or other additions to stay efficient [87]. Some of these advanced deep learning models can also take a week to train [136] and require much tuning to provide useful results.	[68] [3] [129] [87] [4] [152] [136]
RNN	The recurrent neural network differs from a traditional feed-forward neural network in that it has at least one feedback loop [46]. This means that at least one of the network's neurons has its output fed back into the input of an earlier neuron. This enables the modelling of Bi-Directional Long Short-term Memory networks [51], to model interactions between words of either the input query or the code snippets themselves [144].	**Advantages:** RNN architecture enables several code summarization tasks to bridge the gap between semantic user intent and code snippet semantics [144]. Furthermore, it works with pre-trained code models like CodeBERT [36] and is often used in the CodeSearchNet Challenge [56]. **Disadvantages:** It has difficulties capturing long-range dependencies which may occur in code [50]. This is why it is used in more recent studies in conjunction with a transformer-based architecture [50], [94] [82].	[46] [51] [144] [36] [56] [94] [82]
GNN	Many relationships among data can be represented in terms of graphs, which model a set of objects (nodes) and their relationships (edges) [112] [155]. [112] proposed a new neural network model called Graph Neural Network (GNN) that extends existing neural network methods for processing data represented in graphs [112]. They have as a group been categorized into different categories of graph neural networks, being convolutional graph neural networks, recurrent graph neural networks, graph autoencoders and spatial-temporal graph neural networks [135]. These networks combine the advantages of using graph neural networks with the advantages of both recurrent and convolutional neural networks to improve quality [157].	**Advantages:** A graph representation captures several mechanics of code snippets that other indexing methods or neural networks cannot capture [136]. Examples of these are using the full AST graph and other code features modelled as a graph as input for training [23] or using graph representations to generate documentation automatically, and therefore semantics from code snippets [78]. **Disadvantages:** The use of a graph neural network requires the input data to be in a graph form. This requires the use or creation of graph embeddings [108] or pre-trained graph models like GraphCodeBert [44] or flow graph building like deGraphCS [148].	[112] [155] [135] [157] [157] [23] [78] [108] [44] [148]

Continued on next page

Table 2.6 – continued from previous page

Technique(s)	Description	(Dis-)advantages	Sources
FFNN	Feedforward neural networks are a simpler form of neural networks compared to recurrent neural networks [86]. They were one of the first forms of Artificial neural networks (ANNs) based on the structure of a real brain, using neurons [121]. Each of these neurons can receive input signals, process them and send an output signal to the next layer or single neuron [121]. It is a supervised learning technique in which the network is trained based on the desired output from a given user query, and the weights of different neurons are trained [121].	**Advantages:** Simple and easy to understand and train compared to other models [46]. It can be extended using multiple layers or networks and is always one-directional [46]. **Disadvantages:** Most newer neural network models improve upon the flaws and simplicity of FFNNs [86] [87]. Therefore the state-of-the-art has moved past most simple FFNN models. [46]	[86] [121] [46] [87]
Reinforcement Learning	Reinforcement Learning is the problem an agent faces that must learn behaviour through trial-and-error interactions with a dynamic environment [62]. It is also one of three main machine learning paradigms: supervised, unsupervised, and reinforcement learning [70]. Reinforcement learning problems always involve interaction between an active decision-making agent and its environment to reach a certain goal [119]. For a code search engine, this maps to a developer interacting with a code search engine to retrieve a code snippet that matches the developer's intentions.	**Advantages:** Reinforcement learning improves over time based on interaction with the system, making it a useful technique to use in conjunction with user intent modelling (interaction-based) as the quality of the system improves over time based on user feedback [72]. Furthermore, reinforcement learning could bridge the semantic gap between a user query and the code search itself, as shown in QueCos [128]. Most code search applications do not consider this in improving their algorithms [128]. In addition, RL may also be used in conjunction with neural network algorithms for automatic code summarization [127]. **Disadvantages:** As user queries are often short and ambiguous [141], a personal search process may have to be implemented to improve the system's results [141].	[62] [70] [119] [72] [128] [141]

Since User Intent is a fundamental aspect of this study, it has its own dedicated research question, and as a result, the User Intent techniques table differs from the others. Initially, the table was organized into three domains of user intent modelling:

- Textual Conversation - Textual conversation with an AI serves as one approach to capturing user intent within an information system. In this method, a search engine can model a user's intent based on the formulation of a query string, utilizing various taxonomies [59, 16]. Subsequently, the system can respond by seeking clarification or providing an appropriate answer based on the user's phase in the personal search process [139].
- Interaction (event-oriented) - These user intent modelling techniques rely on user interaction data instead of engaging in textual conversation [102]. Statistical approaches such as Markov Models [110] or Matrix Factorization [48] can be particularly beneficial in this context. This approach holds promise, especially in exploratory search domains [109], where users may not have a clear idea of what they are searching for. Considering the nature of a code search engine, where developers often encounter situations where they lack precise knowledge of their desired outcome [104], this attribute becomes highly valuable.
- Multiple Choice - Multiple-choice intent modelling systems present the user with various potential results, from which they can select one or multiple options that align with their intent. Based on this user behaviour, the system models the user's intent, enabling more exploratory search capabilities [109]. Moreover, it facilitates the use of hidden semantic information to construct supervised signals for intent feature learning [150].

These domains served as a foundation for identifying relevant algorithms, methods, or techniques to be applied to the code search domain. Table 2.7 presents the outcomes of employing this approach.

TABLE 2.7: User Intent techniques and their (dis-)advantages

Technique(s)	Description	(Dis-)advantages	Sources
User Intent			
Textual Conversation			
Utterance/Dialogue Classification	Many different statistical techniques exist for Utterance/-Dialogue classification. Support Vector Machines or Hidden Markov Models are used for many classifications in the field [118] [100]. These classify a user's text input into different categories and respond accordingly, increasing the probability of matching the ranked document to the user's intent.	**Advantages:** By classifying the queries, they can be responded to and improved for document retrieval [118]. By satisfying an information need before the actual ranking of results, many users may already be satisfied or pushed in a different direction than expected [63]. **Disadvantages:** With trying to classify the utterance or dialogue come unique challenges because of the complexity and diversity of human information-seeking conversations [100].	[118] [100] [63]

Continued on next page

Table 2.7 – continued from previous page

Technique(s)	Description	(Dis-)advantages	Sources
Conversational Search	In the conversational search system, the user is unsure exactly what kind of code snippet they are looking for. Oddy [91] was one of the first to explore this concept by retrieving information for the user through dialogues without having the user formulate a query [100]. Qu et al. [100] focus primarily on predicting user intent in an information-seeking setting. The natural language dialogue this form of user intent modelling brings could also be highly beneficial in a code search setting since the user does not need to provide a query at all but is given code based on just a natural language description of the user's intent. With recent progress in machine learning algorithms, this approach becomes increasingly more promising in user intent modelling [101].	**Advantages:** When users are unfamiliar with the domain they are searching and would rely on effective interactions with the search engine, conversational search is incredibly effective [100]. **Disadvantages:** With the user never actually typing a query, there may be much slack in getting the final code snippet. Even though user intent will be aligned with the final result, it may not be fast enough for the developer. Furthermore, the system may not be able to incorporate prior knowledge about the corpus of code snippets in a practical scenario [101]	[91] [100] [101]
Multi-turn Question Answering	Is a system where the user asks questions and is then constantly given an answer or return question based on the nature of that question [145].	**Advantages:** Many QA sites like Stackoverflow or Pastebin on source code exist, which allows for these ready-to-use question and answer databases to be used [53] [29] [38] [143]. **Disadvantages:** It is practically impossible to store all the knowledge in a neural network to achieve desired precision and coverage in real-world QA [145]. In the field of just source code, this problem is not infinitely large, which is why it may be possible. Especially with mining of QA pairs from StackOverflow [38] [143]	[145] [53] [29] [38] [143]
Term Dependency Modelling	Sometimes also depicted in literature as term proximity or term frequency [47], term dependency modelling is a method of collecting intent by retrieving information from the query based on terms used [113]. Older papers proposed using Markov Random Fields to model this term dependency and probability [85]. Later research started considering the independency of terms used and moved to include phrases on top of single terms to see terms in a more user-intent context [113]. These phrase-based approaches extend upon earlier single-term studies.	**Advantages:** Possibility to add to existing models and easier to understand than applying neural networks [113]. **Disadvantages:** Do not have the possibilities neural networks have when storing semantic values. However, it could be extended by using the semantic model presented in [113]	[47] [85] [113]
Interaction (event oriented)			

Continued on next page

Table 2.7 – continued from previous page

Technique(s)	Description	(Dis-)advantages	Sources
Attribute Aggregation	Is the process of aggregating the attributes of the document (semantic or syntactical code snippet features) to the personal query attributes [10]. This could be modelled in a query document graph, which is close to the equivalent of a query snippet model in code search [143].	**Advantages:** Attribute aggregation is comprehensive and can be improved upon by considering many specific features from the documents [10]. Examples include using CTR, Access Counts, Cursor tracking, or even touch gestures as part of the user query attribution [106] [10] **Disadvantages:** It is highly dependant on the quality of the indexed documents, in this case, code snippets, as well as how to query input is facilitated for its functionality [10]. Dialogue classification could be combined with this attribute aggregation to improve matching results.	[143] [10] [106]
Deep Learning Techniques	Many deep learning techniques exist to model the interaction between the user and the search engine. Some examples are Multilayer Perceptron, Autoencoder, CNN, RNN, and Attentional Models [151]. These options are discussed in further detail in Zhang et al. paper on deep learning-based recommender systems [151]. The neural networks that are relevant and overlapping with code search are described in their respective section in Table 2.6.	**Advantages:** Deep learning techniques are more recent and state-of-the-art when compared to relatively older machine learning techniques. In general, they achieve a very high-level accuracy when compared to older machine learning models [151] [141] [88][39] **Disadvantages:** As with many deep learning techniques transparency of how the algorithm work is essential. Furthermore, implementing these systems is resource-heavy and requires mathematical optimization to become practically viable [86][121].	[151] [141] [88] [39] [86] [121]
Multiple Choice			
Deep Aligned Clustering	While many models label intent based on extracted intent features for a single case, Deep Aligned Clustering leverages the prior knowledge of known intents [150]. Combining BERT [28], knowledge transfers from known intents with limited labelled data and an alignment strategy to provide signals for learning clustering-friendly representations [150].	**Advantages:** Deep Aligned Clustering successfully transfers the prior knowledge of limited known intents and estimates the number of intents by eliminating low-confidence clusters [150]. Using these clustering techniques, it is way faster at discovering the approximate intent of a user when compared to other methods. **Disadvantages:** Learned features of this method are much more biased towards labelled data and require the prior knowledge of known intents to be of high quality. Otherwise, the performance of the algorithm drops dramatically [150]	[150] [28]

2.2.4 Decision Making

This section outlines the decision-making process for constructing an example code search engine configuration based on the findings of the conducted SLR. The process involves quantitative and qualitative analyses, which will be further explained in Section 2.2.5 and Section 2.2.6, respectively.

2.2.5 Quantitative Analysis

Figure 2.2 presents a table displaying the extracted components from the analyzed papers. By examining the numbers within each cell, one can determine the frequency of occurrence for each component in the literature study. Additionally, the table illustrates how often components are mentioned together. For instance, Machine Translation is mentioned in 103 of the analyzed papers, and in 78 of those instances, it is combined with a Code Based querying mechanism.

Based on the insights provided in Figure 2.2, the components can be combined in various configurations to evaluate their effectiveness when applied together. For instance, if one chooses to incorporate Neural Network Approaches in their code search engine, one can observe that 66 papers describe this combination with a text-based user input query. At the same time, 55 articles mention its combination with code-based querying. This quantitative analysis guides the development of a configuration based on the number of components, partially addressing the first research question: "What are the essential components for a code search engine?".

Figure 2.3 showcases this thesis's chosen combination of techniques. As depicted, all the techniques are prominently mentioned in most analyzed studies. Although the interaction-based intent modelling technique appears less frequently than the other techniques, it is still derived from 22 of the 30 intent modelling papers in the literature study. It is worth noting that the absence of papers mentioning the combination of machine translation and interaction is because the 30 analyzed papers did not discuss code search engine indexing methods, indicating a gap between code search and user intent studies. The specific applications of these techniques will be elaborated on in Section 2.2.6 during the qualitative analysis.

Components	Techniques	Machine Translation	Text-Based	Multi-Feature Ranking	Neural Network Approaches	Interaction (event oriented)	Quantitative Method
Indexing	Machine Translation	103	78	46	63	0	76
Quering	Text-Based	80	122	32	66	18	88
Ranking	Multi-Feature Ranking	46	32	69	40	9	53
Searching	Neural Network Approaches	63	66	40	82	10	65
Intent Modelling	Interaction (event oriented)	0	18	9	10	22	16
Evaluation	Quantitative Method	76	88	53	65	16	112

FIGURE 2.3: shows the design for code search engine based on component analysis

2.2.6 Qualitative Analysis

This section describes the qualitative analysis and decision-making for the final design of the code search engine. Based on the qualitative analysis performed in table 2.3, 2.4, 2.5, 2.6 and 2.7 a set of applications were considered to incorporate in the practical design for the code search engine.

Based on the (dis-)advantages of several studied techniques, many different choices regarding the component configuration of the code search engine may be made. Figure 2.2.6 depicts the choices made for this thesis' design based on these (dis-)advantages. Other configurations could also be chosen based on the analyzed methods from the literature.

Components	Techniques	Selected methods	
Indexing	Machine Translation	Code Input Encoder/Decoder	Abstract Syntax Tree
Quering	Text-Based	Conversational AI	
Ranking	Multi-Feature Ranking	Intent Based Ranking	
Searching	Neural Network Approaches	Graph Neural Network	Reinforcement Learning
Intent Modelling	Interaction (event oriented)	Conversational Search	Attribute Aggregation
Evaluation	Qualitative Method	Technology Acceptance Model	

FIGURE 2.4: shows chosen methods based on qualitative analysis of code search components

The rationale behind selecting these methods is as follows:

The combination of Code Input Encoder/Decoder with AST (Abstract Syntax Tree) was chosen for the indexing component. This approach enables a semantic and syntactic representation of indexed code snippets [131] [22]. Additionally, this combination lends itself well to graph representation, allowing for using a graph neural network in the search process.

Conversational AI was chosen for the querying component, utilizing an interaction-based intent modelling technique. By incorporating elements from conversational AI, the aim is to enhance exploratory search capabilities [30]. Furthermore, attribute aggregation [10] was selected to assign an intent score to user interactions with the system, providing a more comprehensive understanding of user intent.

The searching component will employ a graph neural network (GNN) architecture to search through semantically extended ASTs (Abstract Syntax Trees) [22] [131] for relevant code snippets. These snippets will then be ranked using multi-feature-ranking techniques such as CRaDLe or CodeMatcher, depending on which algorithm performs best within the chosen configuration.

Since a Conversational AI approach is used, combined with intent modelling, reinforcement learning is also selected. This combination addresses a major drawback in reinforcement learning for code search engines, as user queries are no longer short and ambiguous [141]. Instead, they are supported by an intent model, providing more context and guidance.

Based on the studies conducted in the literature and the information presented in the tables above, the final code search engine will be developed using these selected techniques.

2.2.7 Final Design

Based on analyzing qualitative and quantitative metrics, an example configuration has been created with its respective reasoning. They are all based on applying some user intent analysis or improving upon the query/ranking relationship, as this thesis's leading research goal. The following table shows the different configurations with references to their respective reasoning. The decision-making for a final search engine based on the literature study has two sides, quantitative and qualitative. Firstly, a quantitative analysis was performed to see the broad spectrum of code search engines and their components available in the literature, as seen in figure 2.2. Secondly, based on qualitative data from the papers, this component analysis was specified for the

different applications as they were being used in the literature. This concludes in a final search engine design based on figure 2.4. The extensive design and architecture will be described in the final thesis document. Furthermore, this design will be implemented in a real-life software artifact to evaluate the system and therefore answer the last research question: "How can the code search engine be evaluated?"

Figure 2.5 shows the practical choices regarding the final design regarding practical applications used in the code search engine. These practical applications were taken from the literature and will be implemented in the final software artifact. The practical applications are based on the methods and techniques depicted in figure 2.3 and figure 2.4. Chapter 3 will go into further detail on how these literature study findings were adopted in the design science software artifact.

Components	Techniques	Selected applications
Indexing	Machine Translation	CodeBert
Quering	Text-Based	RASA AI
Ranking	Multi-Feature Ranking	kNN with weighed intent query
Searching	Neural Network Approaches	CodeBert
Intent Modelling	Interaction (event oriented)	DIETClassifier

FIGURE 2.5: shows chosen applications based on qualitative analysis of code search components

2.2.8 Research Directions

The summarized results above provide an overview of the current research trends in the field of Code Search. Notable contributions, such as CodeSearchNet by Husain et al. [56], has elevated the standard of code search by leveraging machine learning techniques for semantic code retrieval instead of relying solely on keyword-based search. Furthermore, researchers have made significant efforts to create datasets, such as PyTorrent [7], Codexglue [79], and Staqc [143], to facilitate future research and benchmark the performance of code search engines.

A significant focus of the literature is finding optimal ways to encode and decode code snippets to capture the developer's intention accurately. Many researchers have explored approaches that combine or generate code snippets with comments or annotations, integrating syntax and semantic descriptions in vector space [142, 50, 82]. Various machine-learning techniques have also been investigated and proposed to effectively transform code snippets into vector representations. Several approaches utilize pre-trained models to incorporate new code snippets into the existing vector space of code snippet models, such as GraphCodeBert [44], CodeBert [36], and ProphetNet-x [98]. The VRE search engine [154] offers an advanced interface for searching research assets within the Virtual Research Environment (VRE) community, employing information retrieval and extraction techniques, including textual content analysis, topical coding, pattern clustering, and topic models [35]. Additionally, there is research focused on enhancing code snippet indexing using advanced neural networking and encoding techniques, exemplified by Code2Vec [6], Code2Seq [5], and deGraphCS [148].

2.2.9 Gap Analysis

A gap exists in considering the importance of user queries in achieving accurate rankings. While research emphasizes improving code encoding techniques, inadequate query formulation and translation to vector space often lead to inaccurate results that do not align with the user's intent. A Neural Code Search (NCS) study by Sachdev et al. [111] found that in nearly a quarter of cases, the query itself was the obstacle in obtaining the desired answers. Query alternation, as demonstrated in code-to-code search engines like FaCoY [64], has been employed to address this issue. However, the utilization of replacement queries or conversational AI techniques is infrequent, with less than half of the studies in the SLR incorporating them. We believe that by leveraging an interaction-based dialogue system, users can refine their queries through an interactive conversation, thereby improving the quality and precision of the vector-encoded question. When the query is more specific and well-defined in vector space, similarity functions have a better chance of ranking similar code snippets higher, ultimately enhancing the code search experience.

2.2.10 User Intent Modelling

This section contains findings on User Intent Modelling and its appliance to the code search field. Many papers on user intent have been studied to answer the second research question. This study aims to validate whether these modern user intent modelling techniques hold in a code search environment. This is why generic user intent modelling literature on search engines and recommender systems was used for the literature study—treating a code search engine as a recommender system for code snippets. Table 2.7 shows some techniques used to model user intent in recommender systems; these techniques will be used in the software artifact design for this thesis. By incorporating these user intent modelling techniques into the software artifact design, the thesis aims to develop a code search engine that effectively captures and understands user intent in the code search process. The goal is to provide relevant and accurate code snippet recommendations to improve the code search experience for developers.

2.2.11 Final Studies

In conclusion, the performed systematic literature review (SLR) has provided comprehensive insights and addressed the research questions in the context of user intent modelling for code search. The selected studies for the master thesis have contributed to answering the research questions as follows:

2.2.12 RQ1: Analysis of Search Engine Categories

This research question is answered based on the different categories studied for search engines during the SLR. Each category from the component analysis was utilized, and relevant literature was explored. Many code search engines investigated in this SLR incorporated one or more components from the different categories, as depicted in Figure 2.2. Di Grazia et al. [41] recently published a comprehensive analysis of 109 code search articles, highlighting various techniques for indexing, querying, searching, and ranking the search results. This observation confirms the widespread use of these categories in various code search engine studies. The implementation details of these categories are described in the following sections.

Indexing

Effective indexing of code snippets in a code search engine requires consideration of both semantic and syntactic aspects. Several studies, such as [138] and [6], have presented different approaches to representing code features using vector representations. An encoder system combined with an abstract syntax tree (AST) is commonly employed, where the AST captures the syntax of a code snippet, and the semantics are captured using natural language processing (NLP) encodings. Various code snippet encodings have been proposed, including pre-trained embeddings such as CodeBert [36] or GraphCodeBert [44], as well as other techniques [75, 138]. Additionally, storage improvements like deep hashing [42] have been explored to enhance the speed of the code search engine.

Querying

Querying is a prominent aspect of this research, particularly in dialogue systems and user intent modelling. The literature reveals room for improvement in this area [52]. Most studies mention text and code-based input queries in conjunction with each other. A possible approach is to combine textual search queries, similar to those provided by Google, with code input involving variables and methods. Replacement queries were scarcely mentioned in the literature, while dialogue systems to improve the querying process are gaining popularity. This thesis aims to contribute to advancements in this area.

Searching

Modern code search engines extensively employ advanced neural networking and machine learning techniques to enhance the search process. Many research papers focus on improving existing search algorithms by utilizing approaches such as graph neural networks [131], self-attention mechanisms [32], and enhanced vector indexing [5, 6]. Searching techniques go hand in hand with indexing approaches, as the quality of the indexed data structure directly impacts the performance of deep code search [17]. Refactoring the input can also improve search, emphasizing the need for cohesive collaboration among all components in a code search engine to provide optimal results [41].

Ranking

Ranking schemas have been introduced by Niu et al. [90] to rank code examples based on user queries. The proposed ranking algorithms are closely tied to the nature of the query and the underlying search mechanism. Users can even train ranking algorithms to improve the quality of the ranking itself. Furthermore, the transparency of the algorithm's base functionality is becoming increasingly important, emphasizing the need for an interpretable approach rather than a black-box solution [125].

User Intent

User intent modelling has been incorporated into this study's code search engine component list. Validating the applicability of existing user intent modelling techniques in a code search engine and improving the overall user experience is one of the research goals of this study.

2.2.13 RQ2: User Intent Modelling Techniques

Various techniques exist to assist users in finding items of interest, traditionally based on past observed behaviour and presenting a ranked list of suggestions [59]. However, more recent approaches leverage advanced transformer neural networks [139] to model intent in conversational systems. Although these approaches have been successful in conversational programming scenarios, which emphasize AI code generation, they may not be directly suitable for a code search engine focused on code snippet recommendation. As highlighted in a recent paper by Al-Hossami et al. [52], analyzing modern AI algorithms such as GPT-2 and GPT-3, there is a need to adapt these techniques for code search engines. To apply machine learning intent modelling frameworks, RASA AI's DietClassifier [15] will be utilized in this thesis.

2.2.14 RQ3: Configuration of Components

Quantitative and qualitative analyses were conducted in the decision-making process to answer the third research question. Various code search engines, including FaCoY [64] and those participating in CodeSearchNet [56], employ different combinations of the analyzed components to improve evaluation metrics such as precision and recall. This thesis aims to answer the third research question by implementing a configuration of components and evaluating its performance in a real-life scenario, explicitly investigating whether user intent modelling enhances the code search experience. Figure 2.3 and Figure 2.4 provide an abstract overview of how these components can be combined. This SLR aims to guide future researchers in constructing their configurations based on the findings in the literature. Figure 2.6 shows the conceptual model for the intent-enhanced code search engine based on the knowledge base from the literature study. Chapter 3 goes into a further explanation on the practical implementation of the software artifact based on this conceptual framework.

Overall, this SLR has explored user intent modelling techniques, validated their applicability in a code search environment, and provided insights into the configuration of components for an improved code search engine. The selected studies and findings advance the code search field and offer a foundation for further research.

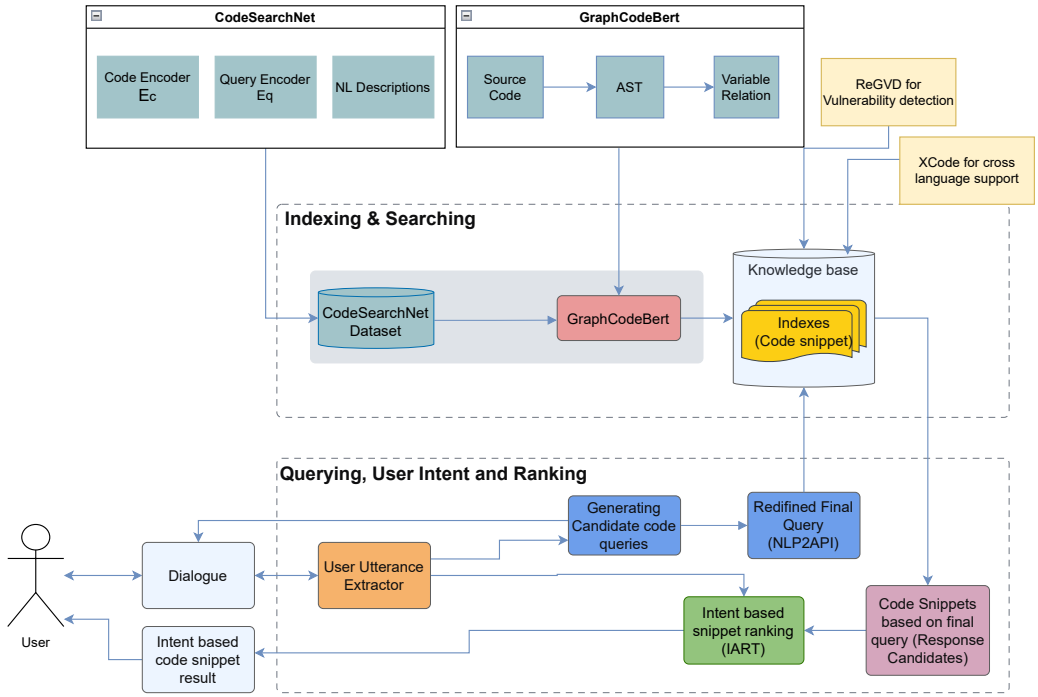


FIGURE 2.6: shows the conceptual model for the intent enhanced code search engine based on the literature study

Chapter 3

Research Method

This chapter provides a comprehensive overview of the research methodology employed in this study, which is divided into two main sections. Firstly, we discuss the design science methodology proposed by Hevner [49] and Wieringa [134] and its application within the context of this thesis. Secondly, we describe the experimental design and procedure.

3.1 Design Science Methodology

In this section, we elucidate the design science methodology adopted for this study, encompassing the iterative design process and the evaluation criteria. The research adhered to the design science research guidelines, following the subsequent steps:

1. Identification of User Needs and Requirements: We initiated the research by identifying the specific needs and requirements of the users. This step aligned the design science research with the environment by conducting a stakeholder analysis proposed by Wieringa [134]. For this study, we focused on regular operators, commonly known as end users, who possess a high level of specificity in their code search queries [8].

2. Design and Development: The design cycle played a pivotal role in creating the code search engine software artifact. Initially, we developed a simple minimum viable product (MVP) prototype, which was then iteratively evaluated and enhanced. This approach aligns with the design cycle proposed by Hevner [49]. The design prototype was implemented using Python, Vue, and several frameworks. The complete source code of the software artifact is available on GitHub for reuse and future studies [40].

3. Evaluation of the Prototype: The evaluation of the prototype primarily involved internal discussions, but we also considered peer reviews from colleagues and other business informatics students. Domain experts played a crucial role in the evaluation process. The experiment, described in Section 3.2, provides an example of how the prototype could be assessed during field testing to align with the current environmental requirements. The evaluation encompassed experienced as well as less experienced coders to ascertain the value added by the intent-enhanced code search engine. This experiment served to validate the adequacy of the established requirements and identify areas where the software artifact could be further improved.

4. Iterative Refinement: During the development of the software artifact, various methods of iterative refinement were employed. Given the evolving nature of the knowledge base on semantic code search, we emphasized the modularity

of different components of the search engine. This modular design allows for the seamless integration of new ranking, searching, or indexing algorithms created by the scientific community. Moreover, we considered the application of reinforcement learning principles to enhance the existing machine learning algorithms utilized in the prototype. Specifically, we explored the utilization of human-in-the-loop reinforcement learning, also known as Reinforcement Learning Human Feedback (RLHF) [92].

3.1.1 Engineering Cycle

To implement the design science methodology effectively, we initially employed the engineering cycle [134]. This cycle entails the following steps [134]:

- Implementation Evaluation = Problem Investigation: This step involves understanding the problem within its context, which was facilitated by the Systematic Literature Review (SLR) conducted in this study. It also encompasses modeling stakeholders, causes, and effects and their contributions to the established goals.
- Artifact Design: In this step, we mapped the requirements of the software artifact within its context. The design choices were based on the goals identified in the previous step and drew from earlier literature, as presented in Chapter 4.
- Artifact Validation: This step involved validating whether the software artifact, within its context, fulfilled the established requirements. It also assessed the artifact's usability in different contexts and examined potential trade-offs resulting from design choices, such as compatibility with various operating systems or software developer preferences.
- Artifact Implementation: The final step of the engineering cycle encompassed the deployment of the software artifact in its problem context, allowing stakeholders to utilize it effectively. In this case, it entailed the concrete implementation and adjustment of the application in the real world.

These steps are not necessarily performed sequentially and can be interchanged as needed.

3.1.2 Design Cycle

We arrived at the design cycle by excluding the Artifact Implementation step from the engineering cycle. This cycle comprises all the steps within the engineering cycle except for the implementation itself. The design cycle can also be applied to existing systems in the literature. A gap analysis can be conducted to evaluate these existing systems using our newly identified requirements and design, thereby enhancing the final artifact's scientific value. Unlike evaluation, validation is performed before the implementation of an artifact and while it remains within a real-world context.

3.1.3 Empirical Cycle

In addition to the design cycle, we employed the empirical process to acquire knowledge regarding our design within its context. This process aimed to answer knowledge questions arising from our system's implementation. These knowledge questions,

along with the research and knowledge goals, evolved through iterations of the design/engineering cycle. The empirical cycle encompasses the following steps:

- Research Problem Analysis: This step involves defining the research questions and conceptual framework within the research's context. It also determines the target population for the study, as research outcomes may vary based on the population used. In our case, distinctions may exist between scientific software engineers and developers in the field regarding their utilization of the artifact.
- Research and Inference Design: This step entails designing the research itself, specifying the objects of study and the characteristics of the created artifact. It also outlines how results will be measured for given tasks. For example, the ranking capability could be evaluated by analyzing the accuracy of the system's responses to a set of queries.
- Design Validation: In this step, we validate the choices made in the research design, analyzing whether the research would be effective and identifying the contexts in which it may be more or less useful.
- Research Execution: This step involves implementing the designed research to observe and analyze the results within the designated context. It investigates whether the results align with the initial hypotheses and identifies potential reasons for any deviations.
- Data Analysis: In this final step, we evaluate the results and apply statistical analysis to the acquired data. This analysis enables us to interpret the results, make explanations and generalizations, and provide answers to the research questions. If the results are unsatisfactory, additional analysis is required to address the research problem.

The empirical cycle is reiterated in conjunction with the design cycle, allowing for the emergence of new research problems and knowledge questions, which can be addressed in subsequent cycles.

3.2 Experiment Setup

This section describes the experiment setup, including the study design, participants, and data collection methods. The following steps were taken:

1. Dataset creation for Baseline and Intent modelling search
2. Participant recruitment and assignment of questions and instructions
3. Experiment Design and Procedure
4. Data collection and analysis
5. Technology Acceptance Model 2

These steps will be elaborated on further in the following subsections. The main goal of the experiment is to answer the fourth research question *How can the code search engine be evaluated?* and thereby contribute to answering the main research question *How can code search engines understand software developers' intentions?*.

3.2.1 Dataset creation

The StackOverflow-Question-Code-Dataset (StaQC) [143] along with CosQA [54] were used as an evaluation dataset for both the baseline semantic search engine as well as for the intent evaluation. StaQC contains 148417 Python question-code pairs. Of which 85,294 single-code answer posts were used to train the vector embedding machine learning algorithm. These 85,294 were indexed into an elastic search database using CodeBert [36] along with the 20.000 code snippets in CosQA. The CodeBert pre-trained model was finetuned by a user named hamzab on the natural language code search task on Huggingface.

3.2.2 Recruitment of participants

For the recruitment of participants, several inclusion criteria were taken into consideration. The participants should be in the intended use category for the search engine.

- Participants should have at least two years of experience with coding, preferably in Python, to assess both the queries and returned code snippets by both systems.
- Participants should be at least 18 years old.
- Participants should work on code at least weekly to ensure they know about recent developments and regularly use code in practice.
- Participants should acknowledge they search for code whenever they are coding in one way or another, either through Google, ChatGPT, StackOverflow or other tools.

After meeting these requirements, the participants are asked to fill out a consent form to have their data anonymously used in the analysis. Both in storing their conversation data in the MongoDB system and analysing their answers in the technology acceptance model 2 questionnaire.

3.2.3 Experiment design and procedure

This section describes the experiment that will be performed based on the creation of the software artifact. The experiment is an example of how the software artifact could be evaluated in practice and how it could be improved to improve the scientific knowledge base it is based on. The following subsections will describe all experiment elements, the detailed design, the choice of queries and the practical procedure.

Experiment Design

The experiment will be conducted using a within-subjects experiment design, where all participants use both a baseline semantic code search engine and the intent-enhanced code search engine. The participants will receive a list of queries based on different categories as described in 3.2.3. They will then use these queries in both search engines and assess the results by giving qualitative feedback for each query. After this iterative process, the participant is asked to complete the TAM2 questionnaire to evaluate the intent-enhanced code search engine. The combined analysis of the experiment as well as the questionnaire, will be the evaluation of the intent-enhanced code search engine.

Query Selection

The pros and cons of using pre-determined queries and task-based interaction were weighed to assess the usefulness of the intent-based search engine compared to the baseline. As both seem like viable options to evaluate the use of a code search engine in practice, the research objective has to be central in the choice for the experiment. As the research objective is to evaluate the use of intent and entity modelling on the semantic code experience of software engineers, the logical choice seemed to isolate the intent/entity variables as much as possible in the study. Using a programming task-oriented experiment could result in the users randomly looking for things in their own different way, therefore introducing much noise to the evaluation process. This is the main reason the choice was made to present the user with a list of queries they can reference to search for results in both search engines. Simultaneously the users should not just all use the same possibly biased list of queries, which is why based on these categories and example queries, the users will be allowed to imagine three queries based on their own experience. The users then evaluate the results and provide feedback on whether they are satisfactory to their query. This allows the software artifact to apply reinforcement learning and improve its effects in the long term for these types of queries. The following queries were selected and categorized based on the kind of operations the query is asking for:

1. Category 1: Data Conversion
    - Convert int to string
    - String to date
    - Convert a date string into yyyymmdd
    - Serialize/Deserialize JSON
    - Convert string to number
2. Category 2: Data Manipulation
    - Sort string list
    - Deducting the median from each column
    - Get unique elements
    - Filter array
    - Group by count
3. Category 3: Data Storage and Retrieval
    - Save list to file
    - Connect to SQL
    - Read text file line by line
    - Extracting data from a text file
    - How to read .csv file in an efficient way?
4. Category 4: Network Communication and Security
    - AES encryption
    - Custom HTTP error response
    - Sending binary data over a serial connection

- How to get HTML of website
- Get current IP address

The categories were chosen based on a semantic cluster analysis and topic modelling to see the most occurring keywords in each cluster. This process will be explained in the following subsection.

3.2.4 Cluster Analysis

As the code snippet dataset after indexing consists of vectors with 768 dimensions, it is possible to create a clustering based on the semantic vector representation of the code snippets. One of the most well-known and used algorithms to do this unsupervised is k-means clustering [74]. However, as there are more than 100.000 code snippet examples in the experiment dataset, all with their 768-dimension vector representation, it would require out-of-this-world computational power. Therefore dimensionality reduction methods were considered. By reducing the number of dimensions to the top x dimensions that capture the most semantic meaning, it is possible to create clusters based on these dimensions. Using the dimensionality reduction technique Principal Components Analysis (PCA) [93], the number of dimensions for each item was reduced from 768 dimensions to only the first three dimensions to create a 3d visual of the clusters.
First, the elbow method was used to determine the optimal number of k clusters [24] for the k-means algorithm. Then based on this optimal number, k-means clustering was applied. When all clusters were created, the highest frequency of words for each cluster was determined. Based on these words and snippets, the above categories were defined for use in the experiment. No matter the input dataset, combining these technologies allows for the unsupervised cluster analysis of the index.

Experiment Procedure

The following list shows the procedure performed for each participant within the experiment. They will be done in private one-to-one sessions to avoid bias from multiple users using the system simultaneously and talking to one another about it. Or, if one-to-one is not an option and the participants are together, they are not allowed to communicate about their usage of the code search engines.

1. Ask participants to fill out the consent form in Appendix C.
2. Ask participants for demographic data: age, gender, job type and years of programming experience.
3. 1-2m explain the purpose of the study and show the first search engine.
4. Give the participant a list of queries and categories on which to base their queries on. And allow the users to call their queries and play with the search engine to their liking.
5. Show the second search engine and repeat the previous step with the same queries.
6. Gather qualitative user feedback on interacting with both of the systems. For each of the queries. In the intent modelling search engine, users can provide feedback within the system for each retrieved code snippet.

7. 5-10m Have the user fill in the TAM2 questionnaire and explain questions that the user does not understand.

8. Finish up by thanking the participant for their time and effort.

3.2.5 Data collection and analysis

Data will be collected by surveying the relevance of every item for the baseline compared to the intent-based bot. They will assess relevance based on a 7-point Likert scale distribution. The conversations will be stored in a MongoDB database and a local JSON file alongside this relevance evaluation. The structure of the MongoDB contains the conversation data, including the queries the user said, the proposed action based on the intent calculation, and the reply the chatbot gave based on the user's query. The following statistical measures will be used to measure the results from the experiment data.

Descriptive Analytics

Descriptive statistics are the numerical and graphical techniques used to organise, present and analyse data [37]. They are a relatively easy way to describe the results of the presented survey, such as the mean, median and mode of the answers given to the questions. Moreover, the frequency of feedback keywords can also be described to show users' general qualitative feedback while using both search engines.

Factor Analysis

Factor analysis is a technique used to reduce a large number of variables into fewer numbers of factors. In this study, factor analysis will primarily decrease the number of factors used in the TAM2 model only to include the variables with the maximum common variance. This allows for a more straightforward analysis of the results. Instead of analysing the results of all questions separately, it is possible to cluster the statements to the fundamental concepts of the TAM2 model and report on the relationships between them.

Chronbach's Alpha

To address the reliability of the study, which is the degree of probability one would obtain the same result if the study were carried out again under the same circumstances as it was done now, Cronbach's alpha was used. A measurement instrument is concerned with the extent to which an instrument - in this case, the experiment survey and TAM2 questionnaire - measures what it intends to measure [122]. Cronbach's alpha describes how much each measured item is correlated with every other item measured, therefore measuring the extent to which high responses go with highs and low responses go with lows across all items [69]. When combining this with factor analysis, it is possible to create a reliability measure for each of the different aspects of the TAM2 questionnaire and the reliability of the experiment survey for each question. A Cronbach's Alpha value of 0.8 is considered highly reliable, which is why that value will be used as a benchmark for measuring the reliability of this study for each of the instruments.

3.2.6 Technology Acceptance Model 2

The Technology Acceptance Model (TAM) is a widely used theory in the field of computer science research that was first proposed by Davis[26]. The original TAM proposed that users' acceptance of new technology is primarily determined by perceived usefulness (PU) and perceived ease of use(EU). Perceived usefulness refers to the degree to which a user believes that using the technology will improve their performance, while perceived ease of use refers to the degree to which a user believes that using the technology will be effortless [26]. The TAM has been extensively used and validated in various studies. Still, it has also been criticized for its limitations, such as its inability to account for social influence and contextual factors. In response to these criticisms, Venkatesh and Davis proposed an extension of the TAM called TAM2 [126], which incorporates additional factors that influence technology acceptance, such as social influence, cognitive instrumental processes, and affective instrumental processes. The TAM2 proposes that users' acceptance of technology is influenced by its perceived usefulness, ease of use, and external variables, such as social norms, subjective norms, image, and job relevance [126].

In this study, TAM2 was used to evaluate the acceptance of the intent-based search engine. Appendix B contains the Google form used to evaluate the intent modelling system of which the results will be used for the final evaluation. The following list shows the different categories of the TAM2 model along with their evaluation questions for the intent-modelled code search engine. For each question, there is a 7-point Likert scale for the participants to fill in with the following values regarding their degree of agreement or disagreement: extremely likely (1), quite likely (2), slightly likely 3), neither (4), slightly unlikely (5), quite unlikely (6), extremely unlikely (7).

Before the main TAM questions are asked, the TAM2 model also assesses the influence of two more processes, Social Influence and Cognitive instrumental [126]. To assess these processes in the current setting, several starting questions will be asked each participant before filling out the questionnaire. Assessing the voluntariness (V), experience(E), subjective norm(SN), image (I), job relevance (JR), output quality(OQ) and result in demonstrability(RD). The following lists show the questions for the TAM2 questionnaire.

1. Social influence
    - SN: To what extent do you feel compelled to respond to the following questions in a specific manner?
    - V: To what extent do you perceive the adoption of this system as voluntary?
    - I: How likely will you be judged by other coders for using a code search engine in this manner?
    - E: What is your experience in searching for code in search engines?
2. Cognitive instrumental
    - JR: Do you think the search engine is relevant for your job?
    - OQ: How well does the search engine perform the tasks that match your job relevance?
    - RD: Are you able to see and understand the results from using intent modelling in a search engine?

- **Perceived Ease of Use (EU)**
  - EU1: An intent-based code search engine would be easy to learn and use.
  - EU2: The search engine would not require extensive training to use effectively.
  - EU3: It would be easy to remember how to use the search engine.
  - EU4: Using the search engine would be effortless.
  - EU5: It would be easy to become skilful at using the intent-based search engine.
- **Perceived Usefulness (PU)**
  - PU1: An intent-based code search engine would improve the efficiency and effectiveness of code searches.
  - PU2: It would increase the accuracy of search results.
  - PU3: It would improve searching for alternatives when the code is unavailablez.
  - PU4: It would save time when searching for code.
  - PU5: Using the search engine would be valuable to completing coding tasks.
  - PU6: Using the search engine would make it easier to do my job.
  - PU7: I would find the search engine useful in my job.
- **Self-predicted future use/Intention to use (IU)**
  - IU1: I predict that I would regularly use an intent-based code search engine in the future.
  - IU2: I would prefer using the intent-based code search engine over a baseline code search engine for searching code.

The participants of the experiment will assess these questions after they have used the baseline code search engine as well as the intent-enhanced code search engine.

3.2.7 Experiment Summary

The experiment is conducted in practice by following the following steps.

1. Objective
   - The experiment objective is to evaluate the use of intent and entity modelling on software engineers' semantic code search experience. Furthermore, feedback from the software engineers is gathered to improve the software artifact to contribute to the broad scientific knowledge base.
2. Participants
   - The participants in the study have experience with using search engines and programming code as described in section 3.2.2.
   - The participants in the study will have to sign the consent statement presented in Appendix C in their preferred language. These statements explain to the participants the implications of participating in the study, as well as what steps are performed in the study.

- The participants will be asked about their age, gender, job type and years of experience in programming.

3. Design and procedure

- The experiment is a within-subjects design, each participant uses the baseline flask search engine and the intent modelling-based search engine.
- The order in which the participants use either the baseline or the intent first is randomized 50/50 to avoid potential order bias effects.
- The order in which the participants use either the baseline or the intent first is randomized 50/50 to avoid potential order bias effects.
- The participants receive a set of predetermined queries from CodeSearchNet [56] and CodexGlue [79] for benchmarking and evaluation purposes.
- For each of the queries asked the user will be asked to evaluate the performance of both search engines and give feedback on how performance may be improved.
- After using both search engines the participants are asked to fill in a 7-point Likert scale TAM2 questionnaire to evaluate the acceptance of the innovation in the code search field. This 7-point TAM2 questionnaire will first be explained, with all questions and metrics using the same text to decrease bias in reporting on the questionnaire.

4. Data Analysis

- Data is collected using the saved conversations in the MongoDB, in the local JSON data dump and retrieved from the answers to the TAM2 questionnaire and additional feedback.
- The 7-point Likert scale TAM2 results will be analysed using descriptive statistics per question, and factor analysis for each category of TAM2 questions. Furthermore, Cronbach's alpha, a measure of internal consistency reliability will be used to assess the extent to which the items in the scale are measuring the same construct [20].
- User feedback from the questionnaire and conversation data from MongoDB and local JSON will be used to improve the intent and entity classifiers. User annotations can also be used to improve the training process for the intent-based search engine.
- User feedback will be thematically analyzed to identify common themes and suggestions for improvement of the software artifact.

5. Conclusion

- Based on the analysed data it is possible to conclude the evaluation of the intent-based search engine. Showing the major perceived differences by adding intent modelling features to the search experience. Therefore concluding this study.
- Feedback analysis may show areas of future improvement and other ways the software artifact may iteratively improve itself in context.

Chapter 4

Results

In this chapter, we present the results of implementing the software artifact, highlighting the changes made and the impact of user feedback during the evaluation process. The chapter is structured into several parts, each focusing on the software artifact's development and validation. Firstly, we discuss the software artifact created based on the problem investigation conducted in Chapter 2, which was informed by a comprehensive literature study. Next, we delve into the impact and contributions of the software artifact, followed by an exploration of the validation process. Throughout the development of the artifact, we followed the MCDM framework guidelines, as outlined in Section 1.4.1, ensuring that the software artifact aligned with the requirements obtained from evaluations and the scientific knowledge base established through the SLR.

4.1 Design Science

The following section depicts the results of the implementation of the artifact, what changes were made and how the user feedback during the evaluation changed the artifact in context. The section is split into different parts, explaining the software artifact created from the problem investigation in the form of the literature study shown in Chapter 2, the impact and contributions, and the validation. Following the MCDM framework guidelines described in section 1.4.1, the software artifact was created based on the requirements acquired from the evaluations and the scientific knowledge base built in the SLR.

4.2 Software Artifact Development

The development of the software artifact prototype was primarily driven by the theoretical framework created during the SLR. In this section, we provide an overview of the software artifact prototype, emphasizing its development process and the theoretical framework derived from the SLR. We explain the rationale behind the choices made for different techniques and components, which were extensively discussed in Chapter 2. Specifically, we elaborate on the design choices related to qualitative decision-making methods discussed in Section 2.2.6 and quantitative decision-making methods explored in Section 2.2.5. By referring to the literature study, we demonstrate how the software artifact aligns with the findings and recommendations derived from previous research.

4.2.1 Overview of the Software Artifact Prototype

This subsection presents a comprehensive overview of the software artifact prototype, focusing on its main features, functionalities, and approach to addressing the identified problem. Additionally, we emphasize any unique or innovative aspects of the artifact that distinguish it from existing solutions.

The artifact was systematically developed through coding in Python, Vue, MongoDB, and the setup of ElasticSearch using Docker. We carefully considered the requirements gathered from evaluations and insights from the literature study conducted in Chapter 2 during the development process.

The software artifact prototype consists of several key components: indexing, querying, ranking, searching, and user intent modelling. Each component was meticulously designed to fulfil specific tasks and work harmoniously within the system. Moreover, the artifact incorporates modularity, allowing each component to be easily reused in different settings or implementations. Table 4.1 presents the concrete implementations of the theoretical framework described in the systematic literature review (SLR) for each code search engine created. The table highlights that both code search engines employ the same code indexing, retrieval, and ranking techniques. However, the intent-enhanced code search engine stands out by utilizing intent modelling techniques to enhance the overall code search experience.

TABLE 4.1: Comparison of Practices in Baseline and Intent-Enhanced Code Search Engines

Components	**Baseline Code Search Engine**	**Intent-Enhanced Code Search Engine**
Indexing	CodeBert ElasticSearch	CodeBert ElasticSearch
Querying	Text-based Input bar	Conversational Agent
Ranking	Multi-Feature Ranking	Multi-Feature Ranking (Intent Aware)
Searching	CodeBert	CodeBert
User Intent Modeling	None	Intent Classification and Multiple Choice Reinforcement learning

The baseline code search engine was created by using the Python Flask package. This framework combines a backend connection to the ElasticSearch indexing and a Google-like input query bar. The front end for the Flask application was written using pure HTML. Figure 4.1 shows the Baseline code search engine with a participants' query. Figure 4.2 shows the intent-enhanced code search engine and its recent search feature, multiple choice and entity recognition. These are some of the intent modelling features that can be seen from the application's front end. The actual weighing algorithm layer between RASA AI and ElasticSearch can be seen in the Python code on GitHub.

4.2.2 Impact and Contributions

The software artifact prototype created during this study establishes a basis for other researchers of code search to work from. While the initial idea was to apply these

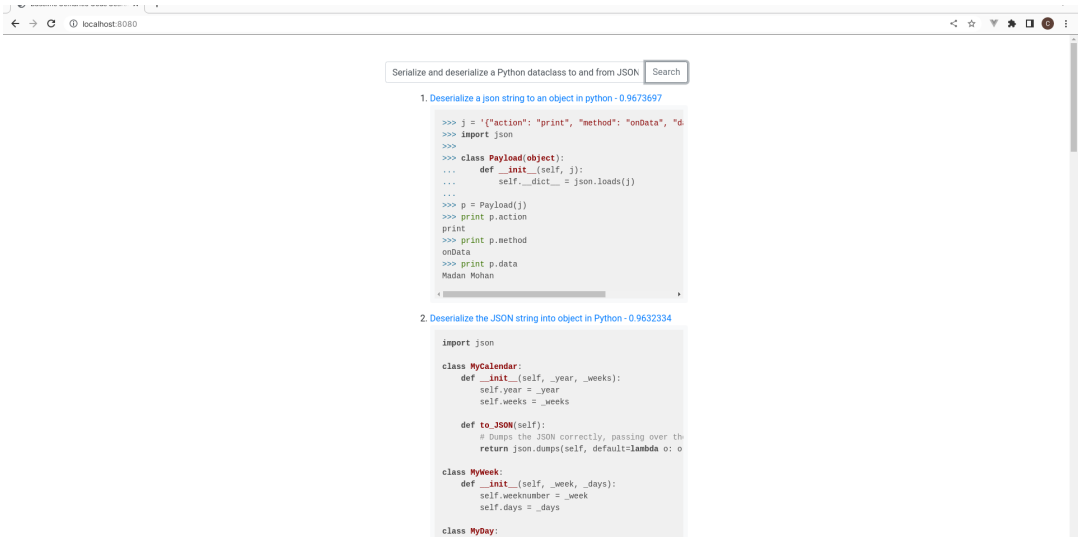


FIGURE 4.1: Example of the Baseline Code Search Engine with a participants' query

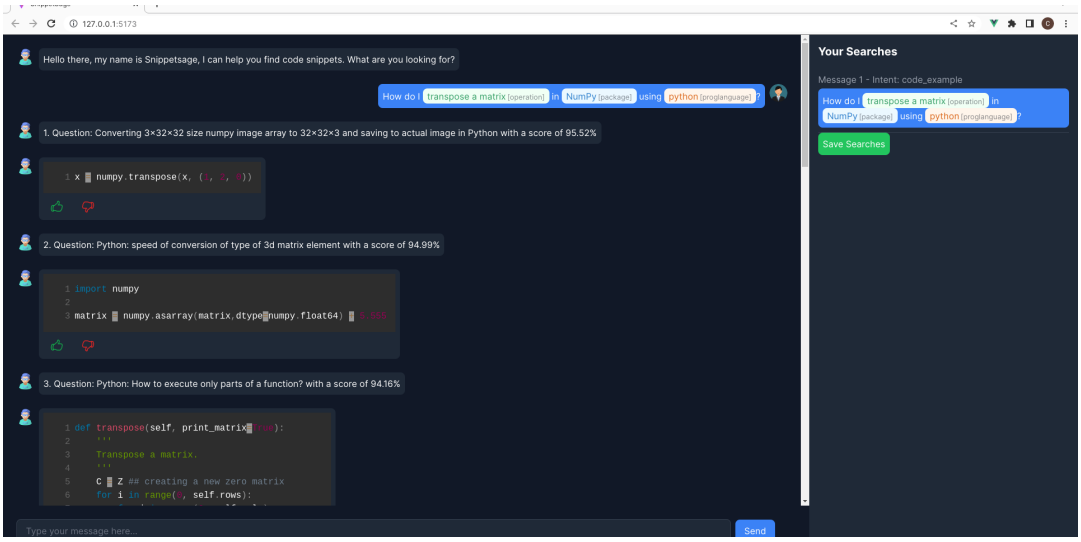


FIGURE 4.2: Example of the Intent-Enhanced Code Search Engine with a participants' query

new techniques to an existing open-source, easy-to-use code search engine, these were not found during writing, resulting in a long development process. This was done to ensure scientific reproducibility, so other machine learning models and intent modelling techniques could easily be applied. The following areas of impact have been considered while implementing the code search engine:

1. **Enhanced Code Search Capability:** The software artifact prototype that combines semantic code search methods with user intent modelling improves the code search capability beyond traditional keyword-based searches. The system provides more accurate and relevant search results by understanding the context and intent behind code queries. This enhancement saves developers time and effort, accelerating the software development process.

2. **Improved Code Reusability and Knowledge Sharing:** The integration of semantic code search and user intent modelling promotes better code reusability

and knowledge sharing. Developers can express their intentions or requirements naturally, allowing the system to retrieve relevant code snippets or modules. This fosters code reuse and collaboration and reduce redundant code development, contributing to shared code repositories and collective learning. While systems like ChatGPT generate code with no license or credit, with a search engine, developers can be rewarded for their contributions and adequately cited.

3. **Self Learning and improving per use case:** To manually annotate and train the results, users can leverage a powerful tool called Prodigy. Prodi.gy is a versatile annotation tool designed to facilitate the manual annotation process and assist in training machine learning models. It offers a user-friendly interface and various features that streamline the annotation workflow. In intent modelling, the users can review the classifications the intent-enhanced code search engine makes, thereby improving the algorithms.

4. **Advancement in Natural Language Processing (NLP) Techniques:** The development of this software artifact prototype pushes the boundaries of NLP techniques, particularly in code understanding and retrieval. Integrating semantic code search methods with user intent modelling requires sophisticated NLP algorithms that bridge the gap between human language and programming code. These advancements contribute to the broader field of NLP, expanding its applicability beyond traditional language processing tasks.

5. **Usability and User Experience Enhancements:** The prototype's easy-to-understand modular system ensures usability for developers of all experience levels. The user-friendly interface and user intent modelling reduce the cognitive burden and enhance the overall user experience. These aspects contribute to human-computer interaction research, providing insights into designing intuitive and efficient tools that effectively cater to user needs. Other methods of user-intent-based user experience could also be studied in the future.

The following sections will describe the evaluation and validation of the developed software artifact in the form of an experiment comparing the baseline with the intent-enhanced code search engine and providing a questionnaire on using the intent-enhanced code search engine.

4.3 Experiment

The experiment was conducted over two weeks, evaluating two search engines in a real-world environment with over 20 participants who all code, search for code and evaluate code weekly. The following sections describe the experiment's results and answer the final and main research questions.

4.3.1 Quantitative results

To understand the experiment results, several exploratory quantitative analyses have been done. The following figures show the demographic data of the participants. The goal during participant selection was to mitigate as much sample bias as possible by selecting a diverse range of participants in age, gender, job type, organization and years of coding experience. Besides the participant quantitative data, the answers were also analysed based on the code search engine used in the experiment. The following sections describe the participant demographics in more detail, along with

the comparison between the baseline and the intent-enhanced search engine in the cases where it did or did not provide the users with valuable answers to their questions.

Participant Demographics

This section shows and analyzes the different demographic of the participants to ensure the experiment is diverse. While the sample is relatively small, some generalizability can still be attained from the results. Figure 4.3 shows the reported gender of the participants, which shows most participants were men, but as there are only six countries in the world where women make up more than 30 per cent of tertiary graduates in information and communication technologies [27], the 25 per cent considered in this study seems reasonable for the computer science field.

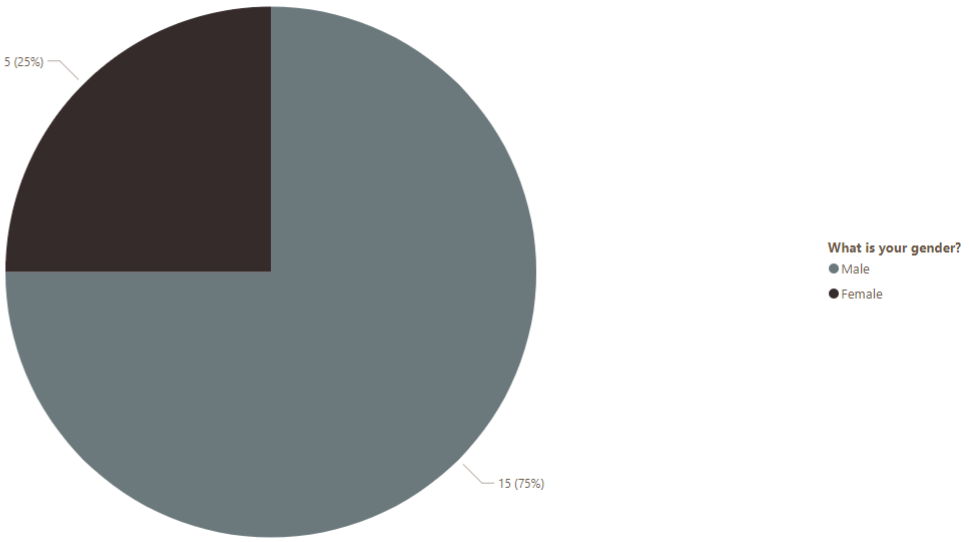


FIGURE 4.3: Participant reported gender

Figure 4.4 shows a big diversity in the job type of the participants, mainly consisting of backend and frontend developers. During the study, it was surprising to find how many people from different fields than pure backend development write code weekly for their jobs and how this affects the type of queries they generate.

Figure 4.5 shows that many users were not willing to give the organization or field at which they work for such an experiment, it also shows that there is a large diversity of different organizations within the participant group and more than just one organization was considered.

Figure 4.6 shows the overall coding experience of the participants. It shows that most participants have 1-6 years of coding experience, with some outliers of 6-10+ years, regarded as senior developers. This ensures that the study is not performed with just junior developers, possibly skewing the results, but that it is also validated with more senior participants.

Answer Comparison

For the quantitative analysis of the answers, the participants were asked to rank and evaluate each answers the search engine provided. Half of the participants started using the baseline code search engine, while the other half started with the intent-enhanced code search engine. Figure 4.7 compares the user evaluation of the code

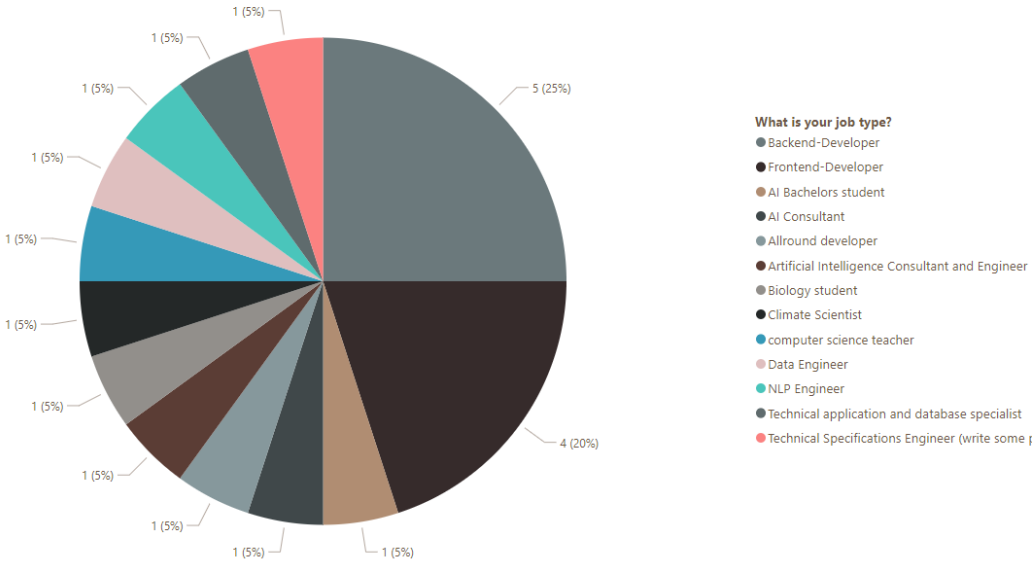


FIGURE 4.4: Participant reported job type

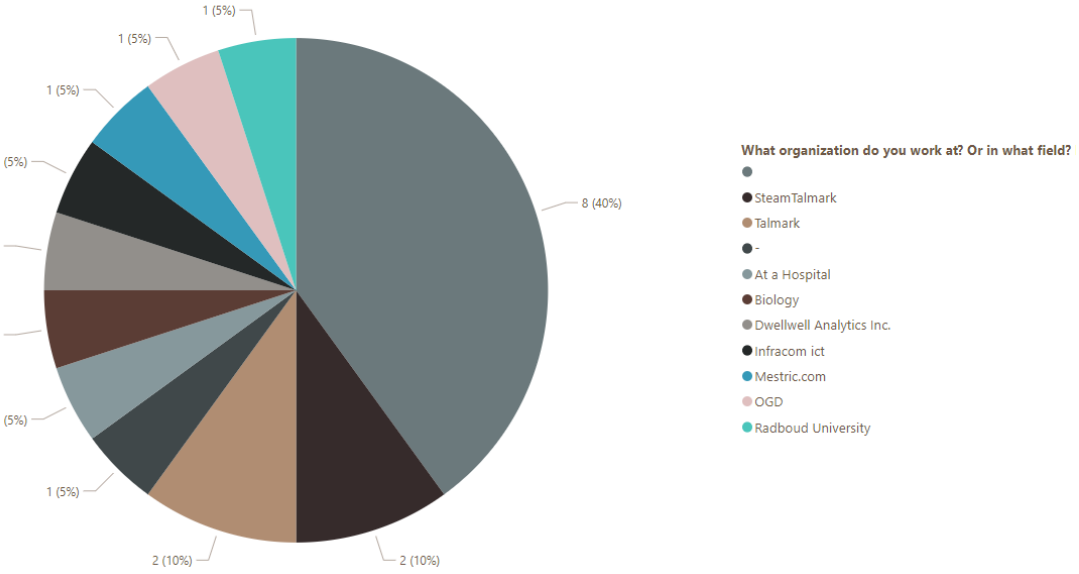


FIGURE 4.5: Participant reported organization or field of work

search engines whenever they provide a helpful answer. It shows that whenever the code search engines provided a beneficial result at what place in the ranking that answer was. On average, the intent-enhanced search engine ranked the helpful answers higher than the baseline and provided valuable answers to more participants. This can be seen by examining the bar charts; when all answers are answered, the bars should consist of 3 different colours, which is more frequent in the intent-enhanced code search engine. Furthermore, when the bar charts are lower, the answers were found earlier in the provided results, meaning a lower bar chart with different colours is considered a good result. In some cases, like with Participant 18, they found their answers better and quicker in the baseline version of the search engine, the reasons for which will be analysed in section 4.3.2.

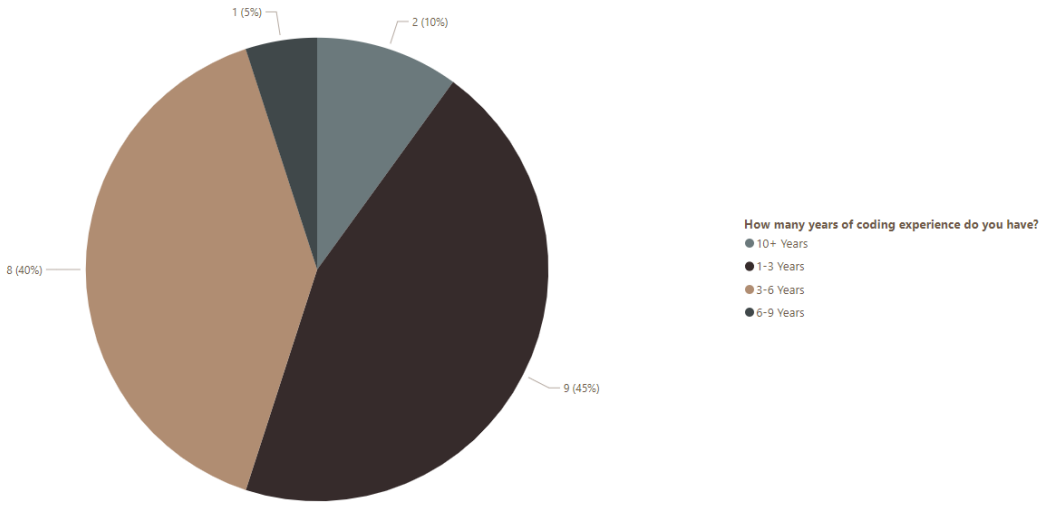


FIGURE 4.6: Participant reported years of experience

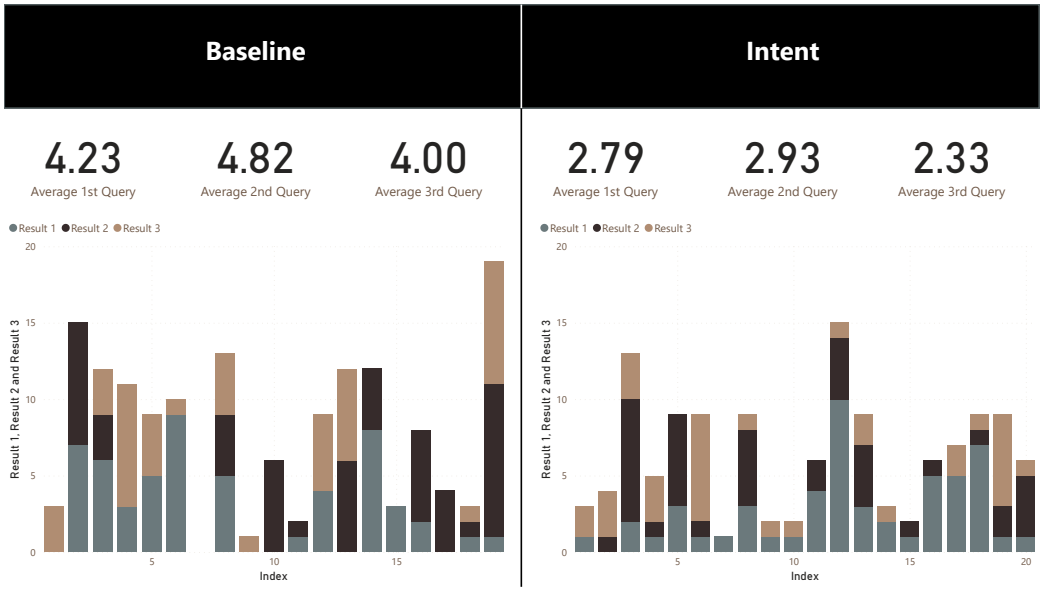


FIGURE 4.7: Comparison of correct answer Metrics between the Baseline Code Search Engine and the Intent-Enhanced Code Search Engine. Showing that the participants on average rated the intent-enhanced results as being correct or relevant earlier, while also retrieving valuable answers more frequently overall.

To determine whether the quantitative difference in the answers of both search engines is statistically significant for the different systems and queries, a (Multivariate) Analysis Of Variance ((M)ANOVA) along with a paired t-test has been performed. The (M)ANOVA was done to see the statistical significance of the questions and the difference in results based on the code search engine used. Table 4.2 shows the (M)ANOVA results for both the Code Search Engine used and the queries the users put in. Showing a significant difference in the code search engine used ($p = 0.00308$) and no significant difference in the queries used or in the interaction between the queries and the user code search engine. This test gives some insight into a statistical difference in the ranking for both search engines. However, this is based on only 20

participants, and 37 out of 120 results did not end up as a successful search, which is why they were not considered for this analysis. Moreover, the paired t-test between the systems for each query did not provide significant results, which could be due to the inconsistent differences between the systems across all queries. However, the overall variation between the systems is still significant, which (M)ANOVA accounts for. As (M)ANOVA has some assumptions on the dataset before use, like independence, normality and homogeneity, which were all considered during the experiment.

TABLE 4.2: (M)ANOVA Results

Factor	**Df**	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Code Search Engine	1	55.5	55.46	9.338	0.00308 **
Queries	2	6.1	3.07	0.517	0.59831
CSE:Queries	2	0.7	0.35	0.058	0.94349
Residuals	77	457.4	5.94		

No Answer Comparison

In 37 out of 120 queries fired at both systems, the search engine did not provide significant results to the participants. In these cases, the reasons for not providing significant results were compared for both types of code search engines. Figure 4.8 shows the number of times the baseline search engine could not retrieve any beneficial results compared to the intent-enhanced code search engine. It also shows the totals and differences based on which search engine was used first. The left-side chart shows the baseline results based on users who used the baseline first and the intent enhanced first, in which no significant differences can be found. The right-side chart shows the intent-enhanced results based on users who used the baseline and the intent-enhanced first. The users who used the baseline first and then switched to the intent-enhanced version were perhaps slightly easier on the performance of the intent-enhanced code search engine because they were used to the answers the baseline provided earlier.

Figure 4.9 compares the reasoning for users to reject the answers the code search engine provided. These reasons are explained in further detail in the qualitative analysis. Still, this quantitative analysis shows that, generally, when users rejected answers for the baseline search engine, they were more likely to blame this on the search engine itself. At the same time, with the intent-enhanced system, they were more likely to blame this on the knowledge base. Primarily when the users first used one code search engine and got the results they were looking for, whereafter they used the other search engine, they were more likely to blame it on the search engine performance, seeing that it should be in the knowledge base.

4.3.2 Qualitative results

Besides quantitatively analysing the experiment's results, qualitative analysis was performed to map the differences between the search engines. This qualitative analysis also leads to a broader understanding of the quantitative analysis and why participants evaluated results in a specific manner. Whenever a code search engine responded helpfully, the participant was asked to elaborate on why they thought these answers were helpful. Whenever the code search engine did not provide beneficial results, the participants were also asked why they thought the results were insufficient. Figure 4.9 shows the participants' answers to the close-ended questions regarding the code search engine's inability to provide an answer. Section 4.3.2 analyses the further

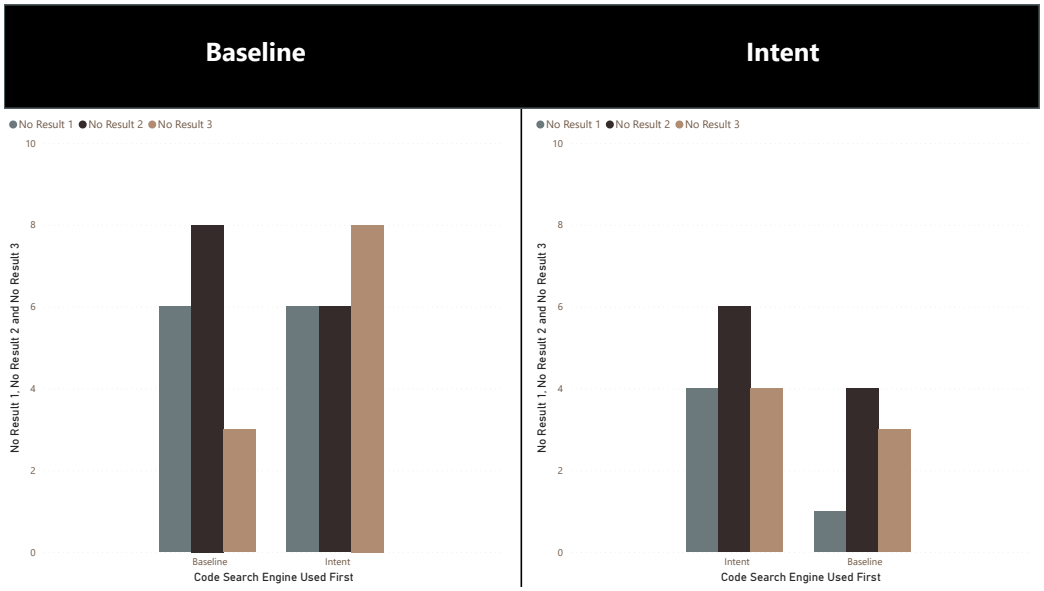


FIGURE 4.8: Comparison of no correct answer Metrics between the Baseline Code Search Engine and the Intent-Enhanced Code Search Engine

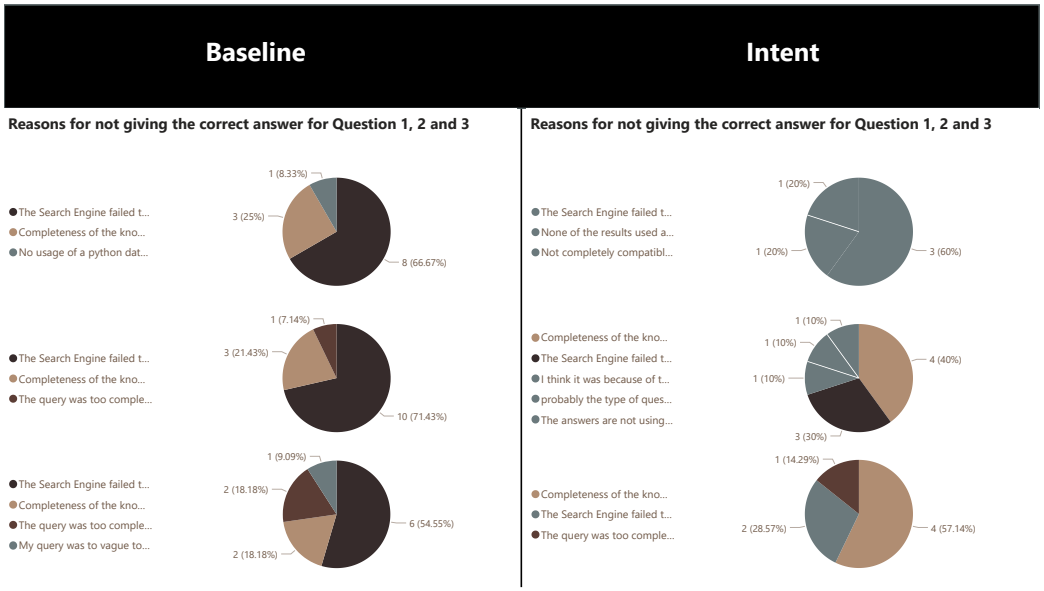


FIGURE 4.9: Comparison of no correct answer responses by the participants

reasoning of the users as to why they thought the answers were not helpful. Besides the actual answers, the questions or user stories that the participants chose could significantly impact their satisfaction with using the system. The full NVIVO [99] project, along with all of the codings, source data, and mappings, can be found in Appendix A.

Participant Queries

As described in section 3.2.3, each participant was allowed the creative freedom to design their queries so that the results would be helpful to them in a real-world scenario. The participants were guided in their query creation by offering several categories extracted from the knowledge base. NVIVO [99] was used to perform a coding and qualitative analysis of the queries generated by the participants. The following list shows the different codings applied to the user queries, which were manually applied to each user story based on their contents. The first four are taken from the categories defined in chapter 3, with the addition of Job-Related search and Random Search. The Job-Related search coding was assigned when a user story resembled some specific task the participant would do in their job. At the same time, random searches are classified as searches that do not match any job relevance.

- Category 1: Data Conversion
- Category 2: Data Manipulation
- Category 3: Data Storage and Retrieval
- Category 4: Network Communication and Security
- Job-Related Search
- Random Search

Figure 4.10 shows a generic overview word cloud of all the words used in querying the code search engines. Stop words like how, can, I, do etc. were left out of the visualisation.

Figure 4.10: Word frequency in user stories of the participants

Table 4.3 shows the NVIVO [99] codes and the number of references for each code. Some queries fit multiple codes, such as 'How can I connect a WordPress website with an external SQL database?', which fits Category 3 and 4 and is a Job-Related search query. SQL is a method of storing and retrieving data, but since the query mentions 'external', it is also considered part of network communication coding.

44 out of 60 queries were considered job-relevant; the following subsections will analyse the relationship between query type and the possible negative or positive result.

TABLE 4.3: NVIVO Codes and References

Code	**References**
Category 1: Data Conversion	9
Category 2: Data Manipulation	22
Category 3: Data Storage and Retrieval	28
Category 4: Network Communication and Security	18
Job Related Search	44
Random Search	4

NVIVO [99] allows for creating visualizations based on participant attributes, like gender, years of experience or frontend developer. However, with the relatively small sample size of each participant attribute, it could not find any relevant differences between these groups.

Answer Explanation

To analyse the results qualitatively, whenever the search engine provided an answer, relevant coding was created for every time the participant thought the answers were correct and elaborated on them. Thirty-two elaborations were given with an answer for the baseline code search engine, while 45 were given with the intent-enhanced code search engine. These elaborations were coded and classified as either positive or negative, as some say they are somewhat satisfied with the results but expected more. Some other participants commented negatively, although they found the result satisfactory enough: "It was the only answer with authentication, but not at all related to my question.". Table 4.4 shows the different codings, with a classification of them being successful or unsuccessful, along with some examples provided by the participants.

Code	**Classification**	**Description**
Data Extraction	Successful	Discussing the extraction of data from various sources
Solution Creation	Successful	Exploring the creation or implementation of a solution
Package/Framework Mentioned	Successful	Mentioning or referring to specific packages or frameworks
Lack of Relevance	Unsuccessful	Expressing the absence of relevant or helpful information
Partial Answer	Successful	Responses that partially address the question or provide incomplete information
Positive Feedback	Successful	Expressing positive sentiments or satisfaction with the provided results
Negative Feedback	Unsuccessful	Expressing dissatisfaction or disappointment with the search results

TABLE 4.4: Classification of Codes

Code	Baseline	Intent Enhanced
Data Extraction	1, 3, 5, 9, 32	
Solution Creation	2, 7, 10, 14, 30	9
Package/Framework Mentioned	4, 11, 16, 20, 23, 24	3, 22, 41
Lack of Relevance	12, 13, 15, 22, 26, 29	4, 11, 14, 15, 19, 21, 23, 26, 29, 31, 33, 34, 35, 39, 42
Partial Answer		2, 5, 8, 10, 12, 17, 22, 24, 25, 27, 28, 30, 32, 36, 37, 38, 40, 41, 43, 44, 45
Positive Feedback	1, 7, 13, 24, 26	1, 7, 13, 16, 18
Negative Feedback	4, 23	

TABLE 4.5: Coded Query Indices, actual queries can be found in Appendix A

Table 4.5 shows the different query indices and their categorization based on the codings. These queries were assigned to a specific coding for each code search engine. As seen in the table, the intent-enhanced search engine showed better results regarding partial answers or answers that were 'closely related' to the answer they sought. Many participants also indicated that while the answers were not perfect, they had the sense that they gave them some indication of where to look next or directions for query reformulation. The following queries taken from the intent-enhanced code search engine elaborations show this effect:

- *'They weren't exact matches but they both gave me a right indication where I could move on from, they both needed some extra details to be exactly what I need, but the basis was what I was looking for.'*
- *'This answer came closest to answering the main question. Same topic, different threshold'*
- *'the answer shows an example of coping a file from a networkpath to a local path but can be used to do the otherway around'*

Overall, when the query was concise and precise, the baseline code search engine outperformed the intent-enhanced code search engine. However, when a participant expanded their query or had a job-relevant vague query, the intent-enhanced code search engine seemed to support them in searching.

The following section describes the qualitative results for when the participants did not retrieve a helpful result from the code search engines.

No Answer Explanation

In some cases, the code search engine did not provide a helpful answer to the participants. In this case, the participant was asked to describe why they felt the code search engine did not retrieve any relevant results. Most cases are visualized for each code search engine in section 4.3.1. However, some participants also elaborated on their specific cases when the code search engine did not provide beneficial results. These cases were grouped and analyzed to see the most common problems when the code search engine did not retrieve any relevant results for the participant. The baseline code search engine had 19 cases, summarized below and compared to the 9 cases in

Category	**Baseline**	**Intent**
Fully Not Helpful	1,3,4,5,6,9,10,12,14,15,16,17,18	1,2,3,4,6,8
Partially Not Helpful	2,7,8,11,13,19	5,7,9

TABLE 4.6: Categorizations for why the code search engines did not provide beneficial results.

the Intent-Enhanced code search engine. When the participants classified the results as unsatisfactory, their queries were classified into two categories. The first category describes those cases where the participants say there is no relevant item. In contrast, the second category classifies all elaborations where the participant found one part of the answer satisfactory but overall was dissatisfied with the results. Table 4.6 shows these different categories and how many elaborations per code search engine were classified.

This shows that for each code search engine, about fifty per cent of the results marked as insufficient could have been marked as sufficient by any other participant with the same query, as this came down to personal experience and possible random variance of the participant with this exact query.

User Experience Feedback

During the experiment, all participants were also asked to give generic user experience feedback on the search engine they were using. This feedback was considered during the experiment and led to intermittent changes to the software artifact. Thereby functioning as field testing for the relevance cycle proposed by Hevner [49]. The baseline user experience feedback consisted primarily of participants finding the baseline engine too basic, there were no numbers to show the index of the questions, the page was too long, and they thought the boxes were not big enough to hold all of the code. Some participants also provided some general feedback on the results from the code search engine, saying the search engine returned very specific results while they were looking for general ones and giving general negative feedback on the code search engine. This feedback was considered by widening the general area of the code snippets and adding score indicators and numbers to the question for easy referencing in the survey. Overall, the UX feedback on the intent-enhanced code search engine was more positive. Some participants mentioned the index missing, the scores not adding value, and the entity recognition not adding value to the front end. Furthermore, multiple participants said the syntax highlighting of the code snippets sometimes made the code snippets very hard to read. This, along with the indexing and some bug fixes regarding entity recognition, was addressed halfway through the experiment.

Summary of the responses

This section presents a broad summary of the participants' responses when using both code search engines. All participants are anonymized and for traceability marked with a unique number. Some examples from the baseline code search engine's results include:

- Positive
  - P4: *This answer is not complete but pushed me towards the genshi package, which I am going to try out.*
  - P11: *Answer 1 seems to do the job*

  - P12: *this is what i want*
  - P14: *For me the important part was about grouping the arrays, this answer helps me with that part of the question.*

- Negative
  - P1: *The program language was not even considered. Besides that, there was nothing te be found about converting to a date type.*
  - P7: *None of the results above were able to answer my question. Some did contain some information about the panda library however none said anything about sorting.*
  - P10: *None of these results were on databases or creating new tables. Some showed some things about keys of dictionaries though.*
  - P20: *None of the answers mentioned anything about permissions or user groups*

And for the intent-enhanced code search engine:

- Positive
  - P4: *The first answer shows how to load a json using simplejson and then render an html template with the data.*
  - P5: *the answer shows an example of coping a file from a networkpath to a local path but can be used to do the otherway around*
  - P6: *I was looking for the code to solve my issue, and this query give me multiple variants for python, which is completely acceptable cause the framework wasn't specified*
  - P9: *This search engine gave me a lot of code on tuples, which is a higher level triplet.*

- Negative
  - P1: *None of the results answered my question. I could got some information out of it regarding how to connect to a SQL database.*
  - P7: *None of the answers contain the pytorch library hence all are not satisfactory.*
  - P11: *Well, it's maybe the best answer, though I expected a trunc(date) - trunc(current date) in number of days as the answer.*
  - P19: *The search engine didn't seem to take bubblesort into account, therefore all of the returned options are not particularly helpful. However, the first option at least suggests splitting up the data, which resembles a sorting algorithm somewhat.*

4.3.3 TAM2 Analysis

To analyse the TAM2 results, several different steps have been performed. Starting with data preparation to ensure the data is clean and ready for analysis. Then some descriptive statistics were created to give a broad understanding of the results. Lastly, both reliability and factors were analysed to see if the results of the TAM [26] match those of this study's analysis.

Data Preparation

For the analysis to be reliable, the raw questionnaire data first had to be cleaned. The primary dataset was split into four parts, one for each primary identifier of the TAM2 model; Subjective Norm, Ease of Use, Perceived Usefulness and Intention to Use. Then, the actual questions were stripped to ensure a clean table for analysis in R. The SN, V and I questions were reversed as their scaling was considered positive towards seven instead of negative like the other questions. Then, the results were exported into .csv format and imported into R for the statistical analyses of the following sections.

Descriptive Statistics

The first analysis phase comes in the form of basic descriptive statistics on the results of the different questions. One can achieve a basic image of the results and their meaning in the study context by describing them. Figure 4.11 shows the different responses the participants gave to each of the TAM2 questions. Overall the scores are pretty positive, with a few outliers that will be explored later. Table 4.7 shows the mean, median, mode, standard deviation (SD) and total range for each question. It shows that the ease of use was generally perceived to be better than the perceived usefulness, especially regarding job relevance and effectiveness. Many participants could not see and understand the use of intent modelling in the intent-enhanced code search engine, depicted by the extensive range in the RD question. And while some participants thought they would find the code search engine relevant to their job, the range of it and their experience in using code search engines was also extensive. Overall, the participants would prefer using the Intent-Enhanced code search engine over the baseline they used, regardless of which code search engine they used first during the experiment. Not all participants would actually regularly use the code search engine in the future. As we later analyse during the factor analysis, these two questions are part of later factors, which explains the difference in the type of question.

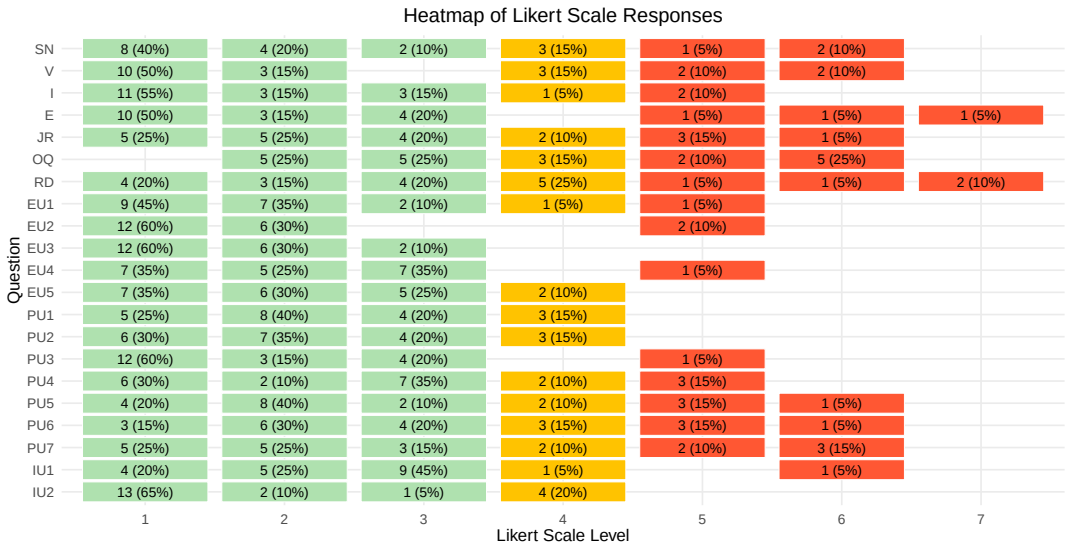


FIGURE 4.11: Heatmap of Likert Scale responses of the TAM2 Questionnaire

TABLE 4.7: Descriptive Statistics TAM2

Question	Mean	Median	Mode	SD	Range
SN	2.55	2	1	1.74	5
V	2.5	1.5	1	1.88	5
I	2	1	1	1.38	4
E	2.3	1.5	1	1.81	6
JR	2.8	2.5	1	1.58	5
OQ	3.85	3.5	3	1.57	4
RD	3.35	3	4	1.88	6
EU1	1.9	2	1	1.12	4
EU2	1.7	1	1	1.22	4
EU3	1.5	1	1	0.69	2
EU4	2.15	2	1	1.09	4
EU5	2.1	2	1	1.03	3
PU1	2.25	2	2	1.03	3
PU2	2.2	2	2	1.06	3
PU3	1.75	1	1	1.12	4
PU4	2.7	3	3	1.42	4
PU5	2.75	2	2	1.56	5
PU6	3	3	2	1.49	5
PU7	3	2.5	1	1.81	5
IU1	2.55	3	3	1.20	5
IU2	1.8	1	1	1.24	3

To see which questions correlated with one another, a correlation matrix was created using R. Figure 4.12 shows the correlation between the answers for each of the questions proposed in the TAM2 questionnaire. It shows a strong correlation between the first two Ease of Use questions EU1, and EU2 and between those and PU3. It also shows a strong correlation between the two PU questions on job relevancy, PU6 and PU7. These strong correlations are later represented together in the factor loadings for the factor analysis. Remarkably, the participant's experience and the other theoretical extension [126] TAM2 questions do not have any strong correlations with the later questions. This also explains the low reliability of factor analyses compared to the base TAM. Exceptions to this rule are the JR and OQ, which shows a relationship in the job relevance questions, and RD questions, which show a mildly strong correlation between recognizing the intent modelling and EU5, and PU3. Showing that participants who recognized the intent modelling felt that it would be easy to become skilful at using the search engine, as well as making it easier to improve searching for alternatives when the code is not available.

Reliability Analysis

To ensure the questionnaire has a high level of reliability, Cronbach's Alpha, the most widely used objective measure of reliability [122], was used. It was developed by Lee Cronbach in 1951 to measure the internal consistency of a test or scale and is expressed as a number between 0 and 1 [122]. All different measures of the TAM2 [126] were analysed for internal consistency and reliability. Generally, a value of 0.8 or higher indicates a reliable action [20]. Table 4.8 shows the Cronbach's Alpha results for the total questionnaire and the internal reliability of all individual items. It also shows

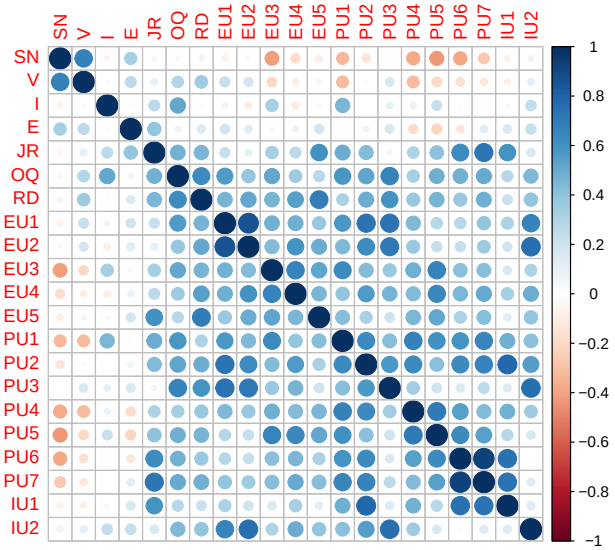


FIGURE 4.12: Correlation Matrix of all questions used in the questionnaire

the reliability of the questionnaire for each item dropped. Overall Cronbach's Alpha value is 0.89, indicating that the questionnaire has good internal consistency and reliability. All items with a minus sign after their measure name display a negative correlation with the other variables, which is logical because of the reverse scale of these items.

Factor Analysis

Factor analysis is a method used to identify underlying factors of the questions asked in the questionnaire. TAM2 inherently hypothesises that Ease of Use and Perceived Usability are factors and that they correlate with the eventual self-predicted future usage. This seems like a logical conclusion. However, the factor analysis is an exploratory technique and may provide different results from the hypothesis. This may lead to discoveries beyond this questionnaire's initial scope of TAM.

R was used to perform factor analysis in different ways. Using the Psych package in R, choosing the number of factors and the rotation function to calculate the different factor loadings is possible. To give a broad exploratory view of the data, it was analysed using multiple rotation functions, including the widely used varimax function and the Promax function, to see if correlations between discovered factors impact the overall analysis. Table 4.9 show the Factor Analysis results based on the number of factors determined with Kaiser's criterion. Kaiser's criterion suggests retaining factors with eigenvalues greater than 1, which is why an analysis with ten factors was done to see which factors hold to this criterion. Figure 4.13 shows the eigenvalues for each of the factors determined from the factor analysis. It shows that 6 is the maximum amount of factors with an eigenvalue of 1 or higher, which is why 6 factors were considered relevant for the final full analysis.

Usually, with the TAM2, the underlying concepts explored by the factor analysis are expected to match the TAM2 concepts [126]. However, the factor analysis for this questionnaire showed more possible factors underlying the data. This is probably through the addition of several questions on job relevance, which seem to fit a different aspect than the previously intended factor from the TAM2 model. Furthermore, most concepts introduced in the theoretical extension were only used by a single question

TABLE 4.8: Cronbach's Alpha Analysis Results

Measure	**Raw Alpha**	**Std. Alpha**	**G6(smc)**	**Avg_r**
TAM2	0.89	0.91	0.99	0.33
95% Confidence Boundaries				
Feldt	0.81	0.89	0.95	
Duhachek	0.82	0.89	0.96	
Reliability if an Item is Dropped				
SN-	0.89	0.91	1.00	0.35
V-	0.90	0.92	1.00	0.36
I	0.90	0.92	0.99	0.35
E	0.90	0.92	0.99	0.36
JR	0.88	0.91	0.99	0.33
OQ	0.88	0.90	0.99	0.32
RD	0.88	0.91	0.99	0.32
EU1	0.88	0.90	0.99	0.32
EU2	0.89	0.91	0.99	0.32
EU3	0.89	0.90	0.99	0.32
EU4	0.88	0.90	0.99	0.32
EU5	0.89	0.91	0.99	0.33
PU1	0.88	0.90	0.99	0.31
PU2	0.88	0.90	0.99	0.31
PU3	0.89	0.91	0.99	0.33
PU4	0.88	0.90	0.99	0.32
PU5	0.88	0.90	0.99	0.32
PU6	0.88	0.90	0.99	0.32
PU7	0.88	0.90	0.99	0.32
IU1	0.89	0.91	0.99	0.33
IU2	0.89	0.91	0.99	0.33

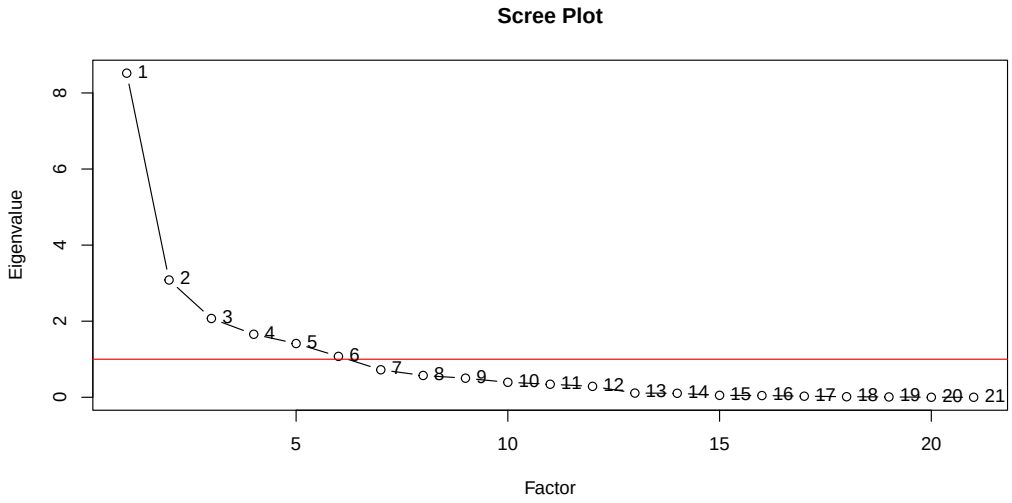


FIGURE 4.13: Scree Plot showing eigenvalues for each factor in descending order, for all questions

for each idea; this could hinder the factor analysis and explain the negative Tucker-Lewis Index (TLI) of -0.851 and high Root Mean Square Error of Approximation (RMSEA) of 0.351.

Another dataset, consisting of only the Ease of Use, Perceived Usefulness and Intention to Use, was used to compare the initial results to validate whether the first seven questions negatively impacted the factor analysis reliability. This shows a different picture for the analysis as the TLI of 0.854 for factoring reliability went from negative to positive, offering drastically increased reliability for the original TAM [26] questions. The RMSEA also shrank to 0.078, which describes a better fit. Figure 4.14 shows the Scree Plot generated from the factor analysis on the original TAM data, depicting only three factors to have eigenvalues more significant than one. Therefore Table 4.10 shows the factor analysis results using only three factors with the reduced dataset. The following list shows the different extracted factors and their questions, which, as shown in the table, do not entirely match the TAM components. This could be due to the small sample size or questions not fitting the components well enough.

- PA1 - Accurate, Ease of Use and Job Relevance -> Intention to use
  - PU2: It would increase the accuracy of search results.
  - PU6: Using the search engine would make it easier to do my job.
  - PU7: I would find the search engine useful in my job.
  - IU1: I predict that I will regularly use an intent-based code search engine in the future.
- PA2 - Easy to learn, alternative searching -> Preference over alternative
  - EU1: An intent-based code search engine would be easy to learn and use.
  - EU2: The search engine would not require extensive training to use effectively.
  - PU3: It would improve searching for alternatives when the code is unavailable.

TABLE 4.9: Factor Analysis Results TAM2

Item	PA1	PA3	PA5	PA2	PA4	PA6
SN	0.03	-0.16	-0.17	0.66	-0.08	0.26
V	0.13	-0.06	0.05	0.88	0.04	0.05
I	0.02	-0.05	0.00	-0.08	0.91	0.09
E	0.14	0.07	0.01	0.30	0.02	0.55
JR	-0.08	0.65	0.46	0.11	0.31	0.50
OQ	0.44	0.36	0.30	0.29	0.64	-0.24
RD	0.41	0.24	0.69	0.35	0.03	-0.06
EU1	0.83	0.24	0.17	0.04	0.09	0.05
EU2	0.87	0.09	0.27	0.03	-0.12	0.09
EU3	0.36	0.16	0.57	-0.33	0.30	-0.10
EU4	0.49	0.27	0.51	-0.15	-0.11	-0.07
EU5	0.23	0.11	0.82	-0.05	0.00	0.34
PU1	0.42	0.47	0.25	-0.41	0.46	0.08
PU2	0.65	0.68	0.11	-0.07	0.00	0.00
PU3	0.84	0.08	0.12	0.14	0.16	-0.11
PU4	0.37	0.40	0.34	-0.42	0.08	-0.12
PU5	0.13	0.35	0.66	-0.33	0.22	-0.29
PU6	0.05	0.88	0.32	-0.17	0.06	-0.21
PU7	0.16	0.83	0.35	-0.09	0.05	0.05
IU1	0.17	0.91	-0.07	-0.07	-0.04	0.17
IU2	0.83	-0.02	0.12	0.00	0.17	0.24

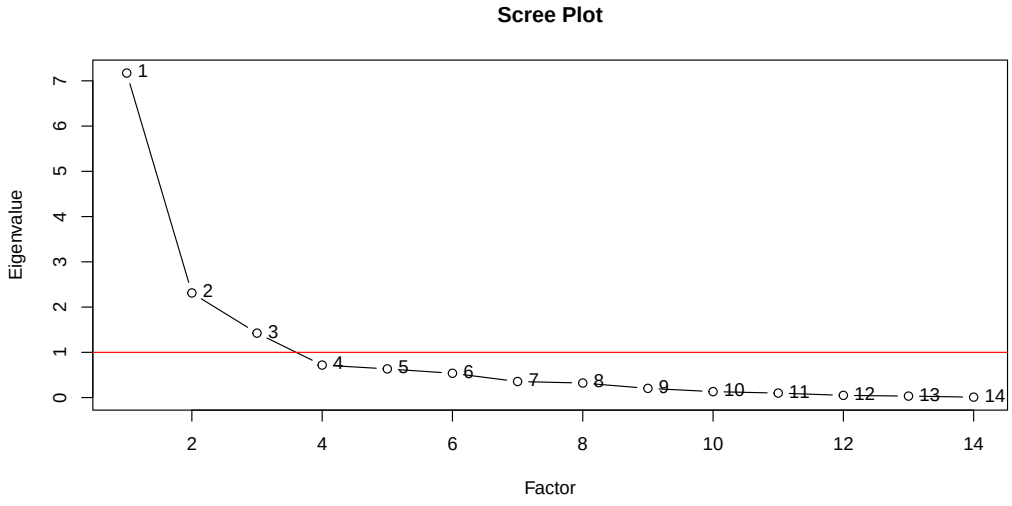


FIGURE 4.14: Scree Plot showing eigenvalues for each factor in descending order, original TAM

  - IU2: I would prefer using the intent-based code search engine over a baseline code search engine for searching code.

- PA3 - Learning and usage curve, time improvement and efficiency -> Actual coding task completion
  - EU3: It would be easy to remember how to use the search engine.
  - EU4: Using the search engine would be effortless.

TABLE 4.10: Factor Analysis Results Original TAM

Item	PA1	PA2	PA3
EU1	0.25	**0.82**	0.24
EU2	0.12	**0.86**	0.27
EU3	0.13	0.29	**0.75**
EU4	0.24	0.42	**0.58**
EU5	0.10	0.28	**0.58**
PU1	0.47	0.34	**0.53**
PU2	**0.71**	0.59	0.22
PU3	0.10	**0.80**	0.16
PU4	0.41	0.26	**0.55**
PU5	0.29	0.01	**0.90**
PU6	**0.83**	-0.03	0.45
PU7	**0.78**	0.12	0.41
IU1	**0.93**	0.12	0.02
IU2	0.01	**0.85**	0.17

- EU5: It would be easy to become skilful at using the intent-based search engine.
- PU1: An intent-based code search engine would improve the efficiency and effectiveness of code searches.
- PU4: It would save time when searching for code.
- PU5: Using the search engine would be valuable to completing coding tasks.

The above-determined factors show a relationship between the questions that are part of the factor. In general, these factors indicate a grouping between their items. For example, when participants find the Intent-Enhanced code search engine accurate and valuable in their job, their overall intention to use the search engine, regardless of alternatives, would increase. Likewise, when they found the code search engine easy to learn, and they provided them with suitable options compared to the baseline, they preferred it over the baseline. IU2 had a very low loading with the first determined factor, depicting a more subordinate relationship with the questions from that factor. The third and final factor groups the questions by efficiency and completing specific code tasks together.

Figure 4.15 summarises all statistical analyses performed with the TAM2 results.

4.3.4 Limitations

Factor analysis improves its quality when more input data is available. In this case, the total number of questions equals the number of participants, so a complete variable analysis could not be performed. When the same experiment was served with a sample size of at least 50, the factor analysis may have shown the loadings more closely to the actual TAM components. Furthermore, the reliability measures of the factor analysis were considered flawed when all questions were analyzed. This is why the regular TAM model was chosen for the final analysis. Some bias may also exist in the general use of the code search engines, whenever participants know someone is watching them and possibly judging their behaviour in using software, they make different decisions and may evaluate their use differently. Nonetheless, the answers the participants gave

		Descriptive Statistics			Factor Analysis			Summary of Responses						
								1	2	3	4	5	6	7
		Mean	Standard Deviation	Cronbach's Alpha	PA1	PA2	PA3	Slightly likely	Quite likely	Extremely likely	Neither	Slightly unlikely	Quite unlikely	Extremely unlikely
Easy to Learn	EU1	1.9	1.12	0.88	0.25	**0.82**	0.24	45%	35%	10%	5%	5%	0%	0%
Require Training	EU2	1.7	1.22	0.89	0.12	**0.86**	0.27	60%	30%	10%	0%	0%	0%	0%
Easy to Remember	EU3	1.5	0.69	0.89	0.13	0.29	**0.75**	60%	30%	10%	0%	0%	0%	0%
Effortless	EU4	2.15	1.09	0.88	0.24	0.42	**0.58**	35%	25%	35%	5%	0%	0%	0%
Easy to become Skillful	EU5	2.1	1.03	0.89	0.1	0.28	**0.58**	35%	30%	25%	10%	0%	0%	0%
Efficiency	PU1	2.25	1.03	0.88	0.47	0.34	**0.53**	25%	40%	20%	15%	0%	0%	0%
Accuracy	PU2	2.2	1.06	0.88	**0.71**	0.59	0.22	30%	35%	20%	15%	0%	0%	0%
Searching for Alternatives	PU3	1.75	1.12	0.89	0.1	**0.8**	0.16	60%	15%	20%	5%	0%	0%	0%
Save Time	PU4	2.7	1.42	0.88	0.41	0.26	**0.55**	30%	10%	35%	10%	15%	0%	0%
Complete Tasks	PU5	2.75	1.56	0.88	0.29	0.01	**0.9**	20%	40%	10%	10%	15%	5%	0%
Easier job	PU6	3	1.49	0.88	**0.83**	-0.03	0.45	15%	30%	20%	15%	15%	5%	0%
Job relevance	PU7	3	1.81	0.88	**0.78**	0.12	0.41	25%	25%	15%	10%	10%	15%	0%
Use Search Engine	IU1	2.55	1.2	0.89	**0.93**	0.12	0.02	20%	25%	45%	5%	5%	0%	0%
Prefer Search Engine	IU2	1.8	1.24	0.89	0.01	**0.85**	0.17	65%	10%	5%	20%	0%	0%	0%

FIGURE 4.15: A summary table showing the descriptive statistics, factor analysis and a summary of all TAM2 responses

during the evaluation seem genuine and the participants were not holding back on any criticism if it was due.

4.3.5 RQ4 *How can the code search engine be evaluated?*

Two different systems were considered to evaluate the improvements of an intent-enhanced code search engine compared to a baseline copy. This evaluation was done with 20 participants with experience in coding and searching for code. The assessment consisted of an experiment where the results of both search engines were evaluated, along with a TAM2 questionnaire to assess the acceptance of the intent-enhanced code search engine. This shows multiple different ways in which one may evaluate a code search engine and its possible enhancements. Therefore answering the fourth and last research question: How can the code search engine be evaluated?. Section 4.3.1 and the team 4.3.2 show different quantitative and qualitative analysis methods that have and can be applied to the experiment and TAM2 results. It offered a significant difference between the baseline and intent-enhanced code search engines and the overall technology acceptance of the intent-enhanced code search engine based on the questionnaire. Showing three different factors that diverted from the usual TAM, but showed underlying relationships between the answers to the questions. It showed that different people use and expect different types of code search experiences, along with areas of improvement.

Chapter 5

Discussion

This chapter comprehensively evaluates the study's implications in the scientific field of semantic code search. It critically examines the methodology and validity of the results, considering potential threats to validity. Additionally, the limitations of the study are discussed. Subsequently, the practical implications of the findings are explored, and possible directions for future research are suggested. Furthermore, the chapter addresses the practical developments and impacts of the study, considering the current global attention to ChatGPT and Large Language Models (LLMs).

5.1 Threats to Validity

Threats to validity are significant considerations in any research study. In this section, we assess the threats to construct, internal, external, and conclusion validity that may impact the findings of this study.

5.1.1 Construct Validity

Construct validity pertains to selecting appropriate operational measures for the concepts under study in the Systematic Literature Review (SLR). Several threats to construct validity need to be addressed. Firstly, choosing databases and venues could introduce limitations if they do not comprehensively cover the relevant literature. To mitigate this, Chapter 3 explicitly specifies the databases and outlets utilized, ensuring a broad scope of literature is considered. Additionally, the inclusion and exclusion criteria for each phase of the SLR are clearly defined in the respective subsections, minimizing the risk of inappropriate selections. Finally, expert evaluation, as mentioned in [156], is crucial to validate the expertise of the surveyed software engineers. The TAM2 questionnaire, including skill-related questions, aids in ensuring the validity of the selected software engineers for the survey.

5.1.2 Internal Validity

Internal validity threats primarily pertain to the user evaluation conducted in this study. It is crucial to address sample size considerations to ensure the validity of the results. Surveying a small number of software engineers from the same company, who share similar gender and age characteristics, may introduce a significant sample bias. To mitigate this, a well-spread and representative sample size should be obtained during the user experience survey. It is essential to avoid cultural bias by including participants from diverse organizations. For example, software engineers working primarily with C++ may not perceive Python code snippets as applicable. Additionally, efforts should be made to minimize potential subjective quality assessments during user analysis.

5.1.3 External Validity

External validity concerns the generalizability of the study's findings. Given the specificity of the research domain, which focuses on the state of the art in code search, the generalizability of the results is limited to a certain extent. Mapping a large set of literature to extract key components aims to generalize the findings of each study to their central concepts. However, there is a risk of overgeneralizing specific techniques or implementation details that may not be universally applicable.

5.1.4 Conclusion Validity

Conclusion validity focuses on the accuracy and consistency of the study's conclusions. To ensure reproducibility, each step of the SLR is thoroughly described to enable other researchers to achieve the same results. The software repository used in the study will be made publicly available, facilitating the replication of the TAM2 survey with different samples or modifying the system through repository forking. All steps of the SLR are stored in Mendeley data for easy referencing and result validation. While there is a potential for bias in study selection, this has been mitigated through peer reviewing with fellow students in the SearchSeco research group and during the Master of Business Informatics Colloquium course.

5.1.5 Study Limitations

Despite the efforts to maintain scientific validity, potential biases can still influence the study's results. For example, the experiments were conducted by participants within the writer's network, which introduces the possibility of sample bias. Most participants were selected from second-line networks rather than direct colleagues to mitigate this. Furthermore, diverse participants from different ages, backgrounds, and genders were included to ensure sample diversity.

5.1.6 Ethical Considerations

In addition to the validity threats and limitations discussed above, it is essential to consider the ethical implications of the design science approach and the experiment conducted. The utilization of ChatGPT and Large Language Models (LLMs) has garnered global attention due to their potential impact on society. Therefore, ethical concerns regarding privacy, data security, biases, and algorithmic fairness should be carefully addressed in the design and deployment of such systems. Furthermore, clear guidelines and policies should be established to ensure these technologies' responsible and ethical use, considering the potential risks and consequences they may introduce.

5.1.7 TAM2 Questionnaire

To enhance the scientific rigour of the study, the Technology Acceptance Model 2 (TAM2) questionnaire was employed to assess users' acceptance and perceptions of the artifact. This questionnaire provides valuable insights into users' attitudes, perceived usefulness, ease of use, and other factors contributing to adopting the semantic code search system. Furthermore, by considering users' perspectives, the study gains a deeper understanding of the practical implications and potential impact of the artifact in real-world scenarios.

5.1.8 Ecosystem/organizational indexing

The software artifact provides classes to index any unstructured data (code/docstring pairs) into a database using any pre-trained model. These indexing methods could be applied to different datasets than the ones considered in this work: CosQA [54] and StaQC [143]. At a bigger scale, the entire CodeSearchNet codebase could be indexed, along with private or organizational repositories, allowing the search engine to perform on organization codebases. The whole process can be hosted on-premises, ensuring the organization's hold of its data and understanding how the model works to fine-tune and improve it.

5.1.9 Improvements in ML models

Like presented in the CodeSearchNet [56] challenge, multiple opportunities exist to evaluate, benchmark and improve the existing machine-learning models in this space. Aside from improvements in precision, recall, accuracy and F1 score, the software artifact presented in this work also allows user testing using different models by configuring the models used for indexing and embedding the user input. One could also implement large language models like GPT-3.5 [14] and onwards to see the results of applying LLMs in a semantic code search environment.

5.1.10 Intent modelling improvements

Another area for future research is combining (more) advanced user intent modelling methods to the semantic search space. This thesis primarily focussed on basic intent modelling and providing a baseline for measuring improvements in this field. More advanced intent modelling techniques may be applied and evaluated using the system. By training Rasa's DIET classifier with a human-in-the-loop approach using Reinforcement Learning with Human Feedback (RLHF) [158], the semantic search engine could use reinforcement learning to improve results based on human feedback.

5.2 Practical Implications and Developments

The past year has seen many extensive NLP and AI developments. Generative AI in the form of advanced Generative Pre-training Transformer (GPT) using Large Language Models (LLMs) have taken the world by storm because of their capabilities. Furthermore, extensive machine learning capabilities previously reserved for researchers with the significant computing power available have become more accessible through large corporations like Amazon, Microsoft and Google, providing consumers access to these hardware components. These developments can drastically change the field of search engines since instead of searching for items, one could regenerate them each time. This sparks an ethical discussion on power usage, data privacy and closed-sourcing solutions. This study aims to steer away from this by improving the searching algorithms used on smaller-scale datasets, open-sourcing the workings of the algorithms and prototype, as well as allowing the possibility of hosting everything on-premise to keep data privacy at the enterprise level.

Chapter 6

Conclusion

This chapter finalises the work that has been done during this master's thesis and shows the overall conclusion of the study. The research questions will be answered, and all research objectives and goals will be assessed.

6.1 What are the essential components for a code search engine?

From an extensive systematic literature study, this thesis concludes that four critical categories of components to a (code) search engine exist. Being an Indexing component, a querying component, a ranking component and a searching component. We found that these components are strongly related, and while some methods may be modular, they often depend on implementing another component.

6.1.1 Indexing

This component ensures the input data is explicitly indexed so it is queryable, rankable and searchable. This thesis describes several methods of achieving this in chapter 2.

6.1.2 Querying

This component ensures the end user has an interface to interact with the search engine. A user may query and retrieve indexed code by typing a textual query, code query, or other means. Chapter 2 depicts different possibilities of querying a code search engine and their respective attributes.

6.1.3 Searching

Once the code snippets are indexed by the indexing component and a user wants to search using the querying component, the searching component searches the index based on the user input. Many ways to handle modern vector search are described in detail in the SLR analysis of chapter 2.

6.1.4 Ranking

A ranking component handles the ranking of the indexed code snippets based on the similarity between them and what a user is looking for. Several ranking algorithms are based on the indexing, searching and querying components. For example, older ranking algorithms often use frequency analysis of words in a document, while newer algorithms focus on the semantics of the words used rather than word occurrence.

6.2 How can software developers' search intent be modelled to support them with the code search process?

Several different intent modelling techniques and their appliance in existing literature were analysed to answer this second research question. In addition, other intent modelling techniques - primarily based on machine learning classifiers - were evaluated in creating the software artifact. The final software artifact used the user interaction with the system to define different search intents and will improve based on user behaviour. As the user intent is personal, users could either like or dislike code snippets presented by the search algorithm, creating a personalised search experience for the end users.

6.3 Which components should be employed in a code search engine?

Research questions one and two form the basis to answer the third research question. The evaluated components and the concrete implementation of the research prototype together show which components should and can be employed together in a code search engine. For the intent-enhanced code search engines, the following components were used together to create the best code search experience based on recent research. These components and their respective reasoning can be seen in figure 2.3 and in figure 2.4.

1. Indexing - CodeBert was used to index the code snippets as it is a pre-trained codee input encoder/decoder and can be extended in the future using Abstract Syntax Tree or other methods.

2. Querying - RASA AI was used as a framework for easily creating task-based conversational AI, allowing flexibility in generating user intent or sending a query to the search algorithm.

3. Searching/Ranking - Reinforcement learning and Cosine similarity were used to search the indexed code snippets using an encoded query and the encoded code snippets.

4. Intent modelling - The intent modelling components used in the code search engine were a combination of RASA's DIETClassifier machine learning algorithm along with a multiple choice modelling system, allowing the user to pick the best answers from a set of 10 results, which in time trains the algorithm based on the human feedback given.

6.4 How can the code search engine be evaluated?

The final research question is answered by creating the research prototype and evaluating it during the experiment in this thesis, showing an evaluation method along with how end users interacted with an intent-enhanced code search engine. This study combined three areas for the evaluation, the Design Science relevance cycle, an Experiment and a TAM2-based questionnaire.

6.4.1 Design Science

Applying the relevance cycle while evaluating the different code search engines and applying feedback attained from the users, both code search engines were constantly improving. The intent-enhanced code search engine is self-learning through the feedback given by the user. This self-learning behaviour improved the personal code search experience of the user, showing better alternatives and adopting the search process to the user's needs. This allows the intent-enhanced code search engine to constantly evaluate its results based on human feedback and create its dataset of different users and their intention. On a large scale, this may show promising results in mapping user intent in code search.

6.4.2 Experiment

The experiment was analysed both quantitatively and qualitatively. Using different techniques described in Chapter 4. The intent-enhanced code search engine outperformed the baseline code search engine regarding results. While the baseline code search engine was sometimes able to instantly come up with the specific solution to a problem described by the user, when the queries became more complex and did not have a single answer, the intent-enhanced code search engine tended to assist the participant in finding the right direction or scope.

6.4.3 TAM2

Besides evaluating both code search engines, the participants also answered questions from a TAM2 questionnaire. It showed overall positive results, with the intention of actual use of the code search engine being steered primarily by the job relevance depicted by participants. The TAM2 results show that the participants found the intent-enhanced code search engine easy to use and learn, and the results provided strong alternatives to what they were looking for. Many participants mentioned that after finetuning and indexing their specific codebase, they would consider using the code search engine in their weekly routine.

6.5 How can code search engines understand software developers' intentions?

The main objective of this study was to gain insight into the workings of semantic code search engines and evaluate how the end-user experience could be improved by applying intent modelling to this process. Using many different techniques described in Chapter 2 and 3, two semantic code search engines were created, one applying intent modelling techniques and one without using them. By evaluating the intent modelling in the code search engine, we get a broad image of the improvements intent modelling brings, but also into the research gaps still existent. Unfortunately, there is no uniform answer to the main research question, as all software developers, user stories, and cases differ. However, this study showed the successful implementation of intent modelling techniques within semantic code search and showed what elements improved the user experiences and possible adoption of such a system. The near future might have Large Language Models predict user intent on a per-use case basis, and the recent rise of AI systems may be able to brute force itself into understanding human intent at any time. While software developers' intentions are not yet fully

mapped when using search engines, this study takes an exploratory leap in semantic code search and intent modelling, showing a good baseline for future studies.

6.6 Future Research Directions

The findings of this study pave the way for future research in the field of semantic code search. Several potential research directions can be explored based on the identified limitations and areas for improvement. For example, further investigations can focus on refining the search algorithms, enhancing the recommendation accuracy, and expanding the scope of languages and code repositories covered. Additionally, exploring the integration of machine learning techniques, natural language processing, or other advanced technologies can contribute to developing more efficient and effective code search systems.

Appendix A

SLR Data

All data for the literature study, the raw experiment, and TAM2 data, as well as github repository used in this thesis, can be found at Zenodo:
https://zenodo.org/record/8038948 https://zenodo.org/record/8038520

Appendix B

User Experience Evaluation

This appendix contains the setup for the user experience evaluation.
Three separate Google Forms were used during the evaluation of the intent-enhanced search engine. Two were used to survey the feedback users had on the results provided by the search engines, and to avoid bias one was for the users using the baseline first, while the second one was filled in by users who used the intent-enhanced system first. The form itself does not say whether it is used for the baseline first or the intent-enhanced system first, to not create possible bias.

Baseline First

Intent First

The third Google Form was used to gather qualitative feedback using the TAM2 model: TAM2

The Baseline Endpoint was used by calling a Flask endpoint at port 8080 and the Intent Enhanced chatbot was used by calling a custom VueJS application running on port 5173. These search engines are both called an ElasticSearch index instance running at the machine at port 9200. To conveniently call these different endpoints Teamviewer or a reverse proxy was used, to ensure a secure connection to the applications.

Appendix C

User Consent Form

This appendix shows the participant consent form to be signed by the participants in the experiment. It contains both an English version as well as a Dutch version for the primarily Dutch participants.

C.1 English

Information for subjects invited to participate in (social) scientific research

A study on using user intent modelling to improve semantic code search results 31-03-2023, Utrecht

Dear Sir, Madam,

Introduction

Through this letter, we would like to invite you to participate in the research project **A study on using user intent modelling to improve semantic code search results**. The purpose of this study is to assess the extended value of applying user intent modelling to the semantic search process of code snippets.

Design/execution of the study The study will be executed by having the participant use both a baseline semantic search engine along with Snippetsage, our user intent extended model of the semantic search engine. A set of queries is given to each participant and the participant is then asked to assess the answers in terms of relevance for each answer. After several queries and their answers have been evaluated the participant is asked to fill out a questionnaire on their usage and predicted future use of the system. General feedback is also asked and highly appreciated for the study.

Background of the study This study uses two semantic code search applications, both using the same programmed backend but with the main difference being one application calls the algorithm while the other uses intent modelling to improve the results based on the user intent. Different machine learning algorithms are used to embed the query string into vector space to improve search results, the main goal of the research is to assess to what extent user intent modelling can improve the search process and results.

What is expected of you as a participant The participant is expected to join a 15/30-minute experiment in using both a regular semantic code search engine given a set of queries as well as in our created intent extended search engine. After this, we ask you to fill in a questionnaire to assess the quality and possible improvements of the system.

Confidentiality of data processing None of your personal data aside from your first name will be stored for credibility and future reference of the study. Other data that is stored consists of the anonymized answers to the questionnaire, along with anonymized conversation data with the conversational agent. Any possible connections to your real-life identity will be destroyed before any publication.

Voluntary participation Participation in this study is voluntary. You can end your participation in the study at any time, without any explanation and without any negative consequences. If you end your participation, we will use the data collected up to that point, unless you explicitly inform us otherwise.

Independent contact and complaints officer If you have an official complaint about the study, you can send an email to the complaints officer at klachtenfunctionaris-fetcsocwet@uu.nl.

If, after reading this information letter, you decide to take part in the research, I would kindly ask you to sign the attached reply slip and hand it to the researcher(s).

With kind regards,

Chris Pfaff,

Master Student Business Informatics

Consent statement: I hereby declare that I have read the information letter about the thesis A study on using user intent modelling to improve semantic code search results and agree to participate in the study.

First Name <Do not include any further identifying information, such as a subject number or other codes, date of birth, etc.>

Date:

Signature:

C.2 Dutch

Informatie voor deelnemers uitgenodigd om deel te nemen aan (sociaal) wetenschappelijk onderzoek **Een studie over het gebruik van gebruikersintentie-modellering om de resultaten van semantische code-zoekopdrachten te verbeteren** 31-03-2023, Utrecht Geachte heer, mevrouw, **Inleiding** Via deze brief willen wij u uitnodigen om deel te nemen aan het onderzoeksproject "**Een studie over het gebruik van gebruikersintentie-modellering om de resultaten van semantische code-zoekopdrachten te verbeteren**". Het doel van deze studie is om de toegevoegde waarde te beoordelen van het toepassen van gebruikersintentie-modellering op het semantische zoekproces van codefragmenten.

Ontwerp/uitvoering van de studie De studie zal worden uitgevoerd door de deelnemer zowel een basislijn semantische zoekmachine als Snippetsage te laten gebruiken, ons gebruikersintentie-uitgebreide model van de semantische zoekmachine. Aan elke deelnemer wordt een set zoekopdrachten gegeven en vervolgens wordt de deelnemer gevraagd om de antwoorden te beoordelen in termen van relevantie voor elk antwoord. Nadat verschillende zoekopdrachten en hun antwoorden zijn geëvalueerd, wordt de deelnemer gevraagd om een vragenlijst in te vullen over hun gebruik en voorspelde toekomstige gebruik van het systeem. Algemene feedback wordt ook gevraagd en zeer op prijs gesteld voor de studie.

Achtergrond van de studie Deze studie maakt gebruik van twee semantische code-zoektoepassingen, beide met dezelfde geprogrammeerde backend, maar met als belangrijkste verschil dat de ene toepassing het algoritme oproept terwijl de andere gebruikersintentie-modellering gebruikt om de resultaten te verbeteren op basis van de gebruikersintentie. Verschillende machine learning-algoritmen worden gebruikt om de zoekopdrachtstring in vectorruimte in te sluiten om zoekresultaten te verbeteren, het belangrijkste doel van het onderzoek is om te beoordelen in hoeverre gebruikersintentie-modellering het zoekproces en de resultaten kan verbeteren.

Wat er van u wordt verwacht als deelnemer Van de deelnemer wordt verwacht dat hij deelneemt aan een experiment van 15/30 minuten waarin zowel een reguliere semantische code-zoekmachine wordt gebruikt voor een set zoekopdrachten als onze gecreëerde intentie-uitgebreide zoekmachine. Daarna vragen wij u om een vragenlijst in te vullen om de kwaliteit en mogelijke verbeteringen van het systeem te beoordelen.

Vertrouwelijkheid van gegevensverwerking Geen van uw persoonlijke gegevens, behalve uw voornaam, wordt opgeslagen voor geloofwaardigheid en toekomstige referentie van het onderzoek. Andere opgeslagen gegevens bestaan uit geanonimiseerde antwoorden op de vragenlijst, samen met geanonimiseerde gespreksgegevens met de gespreksagent. Eventuele mogelijke verbanden met uw echte identiteit zullen worden vernietigd voordat er enige publicatie plaatsvindt.

Vrijwillige deelname Deelname aan dit onderzoek is vrijwillig. U kunt op elk moment uw deelname aan het onderzoek beëindigen, zonder enige uitleg en zonder enige negatieve gevolgen. Als u uw deelname beëindigt, zullen we de tot dat moment verzamelde gegevens gebruiken, tenzij u ons expliciet informeert dat u niet wilt dat we uw gegevens gebruiken.

Onafhankelijke contactpersoon en klachtenfunctionaris Als u een officiële klacht heeft over het onderzoek, kunt u een e-mail sturen naar de klachtenfunctionaris op klachtenfunctionaris-fetcsocwet@uu.nl. Als u na het lezen van deze informatiebrief besluit om deel te nemen aan het onderzoek, wil ik u vriendelijk verzoeken om het bijgevoegde antwoordformulier te ondertekenen en aan de onderzoeker(s) te overhandigen. Met vriendelijke groet, Chris Pfaff, Masterstudent Bedrijfsinformatica

Toestemmingsverklaring: Hierbij verklaar ik dat ik de informatiebrief over de scriptie Een studie naar het gebruik van gebruikersintentie-modellering om de resultaten van semantisch code-zoeken te verbeteren heb gelezen en akkoord ga om deel te nemen aan het onderzoek. Voornaam <Voeg geen verdere identificerende informatie toe, zoals een subjectnummer of andere codes, geboortedatum, etc.>

Datum:

Handtekening:

Appendix D

Code

This Appendix shows R, Python and other codes used during the creation of the software artifact and during quantitative and qualitative analysis.

D.1 R Code for Statistical Analysis

D.1.1 R Code for Quantitative Analysis

The following code snippets show the ANOVA and T-Tests performed to do the quantitative analysis depicted in Chapter 4.

```r
#Read CSV Data into dataframe
data <- read.csv("ExperimentAnova.csv")

#Split query results per search engine
baseline_queries <- paste0("baseline_query", 1:3)
intent_queries <- paste0("intent_query", 1:3)

#Create seperate dataframes for the baseline and intent-enhanced results
baseline <- data[, baseline_queries]
intent <- data[, intent_queries]

#Split the independent variables in the seperate queries and the code search engine system used
queries <- rep(c("query1", "query2", "query3"), each = nrow(data))

system <- rep(c("Baseline", "Intent"), each = nrow(data) * 3)

#Combine the dataframes again
combined_data <- data.frame(system, queries, performance = c(as.matrix(baseline), as.matrix(intent)))

#Use the Psych package to do the ANOVA analysis
result <- aov(performance ~ system * queries, data = combined_data)

anova_summary <- summary(result)
print(anova_summary)

```

```r
#Do a sampled T-test for all three different queries (replaced baseline_queryX and intent_queryX)
result_ttest <- t.test(data$baseline_queryX, data$intent_queryX, paired = TRUE, na.action = na.omit)

print(result_ttest)
```

D.1.2 R Code for TAM2 Factor Analysis and Cronbach's Alpha calculations

The following code snippets show the Factor Analysis and Cronbach's Alpha calculations used for the quantitative analysis of the TAM2 questionnaire described in Chapter 4.

```r
TAM2Data <- read.csv("TAM2Short.csv")

#Descriptive Statistics
likert_long <- melt(TAM2Data)

likert_freq <- likert_long %>%
  group_by(variable, value) %>%
  summarise(Frequency = n())

likert_perc <- likert_freq %>%
  group_by(variable) %>%
  mutate(Percentage = Frequency / sum(Frequency) * 100) %>%
  mutate(Label = paste0(Frequency, " (", round(Percentage, 2), "%)"))

mean_vals <- TAM2Data %>%
  summarise(across(everything(), mean, na.rm = TRUE))

median_vals <- TAM2Data %>%
  summarise(across(everything(), median, na.rm = TRUE))

mode_vals <- TAM2Data %>%
  summarise(across(everything(), function(x) {
    unique_val <- unique(x)
    unique_val[which.max(tabulate(match(x, unique_val)))]
  }))

sd_vals <- TAM2Data %>%
  summarise(across(everything(), sd, na.rm = TRUE))

range_vals <- TAM2Data %>%
  summarise(across(everything(), function(x) diff(range(x, na.rm = TRUE))))

stats_df <- bind_rows(mean_vals, median_vals, mode_vals, sd_vals, range_vals)
```

```
rownames(stats_df) <- c("Mean", "Median", "Mode", "SD", "Range")

likert_colors <- c("#AFE1AF", "#AFE1AF", "#AFE1AF", "#FFC300", "#FF5733", "#FF5733", "#FF5733")

likert_perc$value <- factor(likert_perc$value)

heatmap_plot <- ggplot(likert_perc, aes(x = value, y = variable, fill = value)) +
  geom_tile(color = "white", width = 0.9, height = 0.9) +
  scale_fill_manual(values = likert_colors, drop = FALSE) +
  labs(x = "Likert Scale Level", y = "Question") +
  ggtitle("Heatmap of Likert Scale Responses") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 0, vjust = 0.5),
        plot.title = element_text(hjust = 0.5),
        legend.position = "none")

heatmap_plot <- heatmap_plot +
  geom_text(aes(label = Label), size = 3, color = "black")

heatmap_plot <- heatmap_plot + scale_y_discrete(limits=rev)

heatmap_plot

#Create correlation matrix for TAM2 data
correlation_matrix <- cor(TAM2Data)
corrplot::corrplot(correlation_matrix)

#Cronbach Alpha results
alpha_result <- psych::alpha(TAM2Data, check.keys=TRUE)

print(alpha_result)

#Principal analysis for factor determination using Eigenvalues

principal_analysis <- principal(TAM2Data, nfactors = 10, rotate = "none")

eigenvalues <- principal_analysis$values

par(mar = c(5, 5, 4, 2) + 0.1)

# Plot the scree plot
plot(1:length(eigenvalues), eigenvalues, type = "b",
```

```
76      main = "Scree Plot", xlab = "Factor", ylab = "
           Eigenvalue")
77
78  abline(h = 1, col = "red")
79
80  text(1:length(eigenvalues), eigenvalues, labels = 1:length
       (eigenvalues), pos = 4)
81
82
83  #Try various different rotation methods and explore the
       different results of all analyses.
84  pa <- fa(r = TAM2Data,
85           nfactors = 3,
86           rotate = "varimax",
87           fm = "pa"
88  )
89
90  papro <- fa(r = TAM2Data,
91           nfactors = 3,
92           rotate = "promax",
93           fm = "pa")
94
95  ml <- fa(r = TAM2Data,
96           nfactors = 3,
97           rotate = "varimax",
98           fm = "ml")
99
100 pa
101
102 papro
103
104 ml
```

D.2 Software Artifact Code

The code used for the creation of both software artifacts is publicly available at Github under the MIT License.

Bibliography

[1] Wasi Uddin Ahmad, Kai-Wei Chang, and Hongning Wang. "Context attentive document ranking and query suggestion". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2019, pp. 385–394.

[2] Wasi Uddin Ahmad et al. "A transformer-based approach for source code summarization". In: *arXiv preprint arXiv:2005.00653* (2020).

[3] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.

[4] Miltiadis Allamanis, Hao Peng, and Charles Sutton. "A convolutional attention network for extreme summarization of source code". In: *International conference on machine learning*. PMLR. 2016, pp. 2091–2100.

[5] Uri Alon et al. "code2seq: Generating sequences from structured representations of code". In: *arXiv preprint arXiv:1808.01400* (2018).

[6] Uri Alon et al. "code2vec: Learning distributed representations of code". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–29.

[7] Mehdi Bahrami et al. "PyTorrent: A Python Library Corpus for Large-scale Language Models". In: *arXiv preprint arXiv:2110.01710* (2021).

[8] Sushil Krishna Bajracharya and Cristina Videira Lopes. "Analyzing and mining a code search engine usage log". In: *Empirical Software Engineering* 17.4 (2012), pp. 424–466.

[9] Elena Baninemeh, Siamak Farshidi, and Slinger Jansen. "A decision model for decentralized autonomous organization platform selection: Three industry case studies". In: *Blockchain: Research and Applications* (2023), p. 100127.

[10] Michael Bendersky et al. "Learning from user interactions in personal search via attribute parameterization". In: *Proceedings of the tenth ACM international conference on web search and data mining*. 2017, pp. 791–799.

[11] Francesco Bertolotti and Walter Cazzola. "Fold2Vec: Towards a Statement Based Representation of Code for Code Comprehension". In: *ACM Transactions on Software Engineering and Methodology* (2022).

[12] Hudson Borges, Andre Hora, and Marco Tulio Valente. "Understanding the factors that impact the popularity of GitHub repositories". In: *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE. 2016, pp. 334–344.

[13] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual web search engine". In: *Computer networks and ISDN systems* 30.1-7 (1998), pp. 107–117.

[14] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[15] Tanja Bunk et al. “Diet: Lightweight language understanding for dialogue systems”. In: *arXiv preprint arXiv:2004.09936* (2020).

[16] Wanling Cai and Li Chen. “Towards a Taxonomy of User Feedback Intents for Conversational Recommendations.” In: *RecSys (Late-Breaking Results)*. 2019, pp. 51–55.

[17] Jose Cambronero et al. “When deep learning met code search”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 964–974.

[18] Brock Angus Campbell and Christoph Treude. “NLP2Code: Code snippet content assist via natural language tasks”. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2017, pp. 628–632.

[19] Kaibo Cao et al. “Automated query reformulation for efficient search based on query logs from stack overflow”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 1273–1285.

[20] Edward G Carmines and Richard A Zeller. *Reliability and validity assessment*. Sage publications, 1979.

[21] Jia Chen et al. “Towards a better understanding of query reformulation behavior in web search”. In: *Proceedings of the Web Conference 2021*. 2021, pp. 743–755.

[22] Long Chen, Wei Ye, and Shikun Zhang. “Capturing source code semantics via tree-based convolution over API-enhanced AST”. In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 2019, pp. 174–182.

[23] Zimin Chen et al. “PLUR: A unifying, graph-based view of program learning, understanding, and repair”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 23089–23101.

[24] Mengyao Cui et al. “Introduction to the k-means clustering algorithm based on the elbow method”. In: *Accounting, Auditing and Finance* 1.1 (2020), pp. 5–8.

[25] Samip Dahal, Adyasha Maharana, and Mohit Bansal. “Scotch: A Semantic Code Search Engine for IDEs”. In: *Deep Learning for Code Workshop*. 2022.

[26] Fred D Davis. “Perceived usefulness, perceived ease of use, and user acceptance of information technology”. In: *MIS quarterly* (1989), pp. 319–340.

[27] Jill Denner and Shannon Campe. “Equity and Inclusion in Computer Science Education: Research on Challenges and Opportunities”. In: *Computer Science Education: Perspectives on Teaching and Learning in School* (2023), p. 85.

[28] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[29] Dawn Drain et al. “Generating code with the help of retrieved template functions and stack overflow answers”. In: *arXiv preprint arXiv:2104.05310* (2021).

[30] Marcel Dunaiski, Gillian J Greene, and Bernd Fischer. “Exploratory search of academic publication and citation data using interactive tag cloud visualizations”. In: *Scientometrics* 110.3 (2017), pp. 1539–1571.

[31] Frederico A Durão et al. “Applying a semantic layer in a source code search tool”. In: *Proceedings of the 2008 ACM symposium on Applied computing*. 2008, pp. 1151–1157.

[32] Sen Fang et al. “Self-attention networks for code search”. In: *Information and Software Technology* 134 (2021), p. 106542.

[33] Siamak Farshidi. “Multi-Criteria Decision-Making in Software Production”. PhD thesis. Utrecht University, 2020.

[34] Siamak Farshidi, Izaak Beer Kwantes, and Slinger Jansen. “Business process modeling language selection for research modelers”. In: *Software and Systems Modeling* (2023), pp. 1–26.

[35] Siamak Farshidi and Zhiming Zhao. “An Adaptable Indexing Pipeline for Enriching Meta Information of Datasets from Heterogeneous Repositories”. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2022, pp. 472–484.

[36] Zhangyin Feng et al. “Codebert: A pre-trained model for programming and natural languages”. In: *arXiv preprint arXiv:2002.08155* (2020).

[37] Murray J Fisher and Andrea P Marshall. “Understanding descriptive statistics”. In: *Australian critical care* 22.2 (2009), pp. 93–97.

[38] Zhipeng Gao et al. “Generating question titles for stack overflow from mined code snippets”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29.4 (2020), pp. 1–37.

[39] Songwei Ge et al. “Personalizing search results using hierarchical RNN with query-aware attention”. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2018, pp. 347–356.

[40] *Github Website Kernel Description*. https://github.com/. Accessed: 24-10-2022.

[41] Luca Di Grazia and Michael Pradel. “Code search: A survey of techniques for finding code”. In: *ACM Computing Surveys (CSUR)* (2022).

[42] Wenchao Gu et al. “Accelerating Code Search with Deep Hashing and Code Classification”. In: *arXiv preprint arXiv:2203.15287* (2022).

[43] Wenchao Gu et al. “CRaDLe: Deep code retrieval based on semantic dependency learning”. In: *Neural Networks* 141 (2021), pp. 385–394.

[44] Daya Guo et al. “Graphcodebert: Pre-training code representations with data flow”. In: *arXiv preprint arXiv:2009.08366* (2020).

[45] Rahul Gupta, Aditya Kanade, and Shirish Shevade. “Neural attribution for semantic bug-localization in student programs”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[46] Simon Haykin and N Network. “A comprehensive foundation”. In: *Neural networks* 2.2004 (2004), p. 41.

[47] Ben He, Jimmy Xiangji Huang, and Xiaofeng Zhou. “Modeling term proximity for probabilistic information retrieval models”. In: *Information Sciences* 181.14 (2011), pp. 3017–3031.

[48] Xiangnan He et al. “Neural collaborative filtering”. In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.

[49] Alan R Hevner. “A three cycle view of design science research”. In: *Scandinavian journal of information systems* 19.2 (2007), p. 4.

[50] Geert Heyman and Tom Van Cutsem. “Neural code search revisited: Enhancing code snippet retrieval through natural language intent”. In: *arXiv preprint arXiv:2008.12193* (2020).

[51] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[52] Erfan Al-Hossami and Samira Shaikh. "A Survey on Artificial Intelligence for Source Code: A Dialogue Systems Perspective". In: *arXiv preprint arXiv:2202.04847* (2022).

[53] Gang Hu et al. "Neural joint attention code search over structure embeddings for software q&a sites". In: *Journal of Systems and Software* 170 (2020), p. 110773.

[54] Junjie Huang et al. "Cosqa: 20,000+ web queries for code search and question answering". In: *arXiv preprint arXiv:2105.13239* (2021).

[55] Hamel Husain. "Towards natural language semantic code search, Sep 2018". In: *URL https://githubengineering. com* (2018).

[56] Hamel Husain et al. "Codesearchnet challenge: Evaluating the state of semantic code search". In: *arXiv preprint arXiv:1909.09436* (2019).

[57] Abdullah Al Ishtiaq et al. "BERT2Code: Can Pretrained Language Models be Leveraged for Code Search?" In: *arXiv preprint arXiv:2104.08017* (2021).

[58] Srinivasan Iyer et al. "Summarizing source code using a neural attention model". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, pp. 2073–2083.

[59] Dietmar Jannach et al. "A survey on conversational recommender systems". In: *ACM Computing Surveys (CSUR)* 54.5 (2021), pp. 1–36.

[60] Slinger Jansen et al. "SearchSECO: A Worldwide Index of the Open Source Software Ecosystem". In: *The 19th Belgium-Netherlands Software Evolution Workshop, BENEVOL 202*. CEUR-WS. org. 2020.

[61] Tae-Hwan Jung. "Commitbert: Commit message generation using pre-trained programming language model". In: *arXiv preprint arXiv:2105.14242* (2021).

[62] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[63] Kimiya Keyvan and Jimmy Xiangji Huang. "How to approach ambiguous queries in conversational search? A survey of techniques, approaches, tools and challenges". In: *ACM Computing Surveys (CSUR)* (2022).

[64] Kisub Kim et al. "FaCoY: a code-to-code search engine". In: *Proceedings of the 40th International Conference on Software Engineering*. 2018, pp. 946–957.

[65] Barbara Kitchenham. "Procedures for performing systematic reviews". In: *Keele, UK, Keele University* 33.2004 (2004), pp. 1–26.

[66] Barbara Kitchenham and Stuart Charters. "Guidelines for performing systematic literature reviews in software engineering". In: *ebse technical report.* (2007).

[67] Rainer Koschke, Raimar Falke, and Pierre Frenzel. "Clone detection using abstract syntax suffix trees". In: *2006 13th Working Conference on Reverse Engineering*. IEEE. 2006, pp. 253–262.

[68] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[69] Oliver Laitenberger and Horst M Dreyer. "Evaluating the usefulness and the ease of use of a web-based inspection data collection tool". In: *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No. 98TB100262)*. IEEE. 1998, pp. 122–132.

[70] Aristomenis S Lampropoulos and George A Tsihrintzis. "Machine learning paradigms". In: *Appl. Recomm. Syst. Switz. Springer Intern. Publ* (2015).

[71] Wei Li et al. "Learning code-query interaction for enhancing code searches". In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2020, pp. 115–126.

[72] Yuxi Li. "Deep reinforcement learning: An overview". In: *arXiv preprint arXiv:1701.07274* (2017).

[73] Rong Liang et al. "AstBERT: Enabling Language Model for Code Understanding with Abstract Syntax Tree". In: *arXiv preprint arXiv:2201.07984* (2022).

[74] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. "The global k-means clustering algorithm". In: *Pattern recognition* 36.2 (2003), pp. 451–461.

[75] Xiang Ling et al. "Deep graph matching and searching for semantic code retrieval". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15.5 (2021), pp. 1–21.

[76] Chao Liu et al. "CodeMatcher: Searching Code Based on Sequential Semantics of Important Query Words". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.1 (2021), pp. 1–37.

[77] Jason Liu et al. "Neural query expansion for code search". In: *Proceedings of the 3rd acm sigplan international workshop on machine learning and programming languages*. 2019, pp. 29–37.

[78] Xuye Liu et al. "HAConvGNN: Hierarchical attention based convolutional graph neural network for code documentation generation in jupyter notebooks". In: *arXiv preprint arXiv:2104.01002* (2021).

[79] Shuai Lu et al. "Codexglue: A machine learning benchmark dataset for code understanding and generation". In: *arXiv preprint arXiv:2102.04664* (2021).

[80] Sifei Luan et al. "Aroma: Code recommendation via structural code search". In: *Proceedings of the ACM on Programming Languages* 3.OOPSLA (2019), pp. 1–28.

[81] Lev Manovich. *Software takes command*. Vol. 5. A&C Black, 2013.

[82] Antonio Mastropaolo et al. "Studying the usage of text-to-text transfer transformer to support code-related tasks". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 336–347.

[83] George Varghese Mathew. "Cross-Language Code Similarity and Applications in Clone Detection and Code Search". PhD thesis. North Carolina State University, 2022.

[84] Collin McMillan et al. "Exemplar: A source code search engine for finding highly relevant applications". In: *IEEE Transactions on Software Engineering* 38.5 (2011), pp. 1069–1087.

[85] Donald Metzler and W Bruce Croft. "A markov random field model for term dependencies". In: *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. 2005, pp. 472–479.

[86] Tomas Mikolov et al. "Recurrent neural network based language model." In: *Interspeech*. Vol. 2. 3. Makuhari. 2010, pp. 1045–1048.

[87] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. "Recurrent models of visual attention". In: *Advances in neural information processing systems* 27 (2014).

[88] Agnès Mustar, Sylvain Lamprier, and Benjamin Piwowarski. "On the study of transformers for query suggestion". In: *ACM Transactions on Information Systems (TOIS)* 40.1 (2021), pp. 1–27.

[89] Iulian Neamtiu, Jeffrey S Foster, and Michael Hicks. "Understanding source code evolution using abstract syntax tree matching". In: *Proceedings of the 2005 international workshop on Mining software repositories*. 2005, pp. 1–5.

[90] Haoran Niu, Iman Keivanloo, and Ying Zou. "Learning to rank code examples for code search engines". In: *Empirical Software Engineering* 22.1 (2017), pp. 259–291.

[91] Robert N Oddy. "Information retrieval through man-machine dialogue". In: *Journal of documentation* 33.1 (1977), pp. 1–14.

[92] Long Ouyang et al. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[93] Matthew Partridge and Rafael A Calvo. "Fast dimensionality reduction and simple PCA". In: *Intelligent data analysis* 2.3 (1998), pp. 203–214.

[94] Dinglan Peng et al. "How could Neural Networks understand Programs?" In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8476–8486.

[95] Chris Pfaff et al. "A code search engine for software ecosystems". In: *CEUR Workshop Proceedings*. Vol. 3245. CEUR WS. 2022.

[96] Varot Premtoon, James Koppel, and Armando Solar-Lezama. "Semantic code search via equational reasoning". In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2020, pp. 1066–1082.

[97] Ruchir Puri et al. "Project codenet: A large-scale ai for code dataset for learning a diversity of coding tasks". In: *arXiv preprint arXiv:2105.12655* 1035 (2021).

[98] Weizhen Qi et al. "ProphetNet-x: large-scale pre-training models for English, Chinese, multi-lingual, dialog, and code generation". In: *arXiv preprint arXiv:2104.08006* (2021).

[99] QSR International Pty Ltd. *NVivo*. Computer software. Version Latest Version. Accessed: May 21, 2023. 2023. URL: https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home.

[100] Chen Qu et al. "User intent prediction in information-seeking conversations". In: *Proceedings of the 2019 Conference on Human Information Interaction and Retrieval*. 2019, pp. 25–33.

[101] Filip Radlinski and Nick Craswell. "A theoretical framework for conversational search". In: *Proceedings of the 2017 conference on conference human information interaction and retrieval*. 2017, pp. 117–126.

[102] Sailaja Rajanala et al. “DeSCoVeR: Debiased Semantic Context Prior for Venue Recommendation”. In: *Proceedings of the 45th International ACM SIGIR Conference o n Research and Development in Information Retrieval*. 2022, pp. 2456–2461.

[103] Ashwin Ram et al. “Conversational ai: The science behind the alexa prize”. In: *arXiv preprint arXiv:1801.03604* (2018).

[104] Nikitha Rao, Chetan Bansal, and Joe Guan. “Search4Code: Code search intent classification using weak supervision”. In: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE. 2021, pp. 575–579.

[105] Nikitha Rao et al. “Analyzing web search behavior for software engineering tasks”. In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE. 2020, pp. 768–777.

[106] Matthew Richardson, Amit Prakash, and Eric Brill. “Beyond PageRank: machine learning for static ranking”. In: *Proceedings of the 15th international conference on World Wide Web*. 2006, pp. 707–715.

[107] Stephen Robertson, Hugo Zaragoza, et al. “The probabilistic relevance framework: BM25 and beyond”. In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389.

[108] Vitaly Romanov, Vladimir Ivanov, and Giancarlo Succi. “Representing Programs with Dependency and Function Call Graphs for Learning Hierarchical Embeddings.” In: *ICEIS (2)*. 2020, pp. 360–366.

[109] Tuukka Ruotsalo et al. “Interactive intent modeling for exploratory search”. In: *ACM Transactions on Information Systems (TOIS)* 36.4 (2018), pp. 1–46.

[110] Tuukka Ruotsalo et al. “Interactive intent modeling: Information discovery beyond search”. In: *Communications of the ACM* 58.1 (2014), pp. 86–92.

[111] Saksham Sachdev et al. “Retrieval on source code: a neural code search”. In: *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 2018, pp. 31–41.

[112] Franco Scarselli et al. “The graph neural network model”. In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[113] Yelong Shen et al. “A latent semantic model with convolutional-pooling structure for information retrieval”. In: *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*. 2014, pp. 101–110.

[114] Fran Silavong et al. “DeSkew-LSH based Code-to-Code Recommendation Engine”. In: *arXiv preprint arXiv:2111.04473* (2021).

[115] Susan Elliott Sim et al. “How well do search engines support code retrieval on the web?” In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 21.1 (2011), pp. 1–25.

[116] Nisan Stiennon et al. “Learning to summarize with human feedback”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 3008–3021.

[117] Zhensu Sun et al. “PSCS: A path-based neural model for semantic code search”. In: *arXiv preprint arXiv:2008.03042* (2020).

[118] Dinoj Surendran and Gina-Anne Levow. “Dialog act tagging with support vector machines and hidden Markov models.” In: *Interspeech*. 2006.

[119] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[120] Krysta M Svore and Christopher JC Burges. "A machine learning approach for improved BM25 retrieval". In: *Proceedings of the 18th ACM conference on Information and knowledge management*. 2009, pp. 1811–1814.

[121] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. "Introduction to multilayer feed-forward neural networks". In: *Chemometrics and intelligent laboratory systems* 39.1 (1997), pp. 43–62.

[122] Mohsen Tavakol and Reg Dennick. "Making sense of Cronbach's alpha". In: *International journal of medical education* 2 (2011), p. 53.

[123] Jaime Teevan, Susan T Dumais, and Daniel J Liebling. "To personalize or not to personalize: modeling queries with variation in user intent". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 2008, pp. 163–170.

[124] Medha Umarji, Susan Elliott Sim, and Crista Lopes. "Archetypal internet-scale source code searching". In: *IFIP International Conference on Open Source Systems*. Springer. 2008, pp. 257–263.

[125] Ali Vardasbi, Fatemeh Sarvi, and Maarten de Rijke. "Probabilistic Permutation Graph Search: Black-Box Optimization for Fairness in Ranking". In: *arXiv preprint arXiv:2204.13765* (2022).

[126] Viswanath Venkatesh and Fred D Davis. "A theoretical extension of the technology acceptance model: Four longitudinal field studies". In: *Management science* 46.2 (2000), pp. 186–204.

[127] Yao Wan et al. "Improving automatic source code summarization via deep reinforcement learning". In: *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 2018, pp. 397–407.

[128] Chaozheng Wang et al. "Enriching query semantics for code search with reinforcement learning". In: *Neural Networks* 145 (2022), pp. 22–32.

[129] Hao Wang et al. "COSEA: Convolutional Code Search with Layer-wise Attention". In: *arXiv preprint arXiv:2010.09520* (2020).

[130] Shaowei Wang, David Lo, and Lingxiao Jiang. "Code search via topic-enriched dependence graph matching". In: *2011 18th Working Conference on Reverse Engineering*. IEEE. 2011, pp. 119–123.

[131] Wenhan Wang et al. "Detecting code clones with graph neural network and flow-augmented abstract syntax tree". In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2020, pp. 261–271.

[132] Wenhua Wang et al. "Trans^ 3: A transformer-based framework for unifying code summarization and code search". In: *arXiv preprint arXiv:2003.03238* (2020).

[133] Xin Wang et al. "Syncobert: Syntax-guided multi-modal contrastive pre-training for code representation". In: *arXiv preprint arXiv:2108.04556* (2021).

[134] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[135] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[136] Xiaojun Xu et al. “Neural network-based graph embedding for cross-platform binary code similarity detection”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 363–376.

[137] Di Yang, Aftab Hussain, and Cristina Videira Lopes. “From query to usable code: an analysis of stack overflow code snippets”. In: *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE. 2016, pp. 391–401.

[138] Guang Yang et al. “Deeppseudo: Deep pseudo-code generation via transformer and code feature extraction”. In: *arXiv preprint arXiv:2102.06360* (2021).

[139] Liu Yang et al. “IART: Intent-aware response ranking with transformers in information-seeking conversation systems”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 2592–2598.

[140] Yingrui Yang et al. “Lightweight composite re-ranking for efficient keyword search with BERT”. In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 2022, pp. 1234–1244.

[141] Jing Yao et al. “RLPS: A Reinforcement Learning–Based Framework for Personalized Search”. In: *ACM Transactions on Information Systems (TOIS)* 39.3 (2021), pp. 1–29.

[142] Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. “Coacor: Code annotation for code retrieval with reinforcement learning”. In: *The world wide web conference*. 2019, pp. 2203–2214.

[143] Ziyu Yao et al. “Staqc: A systematically mined question-code dataset from stack overflow”. In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 1693–1703.

[144] Wei Ye et al. “Leveraging code generation to improve code retrieval and summarization via dual learning”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 2309–2319.

[145] Jun Yin et al. “Neural generative question answering”. In: *arXiv preprint arXiv:1512.01337* (2015).

[146] Pengcheng Yin et al. “Learning to mine aligned code and natural language pairs from stack overflow”. In: *2018 IEEE/ACM 15th international conference on mining software repositories (MSR)*. IEEE. 2018, pp. 476–486.

[147] Hao Yu et al. “Incorporating Code Structure and Quality in Deep Code Search”. In: *Applied Sciences* 12.4 (2022), p. 2051.

[148] Chen Zeng et al. “deGraphCS: Embedding Variable-based Flow Graph for Neural Code Search”. In: *arXiv preprint arXiv:2103.13020* (2021).

[149] Feng Zhang et al. “Expanding queries for code search using semantically related api class-names”. In: *IEEE Transactions on Software Engineering* 44.11 (2017), pp. 1070–1082.

[150] Hanlei Zhang et al. “Discovering new intents with deep aligned clustering”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 16. 2021, pp. 14365–14373.

[151] Shuai Zhang et al. “Deep learning based recommender system: A survey and new perspectives”. In: *ACM Computing Surveys (CSUR)* 52.1 (2019), pp. 1–38.

[152] Ye Zhang and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1510.03820* (2015).

[153] Gang Zhao and Jeff Huang. "Deepsim: deep learning code functional similarity". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018, pp. 141–151.

[154] Zhiming Zhao et al. "Notebook-as-a-VRE (NaaVRE): From private notebooks to a collaborative cloud virtual research environment". In: *Software: Practice and Experience* 52.9 (2022), pp. 1947–1966.

[155] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81.

[156] Xin Zhou et al. "A map of threats to validity of systematic literature reviews in software engineering". In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2016, pp. 153–160.

[157] Yu Zhou et al. "Automatic source code summarization with graph attention networks". In: *Journal of Systems and Software* 188 (2022), p. 111257.

[158] Daniel M Ziegler et al. "Fine-tuning language models from human preferences". In: *arXiv preprint arXiv:1909.08593* (2019).