Alma Mater Studiorum · University of Bologna

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Master's Degree in Computer Science

LLMs and Essence: Developing a Chatbot to Support Software Engineering Practices

Supervisor:
Prof. Paolo Ciancarini

Presented by:
Sonia Nicoletti

Session III
Academic Year 2023/2024

"La lutte elle-même vers les sommets
suffit à remplir un cœur d'homme.
Il faut imaginer Sisyphe heureux."
Albert Camus

Sommario

Essence è uno standard progettato per descrivere le metodologie e le pratiche di sviluppo del software. Fornisce un linguaggio comune e strumenti concettuali che aiutano i team a organizzare il lavoro, comunicare efficacemente e adeguare le metodologie alle proprie esigenze, promuovendo una gestione più collaborativa e consapevole dei progetti software.

Questa tesi esplora l'integrazione dell'intelligenza artificiale, in particolare dei Large Language Model (LLM), con Essence. L'obiettivo principale è quello di sviluppare un chatbot supportato da un sistema di Retrieval-Augmented Generation (RAG) per assistere studenti e professionisti nell'apprendimento di Essence e nella sua applicazione pratica.

Dopo una panoramica sull'architettura degli LLM e sullo standard Essence, la tesi descrive la progettazione e l'implementazione del chatbot, delineando le motivazioni dietro alle diverse scelte progettuali e le strategie utilizzate per la sua ottimizzazione. In particolare, l'applicazione utilizza il modello Llama 3 insieme a un sistema RAG basato su un ensemble retriever. Un database di documenti selezionati relativi a Essence è stato utilizzato per eseguire ricerche per parole chiave e vettoriali, al fine di fornire un contesto più approfondito alle domande degli utenti.

Successivamente, per valutare l'efficacia del sistema, è stata condotta una serie di esperimenti che hanno esaminato sia la pertinenza dei contesti recuperati che la qualità delle risposte generate. L'analisi comparativa con un modello generico senza RAG ha dimostrato che il sistema proposto offre prestazioni superiori rispetto alla

sua controparte. La valutazione è stata condotta utilizzando un approccio duplice: da un lato, mediante metriche oggettive come il BERTScore, e dall'altro, attraverso un'analisi qualitativa basata su valutazioni umane.

I risultati di questa ricerca contribuiscono a evidenziare il potenziale degli LLM nel supportare l'apprendimento e la gestione delle pratiche di ingegneria del software. Sebbene siano necessari ulteriori test con utenti reali per perfezionare l'applicazione e comprenderne appieno le potenzialità, questa tesi pone le basi per future ricerche all'intersezione tra intelligenza artificiale e gestione dei progetti software, un'area che al momento rimane poco esplorata.

Contents

List of Figures

List of Tables

Introduction

This thesis explores the intersection between the fields of artificial intelligence and software engineering. Large language models, one of the latest advances in natural language processing, and Essence, a standard for the description of software engineering practices, will be the main topics.

Nowadays, large language models have become one of the most discussed topics not only in computer science research, but also in everyday conversations. With unsubstantiated claims and unreasonable expectations dominating the public discourse, it is becoming increasingly hard to distinguish what these models can and cannot do. It is therefore of utmost importance, now that this technology has the attention and resources it needs to be studied, to find its most suitable use cases.

At the same time, software systems are becoming more and more complex, with some applications reaching millions if not billions of lines of code and tech companies hiring thousands of employees. To manage these software projects, whether large or small, teams need to have an adequate set of practices that they can rely on to organise their work. To help handle these practices, Essence comes into play, creating a common ground for teams to communicate effectively.

The objective of this thesis is to find a way to make use of the new advancements in natural language processing, including large language models, but also information retrieval systems, to promote the adoption of the Essence standard across industry and academia. In particular, it aims to answer these two research questions:

RQ1: How can a system that leverages large language models integrate and retrieve domain knowledge about Essence?

RQ2: How effective is this new system in providing information related to Essence? In particular, how does it compare to other general-purpose systems?

To address these questions, I developed a chatbot designed to provide answers about the Essence standard. This application is capable of retrieving relevant information from selected documents and generating responses based on the given prompt and context.

The thesis starts by explaining the architecture of large language models (LLMs) and their optimisation techniques in Chapter 1. Chapter 2 focuses on the Essence standard, its components, and its applications. Chapter 3 reviews existing literature on LLMs in software engineering, identifying research gaps. Chapter 4 outlines the methodology, including the chatbot's design and implementation phases. Chapter 5 presents experimental results, while Chapter 6 discusses the findings, limitations, and future research directions.

Chapter 1

Large Language Models

This chapter aims to outline and briefly explain some key concepts that are necessary for a full understanding of the following sections of this thesis. It will provide an overview of large language models, explaining their role in natural language processing, their functionality and how they can be optimised.

1.1 Definition of Language Model

A language model is a computational framework designed to understand, generate, and predict sequences of text. At its core, a language model assigns probabilities to sequences of words, determining the likelihood of a given sequence appearing in a language. This predictive capability allows language models to perform a range of tasks in natural language processing (NLP), from sentence completion and text generation to translation and summarisation [1].

Figure 1.1 highlights four major stages in language models development: statistical models (e.g., n-grams), neural models (e.g., RNNs, LSTMs), contextualised embeddings (e.g., BERT, ELMo), and large-scale pretrained models (e.g., GPT-4). In this

thesis, we focus solely on large language models (LLMs).

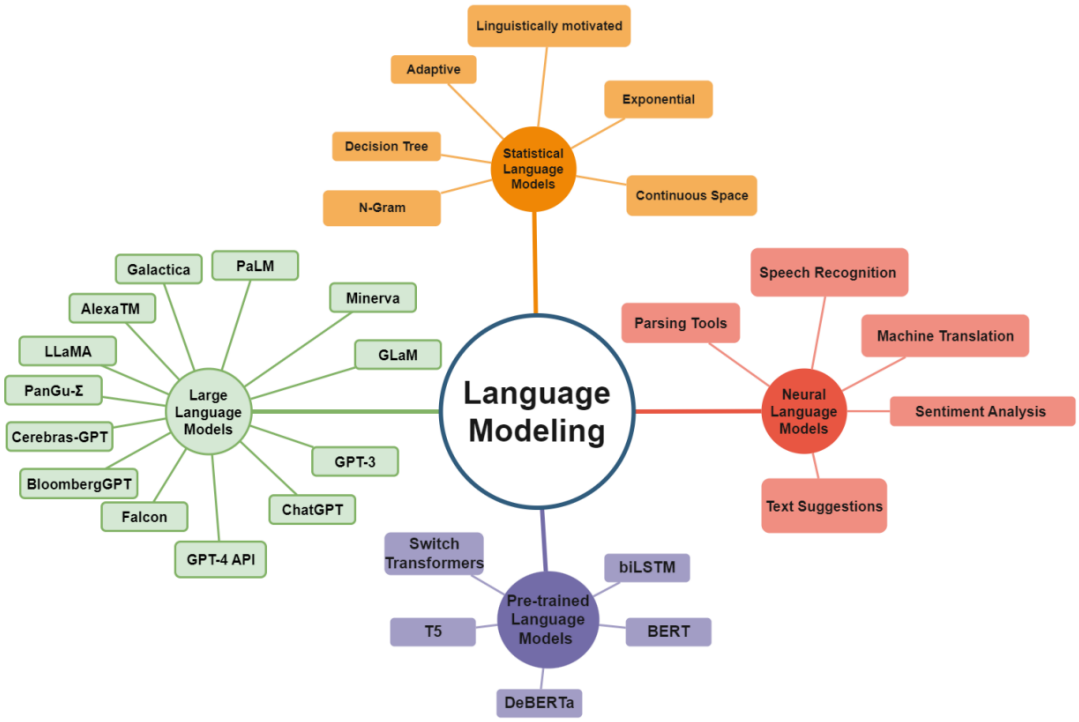


Figure 1.1: Types of language modeling (source: [2]).

Large language models are an advanced class of these systems, characterised by their scale in terms of parameters and training data. By leveraging more sophisticated architectures, such as transformers, and vast datasets, LLMs have shown remarkable potential in understanding and generating text that is increasingly similar to human language. They are revolutionising fields like conversational AI, machine translation, and content creation by enabling machines to process and interpret language with human-level nuance. [3]

1.2 How Large Language Models work

This section of the chapter aims to give a general overview of the key components of large language models and their functionality. The literature presents a variety of approaches to the separation of the system components. For the sake of clarity, I'm following the separation suggested by Minaee et al. [4], which is also summarised in Figure 1.2.

1.2.1 Data Cleaning

The training process of large language models begins with extensive data collection. Since LLMs rely on processing textual input as numerical representations, acquiring diverse and high-quality data is critical. This data can be in a variety of formats, such as books, websites, articles, code repositories, and multimodal content like images and audio. The goal is to provide the model with a comprehensive understanding of human language and other domains.

Before being fed into the model, the data is subjected to a meticulous preprocessing phase to enhance its quality and usability. This process includes noise removal (e.g., filtering out irrelevant or incorrect data), quality filtering (to retain only meaningful information), deduplication (to eliminate repetitive content), and privacy reduction (to remove sensitive or personally identifiable information) [2].

1.2.2 Tokenization

Tokenization is the process of breaking down input text into smaller units, typically words, subwords, or characters, that the model can process. Subword tokenization methods such as Byte Pair Encoding (BPE) and WordPiece are commonly used in LLMs to balance vocabulary size and representational efficiency. These methods split rare or unknown words into subword units, allowing the model to handle

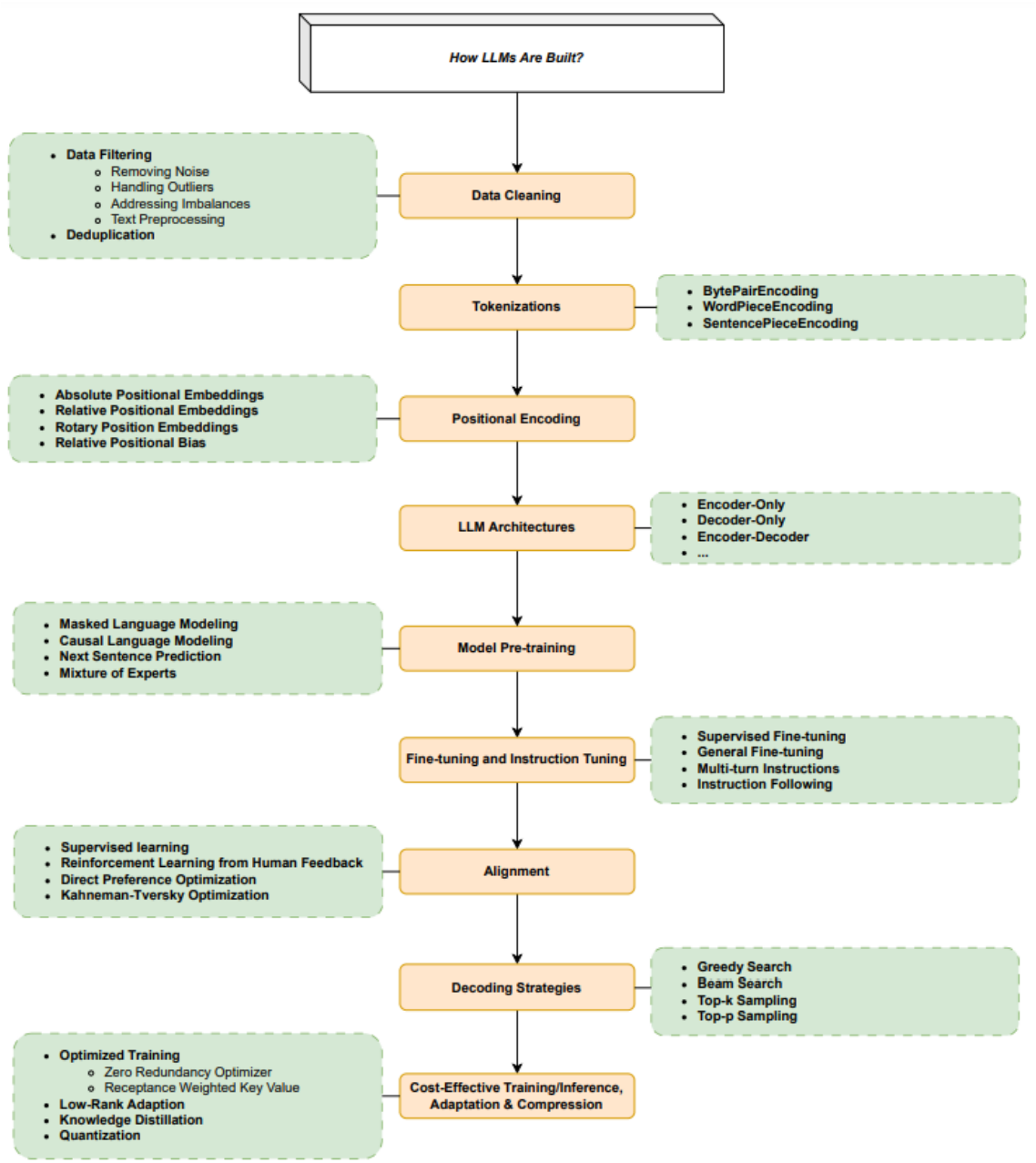


Figure 1.2: Different components of LLMs (source: [4]).

different languages and domains [5].

For example, the word "unbelievable" might be tokenized into ["un", "believe", "able"], allowing the model to leverage its knowledge of common subwords like "believe" and "able".

Figure 1.3 depicts the steps involved in the data cleaning and tokenization processes that we have discussed so far.

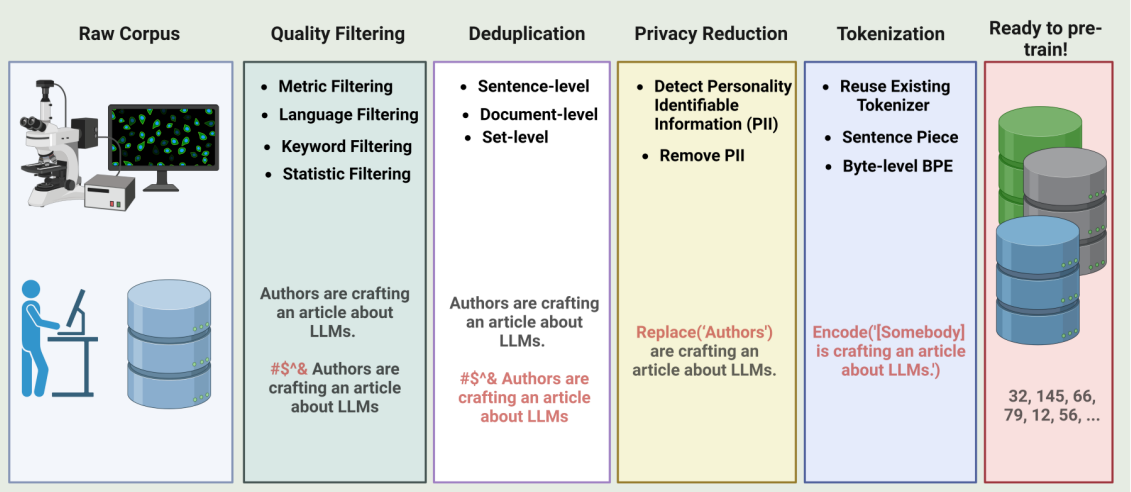


Figure 1.3: The illustration of a data processing pipeline for pre-training LLMs (source: [2]).

1.2.3 Embedding and Positional Encoding

Once tokenized, the text is converted into numerical vectors through a process called embedding. Embedding maps discrete tokens into continuous vector spaces, capturing semantic relationships between words. Words with similar meanings are represented by vectors that are close together in the embedding space [6].

Embeddings are typically initialised with pre-trained weights, such as Word2Vec or GloVe, and further fine-tuned during the training of the transformer model. This

process ensures that the model develops a nuanced understanding of the relationships between words in context [3].

Transformers process sequences in parallel, but this parallelism introduces a challenge: the model lacks inherent knowledge of the order of tokens. Positional encoding addresses this by adding position-specific information to the embeddings, allowing the model to understand the relative positions of tokens in a sequence [3].

Positional encodings are often implemented using sinusoidal functions that generate unique values for each position. These values are added to the token embeddings before being fed into the transformer so that the models can differentiate between phrases like "The cat chased the mouse" and "The mouse chased the cat" [7].

1.2.4 Variants of Transformer Model Architectures

LLMs can be categorised into three primary structures, each tailored for specific tasks [8]:

- **Encoder-Only Models**
  Encoder-only models, such as BERT (Bidirectional Encoder Representations from Transformers), focus on understanding input text by generating a rich, contextualised representation of each token. These models are bidirectional, meaning they analyse context from both preceding and succeeding tokens, making them ideal for tasks like text classification, sentiment analysis, and question answering.

- **Decoder-Only Models**
  Decoder-only models, exemplified by GPT (Generative Pre-trained Transformer) and Llama (Large Language Model Meta AI), specialise in text generation. They process input text auto-regressively, predicting the next token based on previously generated tokens. This unidirectional approach is well-suited for tasks like language generation, code synthesis, and conversational AI.

- **Encoder-Decoder Models**
  Encoder-decoder models, such as T5 (Text-to-Text Transfer Transformer), combine the strengths of both encoders and decoders. The encoder processes the input to create a contextual representation, which the decoder then uses to generate an output. These models are particularly effective for tasks like machine translation, summarisation, and question answering.

1.2.5 Model Pre-training

Pre-training is the core stage where the LLM learns the foundational patterns and relationships within the data. This process leverages a self-supervised learning paradigm, meaning that the model trains itself without labelled data [3]. We could say that it plays a "guess the next word" game, known as language modelling, predicting subsequent tokens in a sequence based on preceding ones.

The underlying architecture for this task is typically a transformer, which employs mechanisms like self-attention to determine the importance of different input tokens relative to each other. Key components such as positional encoding, layer normalization, and activation functions further refine the model's ability to handle sequential data [9].

The training process involves stochastic gradient descent and back-propagation to optimise the model's parameters. When the model predicts a token incorrectly, back-propagation adjusts its internal parameters to reduce future errors. This iterative adjustment allows the model to become proficient at predicting patterns in the data, capturing both syntax and semantics [10].

The next step after pre-training is alignment, a phase where the model is fine-tuned to align its behaviour with human values and task-specific requirements. This can be achieved, for example, through supervised learning, where the model is trained on curated examples of correct responses, or through reinforcement learning with human feedback (RLHF). In RLHF, human evaluators rank model outputs, and

these rankings are used to train the model to produce higher-quality responses [11].

As the LLM processes more data during pre-training, it begins to discern higher-level patterns and concepts, enabling it to perform increasingly complex tasks.

1.2.6 Decoding Strategies

The ability of LLMs to generate coherent and contextually appropriate text relies on their advanced inference mechanisms. During inference, the model processes input tokens and predicts the next most likely token, continuing iteratively until a stopping criterion is reached. This process is powered by the attention mechanism, allowing the model to dynamically focus on relevant parts of the input while processing all tokens in parallel. The final output is determined through a softmax layer, which transforms logits, the raw output values, into probabilities [12].

To improve output quality, LLMs use decoding strategies like beam search, which evaluates multiple candidate sequences for the most plausible result, and greedy decoding, which prioritises high-probability tokens but may compromise coherence. The context window size also influences performance by determining how much preceding text the model considers during generation [13].

Techniques such as temperature, top-k sampling, and nucleus sampling offer control over the style and randomness of outputs. Temperature adjusts the variability of token selection, while top-k sampling limits choices to the most probable tokens, and nucleus sampling dynamically selects from tokens meeting a cumulative probability threshold [4].

Figure 1.4 illustrates the original transformer model structure introduced by Vaswani et al. in 2017 [14]. Models like GPT utilise the transformer decoder architecture, depicted on the right side of Figure 1.4. In these models, the decoder operates independently without an encoder, leading to the removal of Multi-Head Attention and Layer Norm components that connect to the encoder. Unlike GPT, which adopts

the transformer decoder structure, models such as BERT employ the transformer encoder architecture, represented on the left side of Figure 1.4.

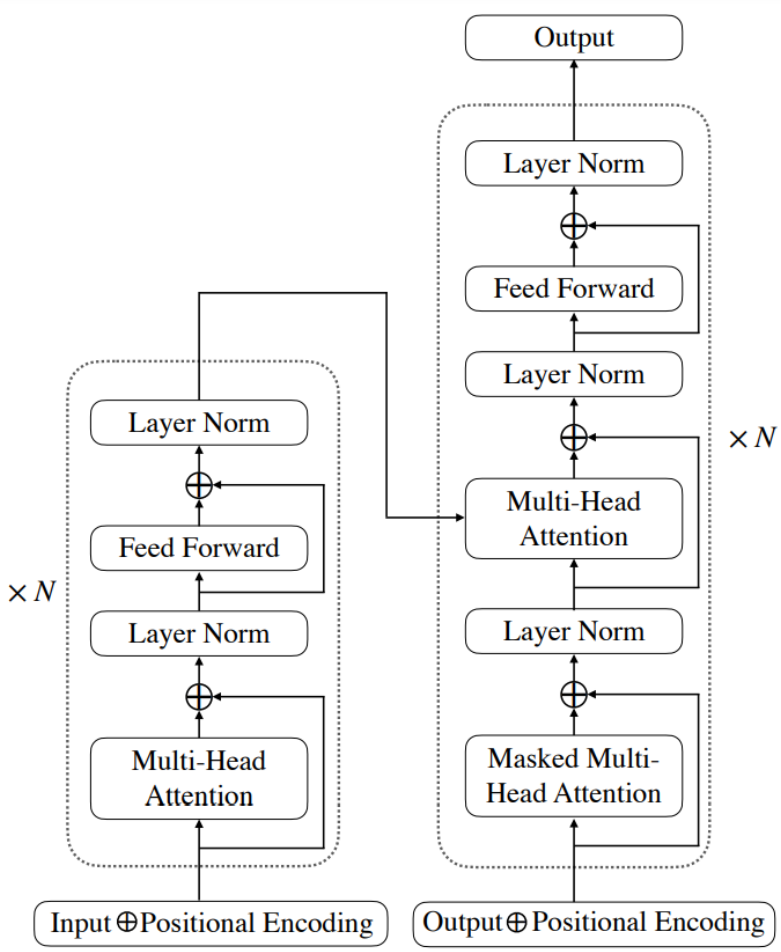


Figure 1.4: Transformer model structure with N encoder blocks (on the left) and N decoder blocks (on the right) (source: [15]).

1.3 Model's Performance Optimisation

There are several advanced techniques for optimising large language models. Fine-tuning enhances model performance by retraining pre-trained models on specialised datasets. Retrieval-Augmented Generation (RAG) combines knowledge retrieval and generation for more accurate outputs. Prompt engineering focuses on writing effective inputs to guide model behaviour, improving performance across various tasks.

Gao et al. [16] created this graph (Fig. 1.5) to compare model optimisation methods based on two factors: "External Knowledge Required" and "Model Adaptation Required". Prompt Engineering demands minimal changes to the model and external knowledge, leveraging the inherent capabilities of LLMs. Fine-tuning, however, involves additional model training. In the early phase of RAG (Naive RAG), model modifications are minimal, but as research advances, Modular RAG has become more integrated with fine-tuning techniques.

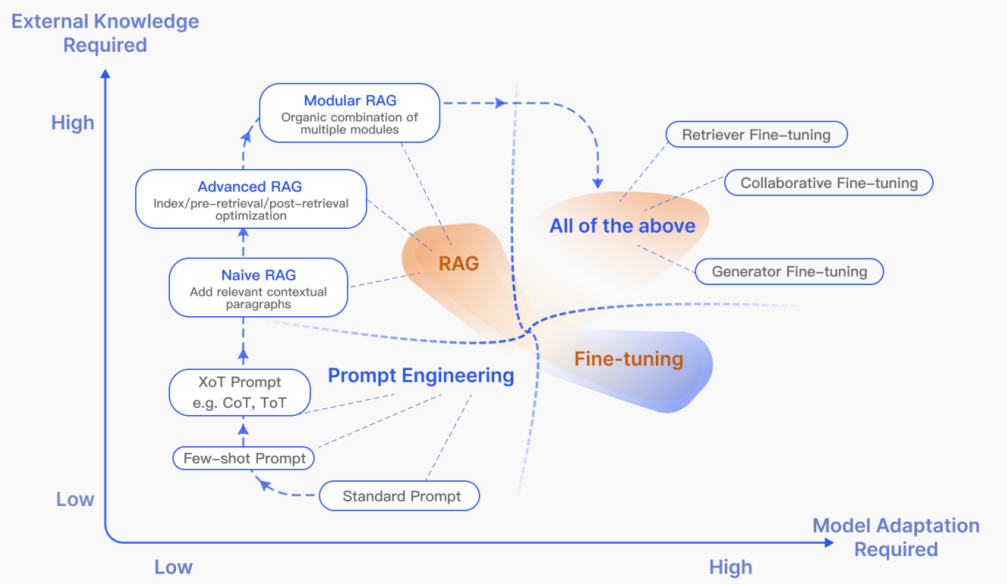


Figure 1.5: Comparison of model optimisation methods in the aspects of "External Knowledge Required" and "Model Adaption Required" (source: [16]).

1.3.1 Fine-tuning

Fine-tuning is a critical step in enhancing the performance of large language models by tailoring their capabilities to specific domains or tasks. It involves retraining a pre-trained model on a specialised dataset, starting from the weights learned during pre-training.

Fine-tuning can be categorised into:

1. **Full Fine-Tuning:** The entire model, including all its parameters, is updated using a labelled dataset. This method is computationally expensive and requires significant resources but results in a highly specialised model [3].
2. **Parameter-Efficient Fine-Tuning:** Techniques like LoRA (Low-Rank Adaptation) and adapters allow only a subset of the model's parameters to be updated, significantly reducing resource requirements while maintaining performance [17].

Moreover, there are more specialised approaches tailored to specific tasks, such as Instruction Tuning. In this approach, the model is fine-tuned using instruction-based data, where input-output pairs are designed to teach the model how to follow commands and complete specific tasks effectively [18].

The fine-tuning process often includes supervised learning to optimise performance for tasks like sentiment analysis, summarisation, or domain-specific text generation. The quality and size of the labelled dataset play a crucial role in determining the success of fine-tuning.

1.3.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation integrates external knowledge retrieval with the text generation capabilities of LLMs, improving their performance and applicability.

This approach is particularly effective in scenarios where the model's pre-trained knowledge is outdated or insufficient for a specific query [16].

The RAG process involves two main components:

1. **Knowledge Retrieval:** When given an input, the system retrieves relevant documents or information from an external database or corpus. This step ensures that the model has access to up-to-date or domain-specific knowledge beyond its pre-training.
2. **Contextual Generation:** The retrieved information is provided as additional input to the LLM, guiding its generation process. This allows the model to produce contextually accurate and relevant outputs while reducing hallucinations (instances where the model generates incorrect or fabricated information) [19].

These steps can be seen in this diagram from Gao et al. [16] (Fig. 1.6).

RAG systems are often implemented using vector stores to store and retrieve embeddings of textual data efficiently. By combining retrieval with generation, RAG systems can connect the static knowledge in pre-trained models with dynamic, real-world information.

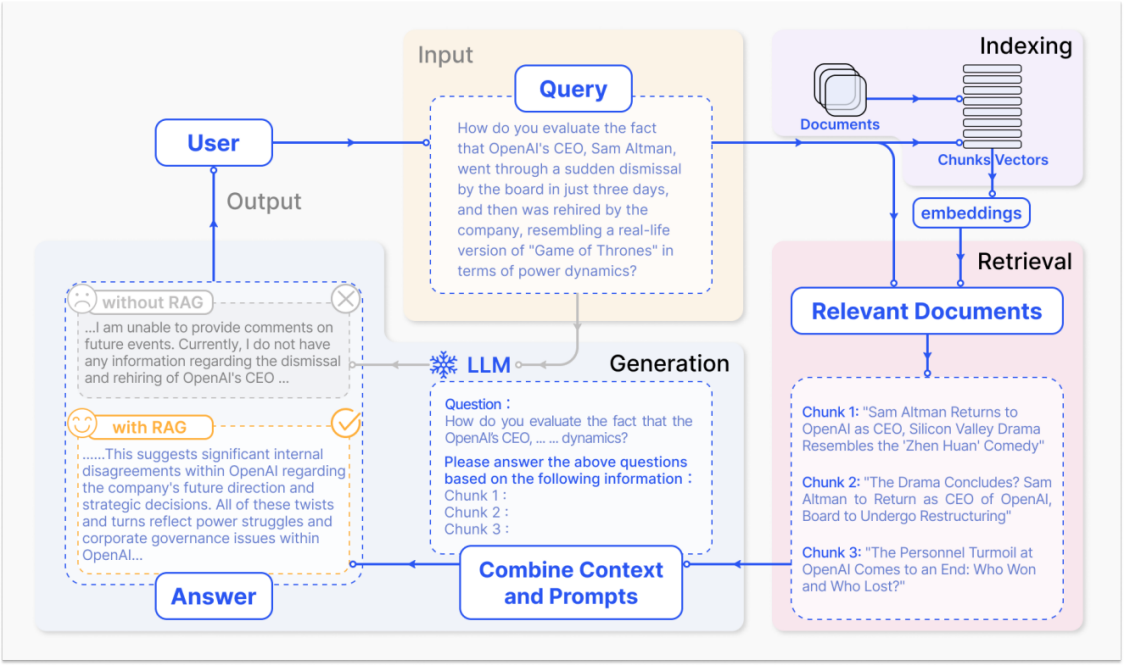


Figure 1.6: Representative instance of the RAG process applied to question answering (source: [16]).

1.3.3 Prompt Engineering

Prompt engineering is the art of crafting effective input prompts to guide an LLM's output. Since LLMs generate responses based on the context provided in the prompt, the quality and structure of the prompt significantly influence the results. Prompt engineering is a lightweight and cost-effective method to optimise a model's performance without requiring additional training or fine-tuning.

Naveed et al. [3] present different strategies in prompt engineering, including:

1. **Zero-shot Prompting:** In this approach, the prompt directly asks the model to perform a task without providing examples. For instance, "Summarise the following article:" relies on the model's inherent capabilities to understand and

execute the task.

2. **Few-shot Prompting:** The prompt includes a few examples of the desired input-output pairs before presenting the actual task. This helps the model infer the task's requirements from the examples.

3. **Chain-of-Thought Prompting:** For complex reasoning tasks, the prompt can include intermediate reasoning steps, guiding the model to break down the problem into smaller components. This approach improves logical consistency and accuracy in outputs.

Chapter 2

Essence

Similarly to the previous chapter, this chapter provides background information to support the understanding of this thesis. It introduces the Essence standard, detailing its kernel, language, and applications in both industry and academia. Unless otherwise stated, all information in this chapter is sourced from the book "The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!" [20].

2.1 Definition and purpose of the Essence standard

The Essence standard, developed by a community of software engineering experts called SEMAT and maintained by the Object Management Group (OMG), is a framework for improving software engineering practices by offering a common ground for software development.

It addresses the complexities of modern software development by enabling teams to define, adapt, and combine practices effectively. By promoting a shared language and kernel, Essence simplifies communication across teams, supports flexibility in

methods, and encourages continuous improvement in software engineering.

2.2 Essence Language

At its foundation, Essence consists of two integral components: the Essence Kernel and the Essence Language.

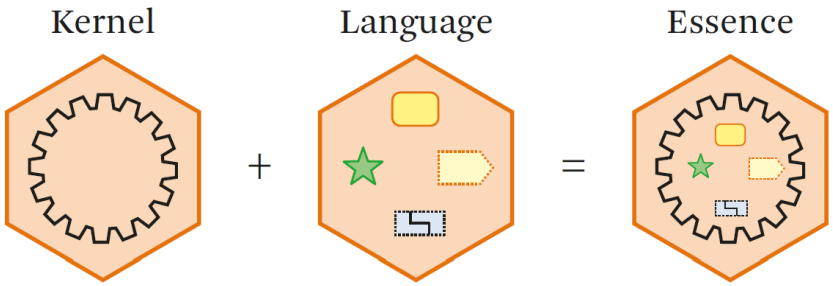


Figure 2.1: Essence and its parts (source: [20]).

The Essence Language is a tool for expressing practices in a simple and visual manner. This language enables practitioners to create modular, reusable practices that can integrate seamlessly into different workflows. The kernel and language together empower teams to assess project progress and identify areas for improvement while maintaining flexibility in their chosen practices.

2.2.1 Key Elements of the Language

The Essence Language provides a structured framework to describe practices, repeatable approaches aimed at achieving specific purposes, and methods, combination of practices.

The key elements of the Essence Language include:

1. **Alphas:** Alphas are critical elements representing progress in a software endeavour. For instance, a *"Team"* achieving a *"Performing"* state or a *"User Story"* progressing to *"Done"*. Alphas ensure focus remains on outcomes rather than secondary concerns like documentation. They move through Alpha States, which are milestones defined by Checklist Items, actionable tasks to evaluate progress.

2. **Work Products:** Work Products are tangible artifacts, such as a Kanban board or a user story card. They have defined Levels of Detail, which describe the depth and breadth of the information captured.

3. **Competencies:** Competencies define the skills and capabilities necessary for a role, such as Development, Leadership, or Testing. Each competency includes multiple Competency Levels, from assisting (Level 1) to innovating (Level 5).

4. **Activities:** Activities are actions undertaken to achieve specific outcomes, like refining a backlog or conducting a daily stand-up. Each activity outlines the expected outcomes, required competencies, and guidance for successful execution.

5. **Patterns:** Patterns offer flexible guidance and reusable solutions. Examples include milestones defined by Alpha States, techniques like Brainstorming or roles such as Scrum Master.

6. **Resources:** Practices reference supporting materials, allowing practitioners to access detailed guidance or supplementary information.

Each of these components interconnects to "tell the story" of how the practice achieves valuable outcomes.

2.3 Essence Kernel

The Essence Kernel is the foundation layer of the standard. It contains core elements that are essential to any software development project. These elements include Alphas, Activity Spaces and Competencies, which are discussed in more detail in the following sections of this chapter.

2.3.1 Areas of Concern

The Kernel is organised into three Areas of Concern, the main categories to which each element in a practice belongs or is related:

- **Customer:** Addresses customer needs, including identifying opportunities and engaging with stakeholders.
- **Solution:** Focuses on defining and delivering a solution, including requirements and the software system itself.
- **Endeavour:** Covers the team, their activities, and their way of working.

Each area contains Kernel Alphas, which represent core concepts, such as “Stakeholders” or “Team”, to monitor the endeavour’s health and progress.

2.3.2 Kernel Alphas

The Kernel Alphas in Essence are the core, universal elements of any software endeavour, representing the critical aspects that need to progress for success. These include seven key Alphas organised into the three areas of concern. Each Alpha moves through defined states, such as a “Requirement” transitioning from “Proposed” to “Satisfied”, with associated checklist items to track progress.

The Kernel Alphas include:

- **Opportunity:** Opportunity is the shared understanding of stakeholders' needs, justifying and shaping requirements for developing or changing software systems.
- **Stakeholders:** Stakeholders are individuals or groups affecting or affected by a software system, providing requirements, funding, and involvement to ensure its success.
- **Requirements:** Requirements define what the software system must do to address the opportunity, satisfy stakeholders, and guide development and testing.
- **Software System:** A software system integrates software, hardware, and data to deliver value, often as part of broader business or social solutions.
- **Work:** Work encompasses all activities undertaken by the team to create a software system that meets requirements and stakeholder opportunities, guided by established practices.
- **Team:** A team is a group responsible for planning and executing tasks to develop, maintain, deliver, or retire a software system.
- **Way of Working:** The way of working is the team's evolving set of practices and tools, continuously adapted to their mission and context.

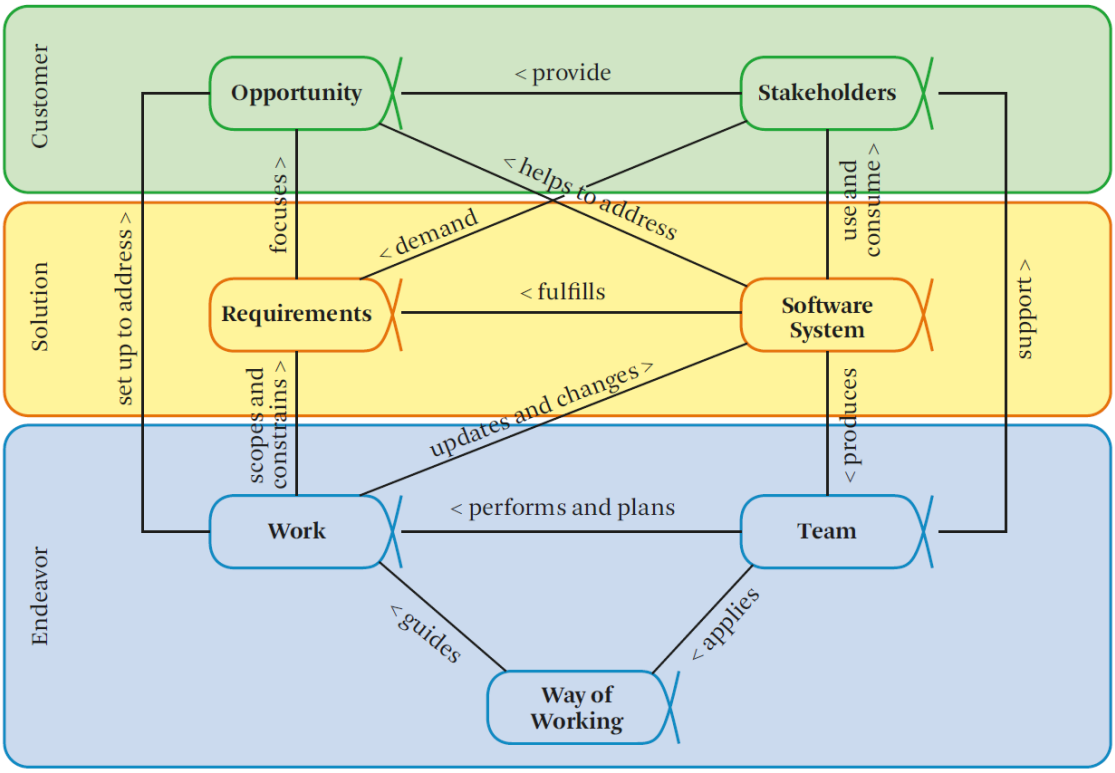


Figure 2.2: The Essence alphas and their relationships (source: [20]).

2.3.3 Kernel Activity Spaces

The Kernel Activity Spaces define the high-level tasks that must be addressed during a software project, such as "Understand the Requirements" or "Build the System". These spaces act as placeholders for the activities within practices, offering a high-level view of what needs to be done. They can be used independently to evaluate existing workflows or integrated with practices to clarify the scope and purpose of specific activities. For instance, the activity "Conduct Daily Stand-Up" might align with the "Coordinate Activity" Activity Space. By mapping activities to these spaces, teams gain a comprehensive understanding of their progress and ensure they

address all critical aspects of their work.

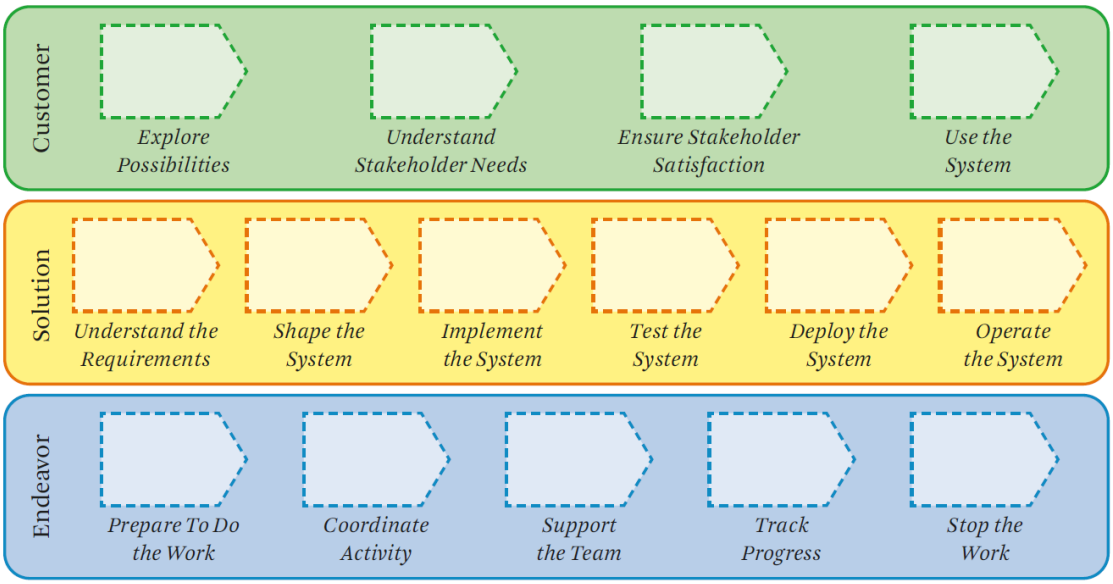


Figure 2.3: Essence activity spaces (source: [20]).

2.3.4 Kernel Competencies

The Kernel Competencies represent the skills, knowledge, and capabilities necessary for successful software engineering endeavours. Essence defines six core competencies that are essential across most teams. These competencies are described across five levels, from basic support (Level 1: Assists) to advanced innovation (Level 5: Innovates). Practices can build upon these core competencies, introducing specialised ones as needed, such as “Coaching” or “Operations”. By focusing on competencies, the Kernel helps teams identify the skills required for specific activities.

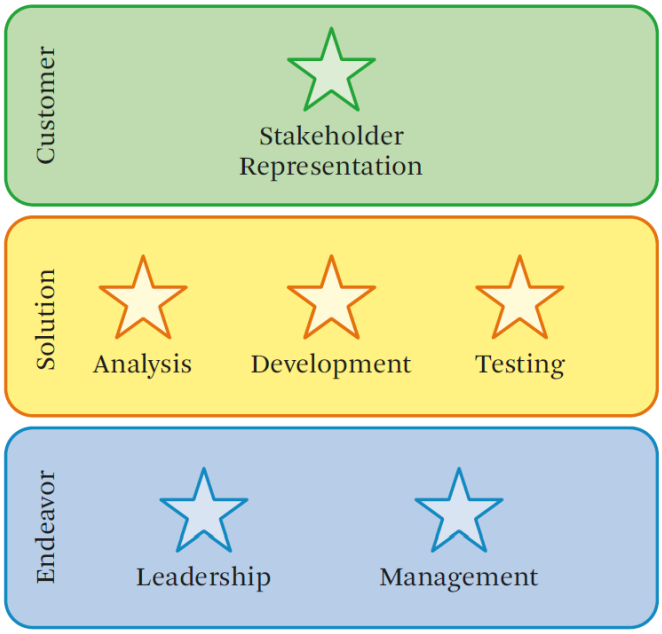


Figure 2.4: The kernel competencies (source: [20]).

2.4 Essentializing a Practice

Essentializing a practice means distilling it into its fundamental components, making it easier to adopt, adapt, and combine with other practices. Practices like Scrum or Kanban can be essentialized into their core activities, alphas, and work products. While it is impractical to essentialize every existing software engineering practice, since there are thousands of them, the shared language provided by Essence enables teams to define and refine their practices collaboratively. This standardisation reduces ambiguity and promotes better decision-making across diverse teams and projects.

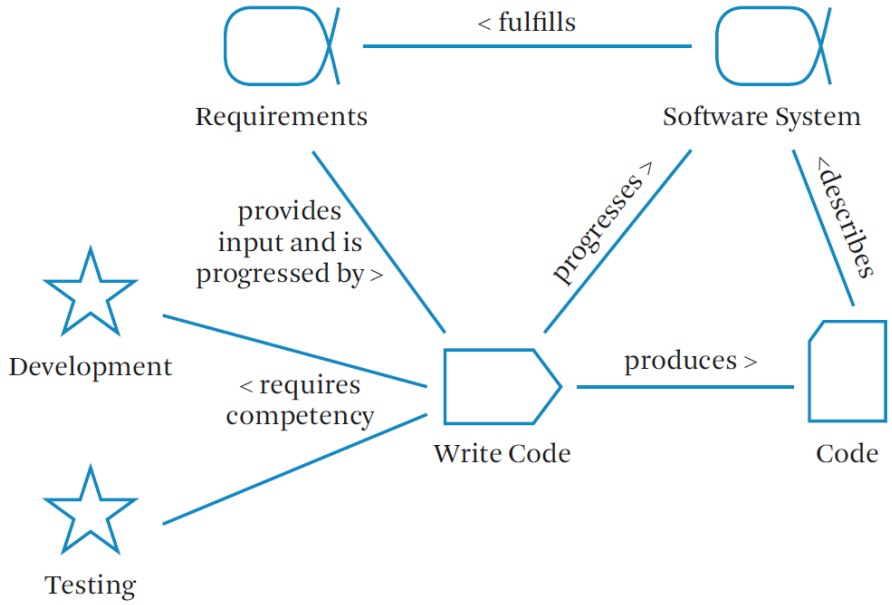


Figure 2.5: Pair programming described using Essence language (source: [20]).

2.5 Essence Games

Essence games use the Essence Cards, physical or digital cards that can represent any Essence element, to improve collaboration and decision-making within software development teams. These games are designed to make abstract concepts tangible, facilitating discussions about progress, health, and objectives in a structured yet engaging way. Each game focuses on specific aspects of the software lifecycle, offering teams a practical approach to improve their methods and outcomes.

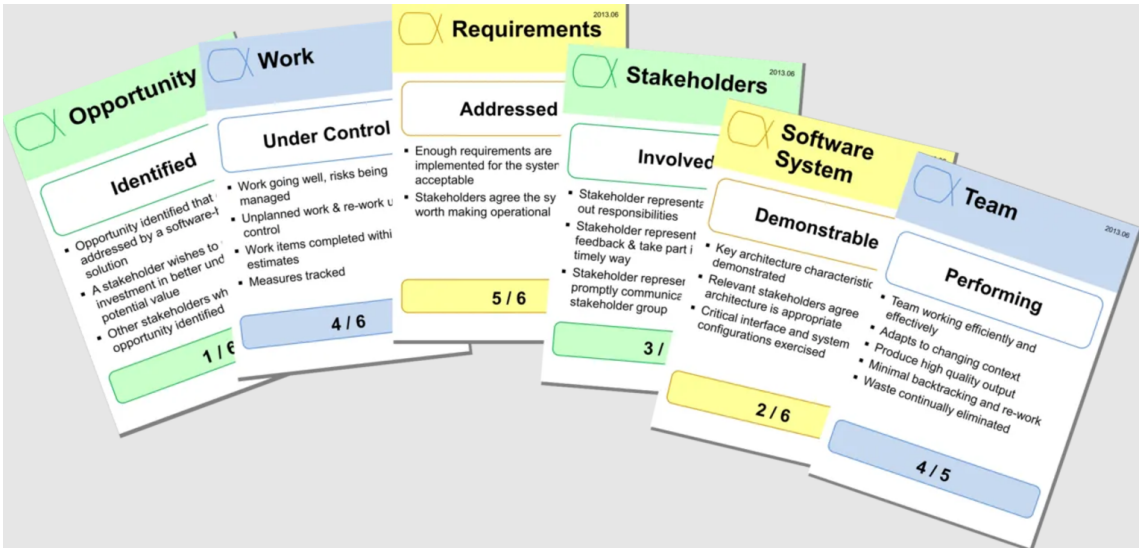


Figure 2.6: Example of Essence cards (source: [20]).

Examples of these games can be:

- **Progress Poker:** Progress Poker is a consensus-driven game where team members independently evaluate the current state of a specific Alpha. By revealing their assessments simultaneously, the team can discuss differing opinions and reach a shared understanding of their progress.
- **Chase the State:** Chase the State is a retrospective activity that prompts teams to evaluate their current position across all Alphas. By methodically reviewing each state, this game encourages a broader perspective on software development health, complementing traditional metrics like burn-down charts.
- **Objective Go:** Objective Go builds upon the insights from Chase the State, helping teams set realistic goals. By identifying the next achievable states for each Alpha, this game ensures that objectives remain balanced and aligned with the team's capabilities and time frames.

Other games, like Checkpoint Construction, Lifecycle Layout, Milestone Mapping, and Health Monitoring, use the Alpha State Cards to define checkpoints, visualise lifecycles, plan milestones, and track project health. Together, these games provide teams with a lightweight toolkit to improve collaboration and planning.

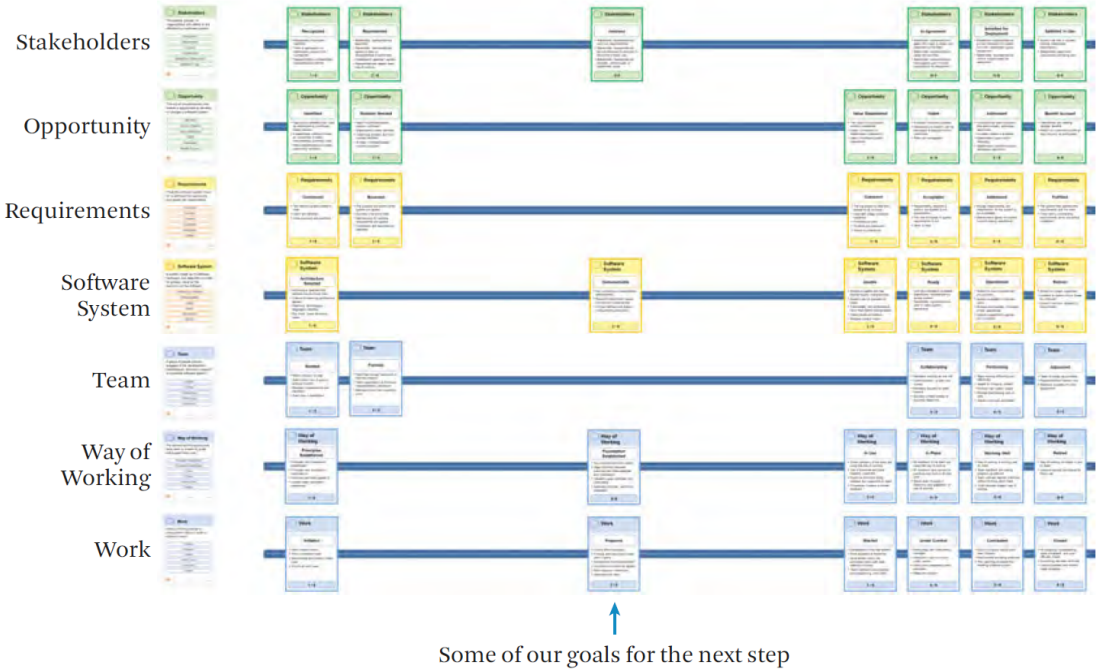


Figure 2.7: Objective Go (source: [20]).

2.6 Applications

The Essence framework offers significant benefits in both industry and academia, allowing organisations and educational institutions to learn, adapt and integrate practices.

2.6.1 Industry

In the software industry, Essence facilitates the integration of various methodologies, allowing organisations to tailor their processes without being locked into a single framework. It supports teams in combining elements from Agile, DevOps, or traditional methodologies to create workflows that address their unique challenges.

Jana & Pal [21] highlight how Essence can be leveraged in large-scale software development, noting its ability to mitigate risks through process health checks and competency assessments, which help ensure alignment with project objectives and agile principles. This approach not only enhances process agility but also supports continuous improvement.

Raharjo et al. [22] further illustrate the practical application of Essence by proposing a model that integrates popular Agile methods, customising practices to fit the specific needs of an organisation. Their work, demonstrated in a national bank in Indonesia, shows how Essence can provide the flexibility needed to adapt Agile methodologies to a diverse set of business contexts while maintaining process coherence and efficiency.

2.6.2 Academia

Academia benefits from Essence by using it as a teaching tool for software engineering concepts. Its modular nature makes it an effective way to introduce students to core principles like requirements analysis, team dynamics, and iterative development. By emphasising essential elements, Essence encourages students to think critically about software engineering practices and how to adapt them to real-world problems.

Ciancarini & Missiroli [23] highlight how Essence can be applied to enhance the teaching of Agile methodologies in software engineering courses. Integrating Essence cards into the curriculum provides students with a structured framework to under-

stand Agile principles, track their progress, and reflect on their learning. This approach not only encourages collaboration, but also provides the flexibility to customise practices, ultimately improving student engagement and performance.

Chapter 3

Literature Review

This chapter provides an overview of the existing literature on the applications of large language models within the field of software engineering. At the beginning of this project, only a limited number of articles were available for review. However, as the study progressed, new research emerged on a monthly basis, with the latest publications included up until October 2024. By synthesising current research, this chapter aims to identify gaps in the literature and highlight potential research directions.

3.1 Applications of LLMs in Software Engineering

The integration of large language models in Software Engineering (SE) has led to a significant change in how software development is conducted. From automating mundane tasks to enhancing productivity and accuracy, LLMs are increasingly becoming crucial tools in modern SE practices. This section explores the current applications of LLMs in SE, drawing insights from the research articles by Hou et al. [24] and Vassilka et al. [25].

1. **Code Generation:** The most significant application of LLMs in software development is code generation. Models like OpenAI's Codex, based on the GPT architecture, are capable of generating snippets of code based on natural language prompts. This feature drastically reduces the time developers spend writing code and facilitates rapid prototyping.

2. **Specification Generation and Requirements Engineering:** LLMs have also found applications in the earlier stages of the software development life-cycle. With their ability to process vast amounts of text and context, they can assist in generating more complete and coherent specifications. Moreover, LLMs can assist in understanding and refining user stories and functional requirements, which are crucial for defining the scope of software projects, especially in Agile methodologies [26].

3. **Test Case Generation:** Generating test cases, especially unit tests, is a critical yet often neglected task in software development. Many developers tend to bypass or shortcut the creation of comprehensive test cases, which can lead to issues in system quality and maintenance. LLMs have shown significant potential in automating the generation of test cases, from unit tests to integration tests, ensuring better test coverage and quality.

4. **Documentation:** Keeping documentation up-to-date with changes in the codebase can be time-consuming. LLMs can be to used to automatically generate or update documentation. They can also summarise complex codebases, making them more accessible to new developers or contributors.

5. **Collaboration and Communication:** LLMs also aid in improving communication within software engineering teams. They can assist in generating summaries of meetings or technical discussions, helping team members who may have missed certain conversations stay informed. Furthermore, LLMs can help automate the creation of reports or status updates, streamlining communication processes within teams or between stakeholders.

3.2 Methodologies

The methodologies of the three studies discussed in this section illustrate different approaches to the introduction of LLMs in software engineering.

The first study, conducted by Lin et al. [27], presents FlowGen, an agent-based model for code generation that emulates software process models using LLMs. It defines roles such as Requirement Engineer, Architect, Developer, and Tester, each of which is responsible for a core software engineering activity. These roles are then represented by LLM agents that interact with each other based on specific software development models, including Waterfall, Test-Driven Development, and Scrum. The study emphasises the iterative interaction between agents, allowing for self-refinement, where the agents review and improve the artifacts they generate, such as requirements, design documents, code, and tests. The methodology also integrates testing throughout the process. The evaluation of FlowGen relies on the use of GPT-3.5 to generate code, which is tested using established benchmarks like HumanEval and MBPP. The results are measured with the Pass@1 metric, ensuring that the generated code is both correct and practical.

In the second study, Khojah et al. [28] take a more observational approach to understanding how software engineers interact with ChatGPT in their daily work. Participants from 10 European organisations were involved in the study, where they interacted with ChatGPT over five business days. The researchers collected 130 dialogues and categorised them into three types of interactions: Artifact Manipulation, Training, and Expert Consultation. These categories helped to illustrate how engineers used ChatGPT for various tasks, like generating code, consulting on software engineering practices, and training. Quantitative analysis was used to track usage patterns and volume, while qualitative analysis, including interpretative phenomenological analysis, was employed to explore user experience and trust. The study also included exit surveys to gain additional insights into how participants perceived ChatGPT's usefulness and effectiveness in supporting their work.

The third study, conducted by Rasnayaka et al. [29], is set in an educational context and it investigates the use of LLMs in a semester-long software engineering project. Students in a course at the National University of Singapore were asked to develop a Static Program Analyser for a custom language. The methodology involved integrating LLMs as optional tools for code generation, allowing students to annotate AI-generated code and track their modifications. An online survey, based on the Unified Theory of Acceptance and Use of Technology (UTAUT), was used to measure factors influencing the adoption of LLMs, such as performance expectancy, effort expectancy, and social influence. The researchers also examined moderating factors like prior experience and coding proficiency. Automated extraction of annotated code submissions, combined with sentiment analysis of survey responses, provided insights into how students interacted with the AI tools and what influenced their usage patterns. This empirical study helps to understand both the behavioural and contextual factors that affect the effectiveness of LLMs in software engineering education.

Together, these studies provide complementary perspectives on the use of LLMs in software engineering. While the FlowGen study explores the potential for LLMs to automate and refine the software development process, the second study offers a closer look at how professionals interact with conversational AI tools in real-world scenarios. Meanwhile, the third study provides insights into how LLMs are perceived and adopted by students, shedding light on factors that influence their use in the learning process.

3.3 Findings

The combined results of the three studies highlight the evolving role of LLMs in software engineering, particularly in improving productivity, learning, and the quality of generated code.

A significant finding from the studies is the notable improvement in code genera-

tion accuracy when LLMs, specifically FlowGen, are applied in emulating software process models. The FlowGen system demonstrated substantial improvements in Pass@1 accuracy, particularly in the Scrum-based model, where it outperformed traditional models like RawGPT. Additionally, the integration of testing, design, and code review activities within FlowGen was found to have a significant positive impact on the reliability and stability of generated code.

In contrast, the studies involving ChatGPT and other LLMs in real-world settings show a mixed but generally positive impact. ChatGPT proved to be highly useful for tasks such as artifact manipulation and expert consultation, with users frequently relying on it for assistance in decision-making, solving problems, and generating software artifacts. These uses reflect the versatile nature of LLMs, as they can handle both routine tasks and more complex queries that require expert-like guidance.

The studies also reveal a strong preference among users for LLMs as tools for learning and decision-making. In the context of software engineering education, students showed a tendency to use AI tools such as GitHub Copilot and ChatGPT for generating initial code structures. While there was a decrease in AI usage over time, particularly after the first project milestone, students still appreciated the efficiency that AI brought to the initial stages of their work. This mirrors findings from the study of professionals, where ChatGPT's usefulness was highlighted in automating repetitive tasks and providing quick, reliable responses to technical questions.

Trust in the tools, while generally positive in most cases, varied depending on user familiarity with the technology and the specific tasks at hand. Some users expressed frustration with the AI's occasional inaccuracies, particularly in situations where high accuracy was crucial. However, trust was largely built through consistent use and successful outcomes.

These studies collectively demonstrate that LLMs have a meaningful impact on software engineering practices, improving code generation accuracy, productivity, and learning experiences. Although, their effectiveness is influenced by factors such as user trust, AI tool reliability, and the context in which they are applied.

3.4 Research Gaps

Despite the promising results highlighted so far, several gaps remain in the current body of research. One notable gap is the limited exploration of how LLMs can be effectively integrated with established software engineering methods and practices. There is a lack of research focusing specifically on how these models can be applied to support the nuanced processes involved with software engineering methodologies. The Essence standard remains largely unexplored in terms of its interaction with LLMs.

Another gap lies in understanding the long-term effectiveness and adaptability of LLMs in software engineering education. While studies have explored the use of AI tools like ChatGPT in academic settings, the research has primarily focused on their short-term utility in specific tasks like code generation or problem-solving. There is a need for more comprehensive studies that investigate how LLMs can be integrated into the broader curriculum of software engineering education.

Lastly, the role of trust, ethical considerations, and the potential impact of LLMs on professional practices and educational standards remains insufficiently explored, particularly in the context of their integration into regulated industries or educational institutions.

These gaps present significant opportunities for future research to guide the application of LLMs in both software engineering practice and education.

Chapter 4

Methodology

This chapter explains how Essence Coach, the chatbot designed to support the learning of software engineering practices, was created, from the early stages of the design process to the final implementation.

4.1 Design

4.1.1 Initial Idea

The development of Essence Coach was inspired by the desire to encourage the use of the Essence standard in software development and software engineering education. The objective was to explore how a chatbot could improve the use cases of Essence, which were highlighted in Chapter 2. Given that Essence has multiple use cases, the chatbot could have served a wide range of purposes. However, for the scope of this thesis, the focus was narrowed down to some specific use cases that would be most suited for this kind of technology, such as providing general knowledge, summarising information, and translating content.

The initial concept was to create a chatbot specialised in answering questions related to the Essence standard. This chatbot would function as a virtual assistant, supporting anyone who wants to understand or apply the Essence framework. By using the chatbot, users could improve the way they learn and apply software engineering practices.

4.1.2 Target Audience

The chatbot is designed to serve a broad range of users, particularly those involved in software engineering and process management. The primary target audience includes:

- **Software Engineering Students:** The chatbot serves as an educational tool, helping students in understanding the Essence framework and related software engineering practices.
- **Development Teams:** New employees can use Essence Coach to familiarise themselves with company-specific practices and methods. The chatbot can help them quickly grasp the essential elements of the practices and its application within the organisation.
- **Methodologists and Managers:** Professionals who wish to translate their existing methods into the Essence language. This could be particularly useful for methodologists looking to align or adapt their approaches to meet the Essence standard, as well as managers overseeing the implementation of new practices.

4.1.3 Use Cases

The primary goals of the chatbot are centred around helping users understand and apply the Essence framework in various software engineering contexts. These goals

are translated into the following main use cases:

1. **Learning Support:** The chatbot assists in the learning process of software engineering by providing information on practices and methods and on the Essence kernel and language. It serves as a guide for students and new employees looking to understand the fundamentals and applications of Essence.
2. **Decision-Making Assistance:** The chatbot assists teams in making informed decisions about which methods and practices to adopt. Whether starting from scratch, modifying existing practices, or scaling up processes, the chatbot can provide guidance in selecting the most appropriate practices and activities tailored to the needs of the team or organisation.
3. **Translation of Practices:** Methodologists and managers can use the chatbot to translate written practices into the Essence language. The chatbot can help define the Alphas, the Activities, and all the other elements needed to describe the practice.

4.2 Implementation

4.2.1 Architecture

The main components of the system architecture are the LLM model, which generates the response to the user's prompt, the RAG system, that retrieves contextual information relevant to the question, and the user interface, which the user can interact with.

The back-end provides an API endpoint for handling user queries. Upon receiving a question, the system uses the ensemble retriever module to retrieve relevant context from markdown files stored in the database. This retriever combines an advanced keyword search and a vector search to find the most appropriate contextual

information.

The processing pipeline starts with generating the augmented prompt by appending the retrieved context to the user query. This prompt is sent to the Groq API, where the chosen LLM, in this case Llama 3, processes the input based on the system-defined prompt, which ensures the bot adheres to its purpose as an "Essence coach". The chatbot maintains a chat history, truncating it when token limits are exceeded.

Finally, the response is returned to the user and its content, along with metadata such as the context and model details, is stored in a database for future analysis.

Figure 4.1 depicts an overview of the system's architecture.

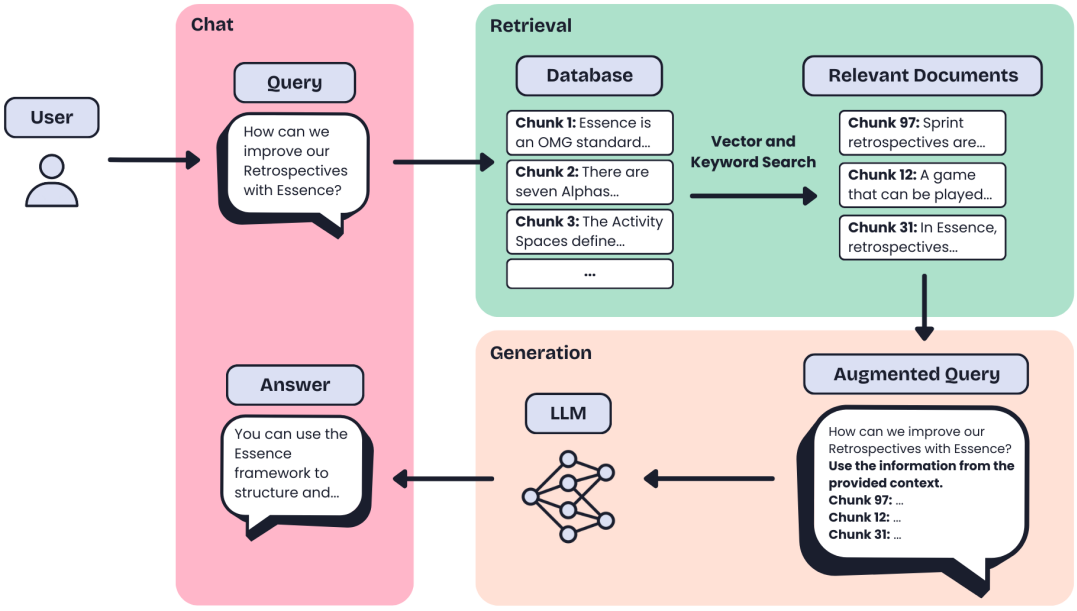


Figure 4.1: Overview of the system's architecture.

The following subsections explain how the various components work and the thought process behind the design choices.

Generation

To generate responses to user queries, the architecture leverages a pre-trained, general-purpose large language model. The selected model is Llama-3.1-70b, a state-of-the-art model trained on 70 billion parameters.

After testing different models and comparing their capabilities, Llama 3 was chosen for the final implementation of the chatbot because of its superior contextual understanding, ability to generate coherent and domain-relevant responses, and robust performance across different kinds of queries.

The LLM is accessed via the Groq API, which provides an efficient interface for using advanced language models without requiring significant computational resources locally. The Groq API simplifies the integration of pre-trained LLMs into applications, handling tasks like tokenization, inference, and response generation. This API was chosen for its reliability and ease of use. Moreover, Groq offers flexible parameter customisation, such as temperature settings and token limits, and system prompts, allowing for a small fine-tuning of the model.

When customising the model parameters, I experimented with various combinations of temperature settings, system prompts, and token limits to optimise performance. Eventually, I chose a temperature of 0.7 to balance creative and practical responses. The system prompt gave the chatbot its role of Essence expert, provided a brief overview of the Essence standard and instructed the model to combine the retrieved context with its own knowledge.

Despite its strengths, the Llama 3 model has certain limitations, primarily the restricted context window imposed by the Groq API. This constraint limits the amount of information the model can process simultaneously, necessitating strategies to manage context effectively. To address this, I implemented a mechanism to truncate older chat messages while maintaining the most relevant ones, ensuring the chatbot remains within the token limit without losing important context.

The decision to use a general-purpose pre-trained LLM instead of a fine-tuned model was largely influenced by computational constraints. Fine-tuning requires significant hardware resources and time, which were not feasible within the scope of the thesis.

Retrieval

The Llama 3 model is integrated with a RAG system that provides a reliable mechanism for contextual response generation [30]. When a user submits a query, the RAG system first processes it to identify relevant context. This involves searching the pre-processed database of documents for matches related to the query. The retrieved context is then appended to the user's query before being passed to the LLM. This ensures that the model has access to relevant background information, improving the quality of the generated response.

To ensure efficient retrieval, all documents were converted into markdown format and manually reviewed for consistency. Using a text-splitting approach, I divided the content at each header, to preserve logical units of information rather than relying on arbitrary character limits. This semantic chunking method ensures that each chunk represents a cohesive section of information, facilitating meaningful retrieval and improving the relevance of the context provided to the model. Figure 4.2 shows the process of converting the original documents, coming from various sources, into a unified structure consisting of standardised chunks and their corresponding embeddings.

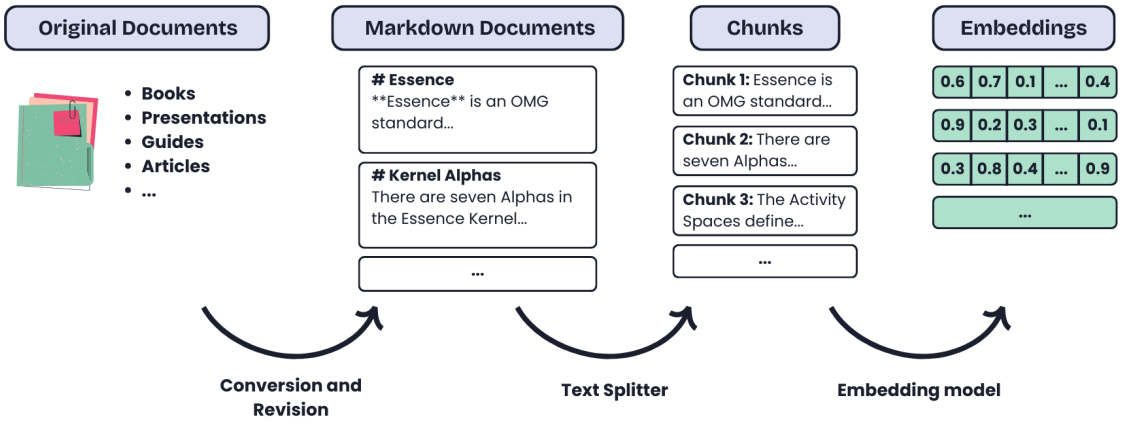


Figure 4.2: Data preprocessing, chunking and embedding.

Determining the optimal chunk size required some experimentation. Smaller chunks improve granularity and retrieval precision but risk fragmenting context, while larger chunks preserve context but may dilute relevance [31]. By balancing these factors, I identified an optimal chunk size that preserves logical units while maintaining sufficient granularity, resulting in a total of 461 chunks from 22 different documents.

The RAG system employs an ensemble retriever that uses both vector-based and keyword-based search methods. The ensemble retriever combines the strengths of both search methods while mitigating their weaknesses [32]. Vector search excels in capturing semantic meaning but may overlook exact matches, while BM25 is highly effective at identifying keyword-based matches but lacks semantic understanding.

For vector search, cosine similarity was used to find the chunks that were most similar to the query [33]. This required embedding the query and the database into 384-dimensional vectors using the all-MiniLM-L6-v2 embedding model. Every embedding was assigned a score (the higher the score the more similar it was to the query) and the top two results with the highest cosine similarity scores were then retrieved and converted back to text.

The cosine similarity is calculated as follows:

$$\text{cosine similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Where:

- $\mathbf{A}$ = the query vector.
- $\mathbf{B}$ = a document chunk vector.
- $\mathbf{A} \cdot \mathbf{B}$ = dot product of the two vectors.
- $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ = magnitudes (norms) of the vectors.

For keyword search, I employed the BM25 algorithm, which retrieves results based on term frequency and relevance. BM25 calculates a relevance score for each document by considering the frequency of query terms in the document, the overall document length, and the average document length across the dataset. These factors are combined using a weighting scheme that prioritises terms appearing frequently in shorter, more focused documents while reducing the impact of terms that are overly common or appear in longer documents [34].

The BM25 relevance score for a document D and query Q is:

$$\text{BM25}(D, Q) = \sum_{q \in Q} \text{IDF}(q) \cdot \frac{\text{TF}(q, D) \cdot (k_1 + 1)}{\text{TF}(q, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Where:

- q: A term in the query.
- $\text{TF}(q, D)$: Term frequency of q in document D.
- $\text{IDF}(q)$: Inverse document frequency of q.

- $|D|$: Length of document D.
- avgdl: Average document length across the corpus.
- k_1, b: Tunable parameters (commonly $k_1 = 1.5$, $b = 0.75$).

The ensemble retriever assigns equal weights (0.5) to both methods, resulting in a total of four contexts (two from each search) for every query. Figure 4.3 summarises the steps taken during the retrieval phase.

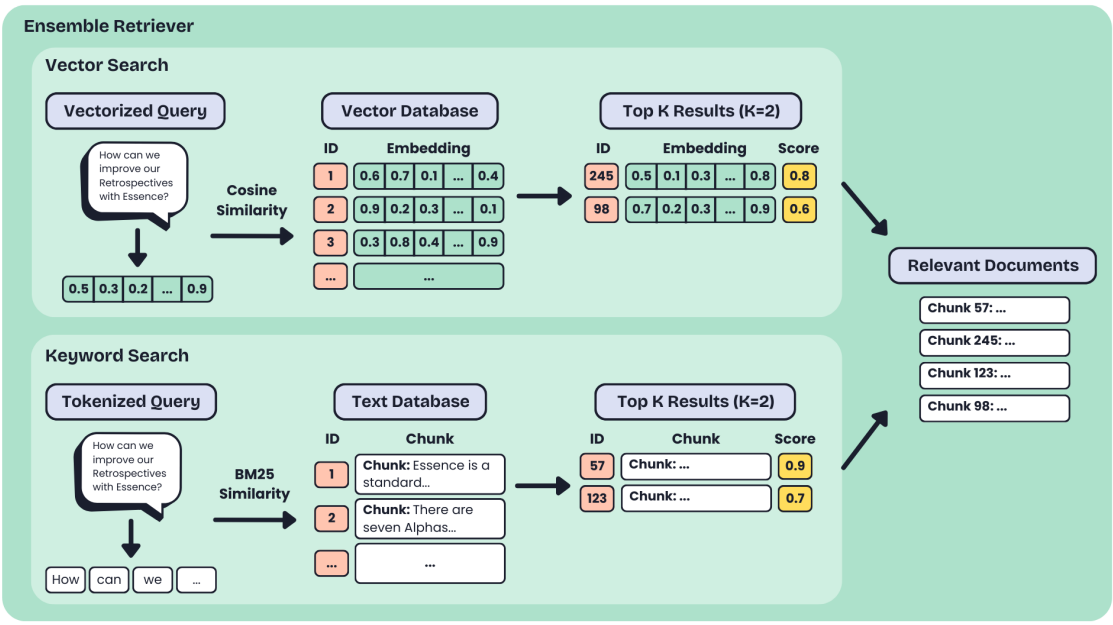


Figure 4.3: Ensemble Retriever.

By combining vector search with BM25, the ensemble retriever ensures more comprehensive and balanced retrieval, providing the model with the most relevant and contextually rich information.

User Interface

The user interface of the chatbot is designed to be intuitive and user-friendly. It allows users to input their queries directly and view the chatbot's responses in a conversational format. On the back-end, the chatbot is powered by Flask, which handles user requests. The back-end also stores query-response data in MongoDB. Figure 4.4 shows what the user interface of Essence Coach looks like.

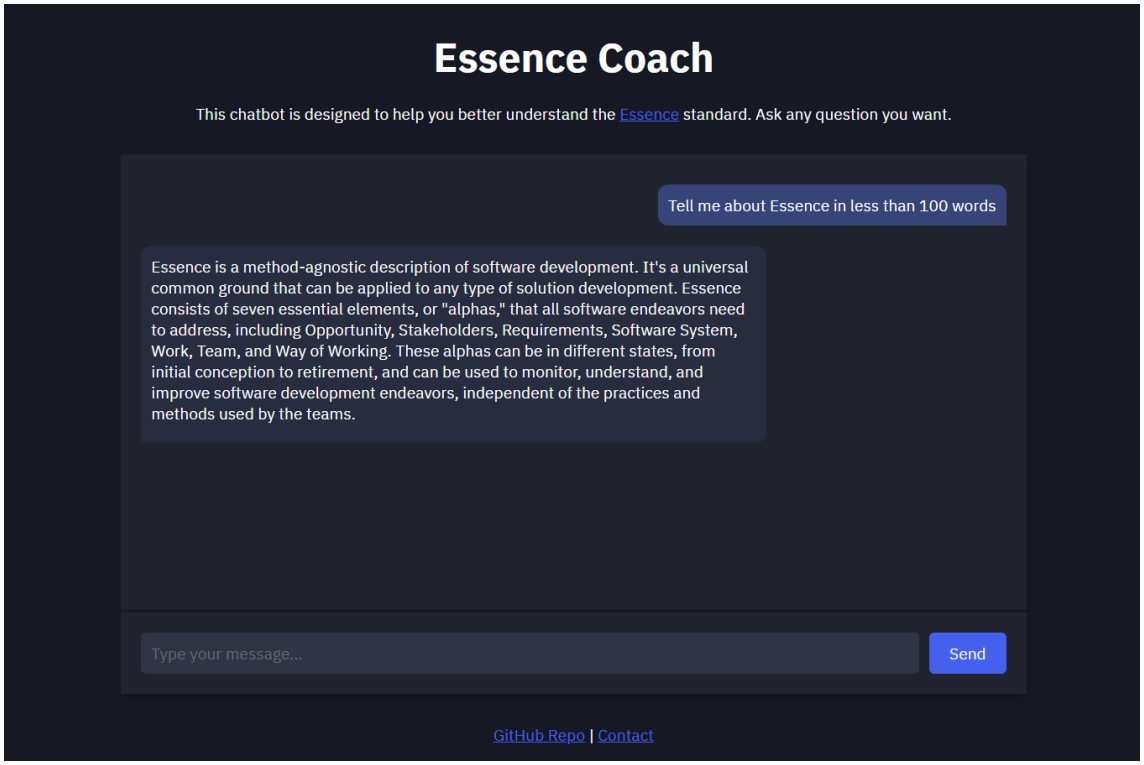


Figure 4.4: Essence Coach user interface.

4.2.2 Data

The foundation of a RAG system lies in the quality and structure of the data it retrieves. For Essence Coach, the system retrieves its information from a carefully

curated database of 22 documents sourced from diverse materials. These documents cover a wide range of content related to Essence and software engineering practices.

Initially, I tried looking for existing datasets that might contain information about software engineering practices and Essence in particular. However, I could not find any suitable resources. This led me to the decision to create my own dataset, tailored specifically to the needs of my project.

The documents I gathered for the dataset came from a variety of sources, including course materials, official Essence documentation, academic research articles, and other relevant publications. These diverse sources provided a rich base of information on Essence and its applications in software engineering, ensuring that the dataset would be comprehensive and representative of different aspects of the topic.

Initially, I attempted to convert the collected documents into JSON format to structure the data. Unfortunately, this approach was unsuccessful due to poor text separation and categorisation, which resulted in inconsistencies and errors in the structure. Additionally, the contextual metadata that JSON format would have provided was not necessary.

As a result, I opted to convert the documents into markdown format, using an automatic converter initially and then manually revising them. During the revision process, I focused particularly on ensuring proper heading levels (such as H1, H2) because the chunking of the text depended heavily on these headings. I also removed duplicate content to reduce noise and standardised the text formatting.

A particularly helpful source of information during the documents revision was the Essence WorkBench from Essify [35]. This platform allows users to play Essence games and visually build their methods by combining previously essentialized practices. Its collection of over ninety practices was especially useful during the revision of the RAG dataset since it allowed me to include more detailed information about the various practices, such as the work products they produce and the activities they involve.

The pie chart in Figure 4.5 illustrates the distribution of document types within the dataset. Of the 22 total documents, 50% are focused on software engineering practices, such as Scrum and retrospectives, and their essentialization. Five documents cover the Essence kernel and language, providing a general understanding of Essence and its foundational elements. Three documents focus on Essence games, and another three are dedicated to Essence cards. These card-based documents are particularly useful for ensuring that the model knows how to structure the information about the different Essence elements. All documents can be found in the project's GitHub repository, along with the source code [36].

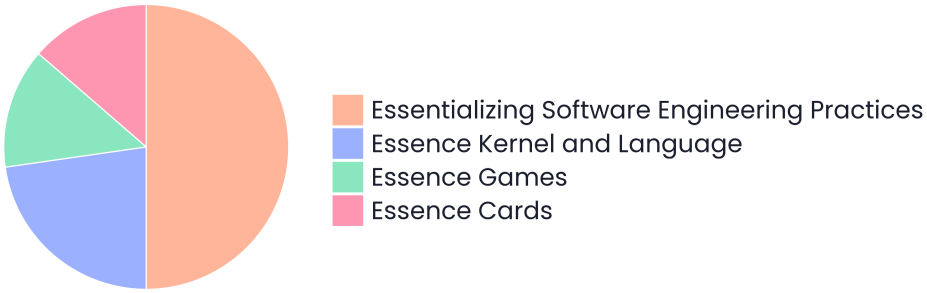


Figure 4.5: Distribution of documents in the retrieval database.

4.2.3 Tools

This project utilises a range of modern technologies to build the Essence Coach chatbot and collect its responses, including:

- **PDF to Markdown Converter:** A tool to convert documents into a structured markdown format.
- **Essence WorkBench:** A platform to conduct teams health-checks, organise the way of working and play Essence games with several essentialized practices.

- **ChromaDB**: A vector database used for storing document embeddings and performing semantic searches.
- **BM25**: A keyword search algorithm used alongside ChromaDB for the ensemble retrieval.
- **LangChain**: A framework that integrates the retrieval system with the language model for seamless context-based response generation.
- **Llama3 (via Groq API)**: A pre-trained large language model used to generate responses based on retrieved context.
- **MongoDB**: A NoSQL database for storing chat history and model's configurations.
- **Flask:** A Python web framework used to create the chatbot's backend, handling API requests and communication with the model.

4.2.4 Optimisation Strategies

The process of creating the chatbot entailed a significant amount of trial and error, as different strategies were tested to improve the overall performance.

One of the key optimisations involved testing different configurations of the language model, such as experimenting with various pre-trained models and adjusting parameters like temperature.

Another important aspect of optimisation was refining the chunking strategy for document retrieval. Initially, I experimented with random chunking based on character lengths, but this approach proved inefficient and often led to fragmented and incoherent contexts. After further experimentation, I adopted the document-based splitter strategy, which splits documents at meaningful header points in the markdown format. This method preserved logical units of information, which significantly improved the quality of the responses.

For document retrieval, I tested several search strategies to get the best results. While simple keyword search or vector search methods provided useful outcomes, they were not always sufficient on their own. After testing both approaches, I decided to implement an ensemble retriever that combines vector search and BM25 keyword search. The ensemble approach improved the retrieval accuracy and therefore the generated answer.

Chapter 5

Results

This chapter presents the experiments conducted as part of this thesis, focusing on the evaluation of the chatbot's performance. It outlines the different use cases tested and discusses the metrics used for the evaluation.

5.1 Challenges of Evaluation

Evaluating the responses of large language models and retrieval-augmented generation systems is challenging due to the inherent complexity and variety of factors that influence the quality of the output. The process involves assessing not only how well the system retrieves relevant information from the database but also how accurately and coherently it generates responses based on that information. Additionally, the subjective nature of evaluating the relevance of the responses further complicates this, as the desired outputs may vary depending on the specific question or user context, making standard evaluation methods harder to apply.

The study conducted by Yu et al. [37] identifies three key components for evaluating RAG systems: retrieval, generation, and the entire system's performance. Within

these components, there are various factors to consider, such as the accuracy of document retrieval, the quality of response generation, latency, scalability, and user satisfaction. In this study, the focus was placed primarily on the first two aspects. Accuracy of document retrieval was necessary to ensure the system fetched the most relevant content, while the response generation quality was essential for producing useful answers. Although speed and scalability are important for large-scale deployments, they were not a primary concern in this case, as the chatbot was not intended for heavy traffic, so they were only marginally taken into consideration.

5.2 Experiments

To assess the effectiveness of the system, I conducted a series of experiments aimed at evaluating the accuracy of the document retrieval and the quality of the generated responses.

I created thirty questions, evenly distributed across three use case categories: providing information about Essence, helping in the decision-making process, and translating practices. Some of these questions have known answers within the RAG system, while others do not. For instance, all the "information" questions were written starting from the contents of the Essence documentation, while the "decision-making" questions were formulated from scratch, knowing that there wouldn't be a precise answer anywhere in the dataset. This allowed me to evaluate how the model performs when the exact answer is not directly retrieved, and whether the retrieved context is still useful for generating an accurate or relevant response.

A few examples of the questions asked can be seen in Table 5.1. Every question also imposes a specific word limit to the model so that the length of the answer doesn't affect its evaluation.

ID	Question Type	Question	Ground Truth
1	Information	What is the Essence standard by SEMAT? Write less than 100 words.	The Essence standard, developed by a a community of software engineering experts...
2	Information	What are the Alphas of the Essence Kernel? Write less than 100 words.	The Kernel Alphas include Opportunity, Stakeholders, ...
...	...	...	...
11	Decision-Making	What practices would you recommend to a group of students who have to develop a mobile game for a university project? Write less than 100 words.	I recommend practices that can be easily implemented by students. These practices should still cover most Activity Spaces...
12	Decision-Making	What practice(s) can help us address the "Explore Possibilities" Activity Space? Write less than 100 words.	A practice that can help fill the "Explore Possibilities" space could be Prototypes...
...	...	...	...
21	Translation	Describe the pair programming practice using the Essence language, in less than 100 words.	Pair programming affects the Software System Alpha, it has one Activity which is Writing Code...
22	Translation	What's the difference between user stories and use cases? Explain it using the Essence language, in less than 250 words.	Use Cases influence the Requirements Alpha and present more activities and work products, therefore they are more detailed...
...	...	...	...

Table 5.1: Examples of test questions.

Next, I input these questions into the chatbot, saving both the model's responses and the retrieved contexts in a dataset for later evaluation.

5.2.1 Evaluating the retrieved context

For evaluating the quality of the retrieved context, I used several standard information retrieval metrics: Precision@k, Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP) [38]. These metrics provide a quantitative assessment of how effectively the system retrieves relevant contexts in response to a given query.

The relevance of each retrieved context was manually set based on whether the context contained relevant information to answer the user's question. Relevance was assigned on a binary scale, marking a context as either relevant or not relevant.

Precision@k measures the proportion of relevant documents in the top-k results, allowing for an evaluation of how many of the retrieved contexts are relevant within the first few results.

$$\text{Precision@K} = \frac{\text{Number of relevant items in top K}}{\text{K}}$$

Mean Reciprocal Rank is a metric that evaluates the rank of the first relevant context. It is particularly useful for assessing how quickly the system retrieves the most relevant context in response to a query.

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank}_i}$$

Mean Average Precision averages the precision scores across all queries, giving an overall evaluation of the retrieval quality over a set of queries.

$$\text{MAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

$$\text{AP}_i = \frac{1}{\text{R}} \sum_{j=1}^{K} \text{Precision@j} \cdot \text{relevance}(j)$$

To present the results, a table was generated showing the evaluation metrics for all thirty questions (Table 5.2).

Metric	**Value**
Precision@K	0.731
Mean Reciprocal Rank (MRR)	0.653
Mean Average Precision (MAP)	0.769

Table 5.2: Precision@K, MRR and MAP of the retrieved context.

5.2.2 Evaluating the response

When evaluating the chatbot's responses, the main challenge lies in assessing the faithfulness and accuracy of the content in relation to the input data. However, the evaluation of correctness is not always straightforward, as it can depend on the specific task or context.

The questions for evaluation were asked in a single setting, meaning within one continuous chat session. This allowed the model to refer to previous messages, which could impact the accuracy and relevance of the responses, as context from prior exchanges can influence the generated content.

To evaluate the quality of the responses, I employed two complementary approaches: BERTScore and human evaluation.

BERTScore Evaluation

BERTScore is a metric for evaluating text generation tasks, as it captures semantic similarity between texts using pre-trained contextual embeddings from BERT. It is especially useful for tasks where meaning, not just exact wording, matters, providing a more nuanced evaluation than traditional metrics like BLEU or ROUGE [39].

It evaluates the precision, recall, and F1 score based on the embeddings, rather than exact token matches.

First, I measured the similarity between the "ground truth", the ideal correct answer, and the answer generated by Essence Coach. Then, I calculated the similarity between the ground truth and the answer generated by the GPT-4o model (without any RAG). I wanted to compare the results with those generated by GPT-4o because it is a widely recognised and advanced model in natural language processing, offering a useful benchmark to assess how well Essence Coach performs in comparison to other state-of-the-art models.

Higher values of precision, recall, and F1 indicate greater similarity between the two texts, which serves as a strong indicator of the correctness of the answer, as it aligns more closely with the ideal, human-written response.

The different metrics in the BERTScore are based on the embedding similarity, which is calculated as follows:

Given:

- $X = \{x_1, x_2, \ldots, x_m\}$: Tokens in the generated text.
- $Y = \{y_1, y_2, \ldots, y_n\}$: Tokens in the reference text.
- $E(x_i)$: The embedding of the token x_i.
- $\text{sim}(x, y) = \cos(E(x), E(y))$: Cosine similarity between embeddings of x and y.

The similarity matrix S is:

$$S = \begin{bmatrix} \text{sim}(x_1, y_1) & \dots & \text{sim}(x_1, y_n) \\ \vdots & \ddots & \vdots \\ \text{sim}(x_m, y_1) & \dots & \text{sim}(x_m, y_n) \end{bmatrix}$$

$$S_{ij} = \text{sim}(x_i, y_j) = \frac{E(x_i) \cdot E(y_j)}{\|E(x_i)\| \|E(y_j)\|}$$

Where $\cdot$ is the dot product and $\| \cdot \|$ is the L_2 norm.

The precision in BERTScore is calculated by comparing the cosine similarity between the word embeddings of the generated text and the reference (ground truth) text. It measures how much of the content in the generated text is relevant and present in the reference text.

The precision score is the average of these maximum similarities:

$$\text{Precision} = \frac{1}{m} \sum_{i=1}^{m} P_i$$
$$P_i = \max_j S_{ij}$$

The recall in BERTScore measures how much of the relevant content from the reference text is captured in the generated text. Higher recall values indicate that more of the key information from the reference is present in the generated response.

The recall score is the average of these maximum similarities:

$$\text{Recall} = \frac{1}{n} \sum_{j=1}^{n} R_j$$
$$R_j = \max_i S_{ij}$$

The F1 Score in BERTScore is the harmonic mean of precision and recall, providing a balanced measure that considers both the relevance of the generated text to the

reference text (precision) and the extent to which the generated text captures the key information from the reference (recall).

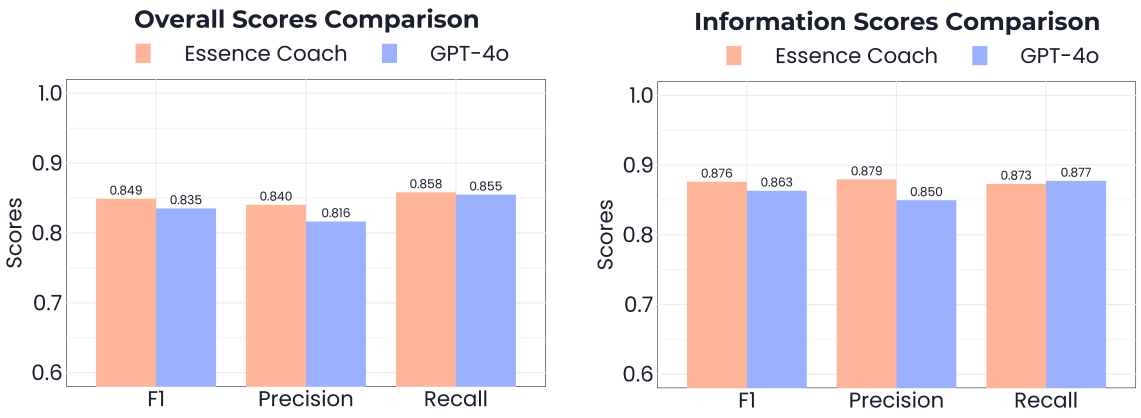
$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The results, divided by question type, are shown in the table below.

	Essence Coach			GPT-4o		
	Precision	Recall	F1	Precision	Recall	F1
Overall	0.840	0.858	0.849	0.816	0.855	0.835
Information	0.879	0.873	0.876	0.850	0.877	0.863
Decision-Making	0.834	0.854	0.844	0.809	0.848	0.828
Translation	0.808	0.847	0.827	0.791	0.839	0.814

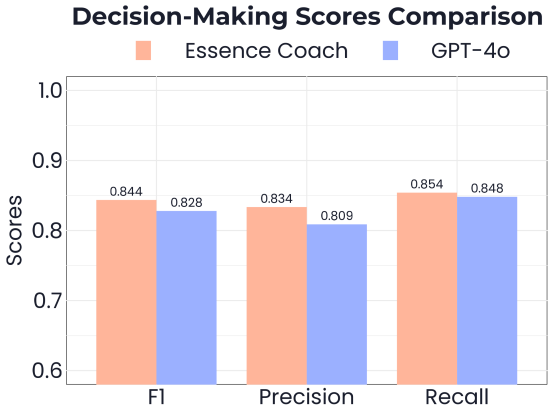
Table 5.3: Comparison of precision, recall, and F1 scores between Essence Coach and GPT-4o.

A better visualisation of the results can be seen in Figure 5.1.

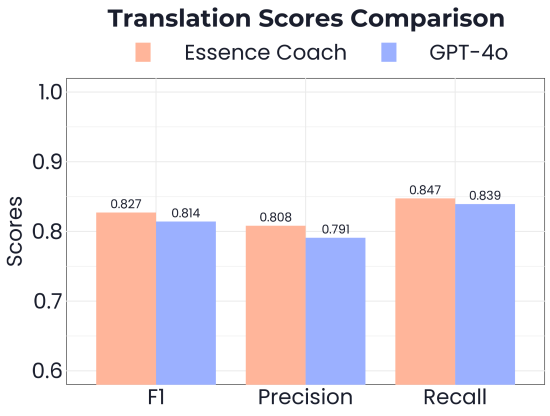


(a) Overall

(b) Information



(c) Decision-Making



(d) Translation

Figure 5.1: Bar plots comparison of F1, precision, and recall scores between Essence Coach and GPT-4.

Human Evaluation

While useful for automated evaluation, BERTScore has limitations. Responses can vary significantly in phrasing yet still be equally correct, or even diverge entirely while maintaining a moderately high similarity score. To address this limitation and ensure a more nuanced evaluation, human judgment was incorporated as an

additional assessment method.

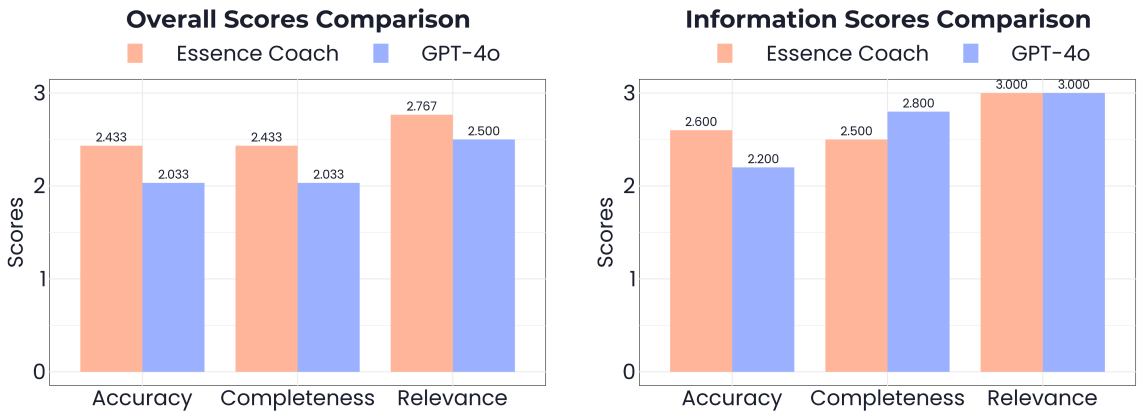
The human evaluation assessed responses based on three parameters: Relevance, Accuracy, and Completeness, each rated on a 0–3 scale [40]. Relevance measured how well the answer addressed the user's question, from completely irrelevant (0) to fully relevant (3). Accuracy evaluated the factual correctness, ranging from completely false (0) to entirely accurate (3). Completeness examined the level of detail, from lacking essential information (0) to providing sufficient and relevant detail (3).

Table 5.4 summarises the average scores for each parameter across all evaluated responses.

	Essence Coach			GPT-4o		
	Relev.	Accu.	Complete.	Relev.	Accu.	Complete.
Overall	2.767	2.433	2.433	2.5	2.033	2.033
Information	3.0	2.6	2.5	3.0	2.2	2.8
Decision-Making	3.0	2.7	2.5	2.8	2.6	2.2
Translation	2.3	2.0	2.3	1.7	1.3	1.1

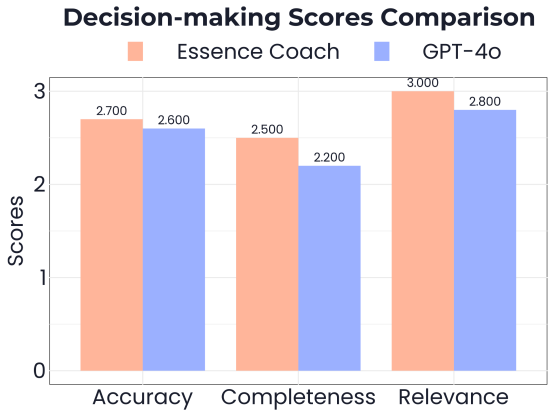
Table 5.4: Comparison of relevance, accuracy, and completeness scores between Essence Coach and GPT-4o.

A better visualisation of the results can be seen in Figure 5.2.

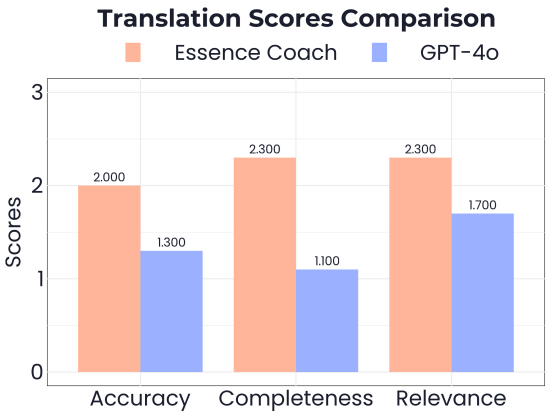


(a) Overall

(b) Information



(c) Decision-Making



(d) Translation

Figure 5.2: Bar plots comparison of relevance, accuracy, and completeness scores between Essence Coach and GPT-4.

Chapter 6

Discussion

6.1 Interpretation of the Results

The evaluation of Essence Coach has demonstrated that integrating a retrieval-augmented generation system with a specialised database of Essence-related documents improves the chatbot's ability to handle domain-specific queries compared to general-purpose language models. The results from both the retrieval and generation evaluation metrics highlight this advantage.

Precision@k, Mean Reciprocal Rank, and Mean Average Precision scores confirmed that the RAG system could retrieve relevant and accurate context around 70% of the time. A Precision@K of 0.731 indicates that 73.1% of the top retrieved contexts were relevant to the query. The Mean Reciprocal Rank score of 0.653 reflects a solid ranking performance, showing that the correct context often appears near the top. The Mean Average Precision of 0.769 demonstrates consistent retrieval accuracy across all queries (Table 5.2).

Moreover, the responses generated by Essence Coach, as evaluated by both BERTScore and human assessments, were more aligned with the "ground truth" answers than

those produced by GPT-4.

Overall, Essence Coach slightly outperforms GPT-4o in all metrics, achieving a higher F1 score (0.849 vs. 0.835), which suggests better alignment with the ideal responses. For questions related to general information about Essence, both models perform well, with Essence Coach achieving the highest scores (F1: 0.876 vs. 0.863). This was not particularly surprising considering most of the retrieval database contains information about Essence. Regarding decision-making questions, the scores are closer. This indicates that while both models give relevant advice, Essence Coach provides slightly more accurate answers. For translation, both models show a drop in performance compared to other tasks. Essence Coach (F1: 0.827) maintains an edge over GPT-4o (F1: 0.814), reflecting its better contextual understanding in translating practices into Essence language (Table 5.3).

Similar results can be seen in the human evaluation. Overall, Essence Coach outperforms GPT-4o in all metrics, particularly in accuracy and completeness. For Information questions, both models achieve perfect relevance (3.0), but Essence Coach shows higher accuracy (2.6 vs. 2.2), reflecting its factual correctness, while GPT-4o provides slightly more detailed responses. In Decision-Making, Essence Coach demonstrates a slight advantage in all three metrics. In Translation, Essence Coach significantly surpasses GPT-4o in relevance, accuracy, and completeness (Table 5.4). In particular, this question type demonstrates the importance of human evaluation, since the BERTScore wasn't able to capture how incorrect GPT-4o's answer was compared to Essence Coach, possibly because it remained similar in the vector space.

These results highlight the value that domain-specific knowledge integration can bring, particularly for topics like Essence that are not widely represented in general LLM training data.

6.2 Limitations of the Study

Despite the positive results, this study had several limitations. First, the dataset used for training the RAG system, though diverse, consisted of only 22 documents. While sufficient for this proof of concept, the limited scope of the database could limit the chatbot's ability to handle edge cases or unexpected queries. Furthermore, the chatbot's performance was only evaluated in a controlled environment without real-world user input. This means that metrics such as user satisfaction and practical utility in educational or professional settings remain unexplored.

Another limitation was the reliance on human evaluation to complement BERTScore in assessing response quality. While the Likert scale provided a structured framework, it introduced potential subjectivity, especially in evaluating parameters like relevance and completeness. This is also due to the fact that the ground truth wasn't the only acceptable answer and a practice could be translated in Essence in several different ways, depending on the level of detail that is required.

Additionally, the decision not to evaluate speed or scalability, though justified by the scope of the thesis, leaves open questions about the system's performance under higher user loads or in more complex operational settings.

6.3 Future Work

Building on what we've learned from this study, there are a few areas we can explore further. One promising direction is conducting user-based evaluations, particularly involving students, to assess how effective Essence Coach is as a learning tool. By incorporating real-world feedback, it will be possible to measure user satisfaction, identify gaps in the system's functionality, and better understand its actual use cases.

Another area for improvement is fine-tuning the LLM with a custom dataset of

input-output pairs specific to Essence and related practices. While this was not feasible in this study, fine-tuning could enable the model to generate even more accurate and context-aware responses. A comparative analysis between a fine-tuned model and the RAG system could provide deeper insights into the strengths and weaknesses of both approaches.

Moreover, a future direction could involve integrating Essence Coach with the Essence WorkBench from Essify [35]. With this integration, the chatbot could look up and collect information directly from the user's current board, such as the cards in use, practices selected, or the set target goals. This could allow Essence Coach to answer questions specific to the game the user is playing, the practice they are mapping, the health check they are conducting, etc. Currently, this is not possible as the chatbot cannot access the user's current board state unless they explicitly write it in the prompt, a process that could be tedious and impractical for the user. Developing a way for the chatbot to automatically access this information would greatly improve its ability to assist users in highly specific, real-time scenarios.

Finally, expanding the database to include more documents and experimenting with ensemble techniques that weigh retrieval methods dynamically instead of statically could further improve the chatbot's responses.

Conclusion

The objective of this thesis was to explore the potential uses of the Essence standard when combined with large language models. In practice, this study aimed to create a chatbot that could provide detailed information about Essence and help with the management of software engineering practices.

To achieve this, an application integrating Llama 3 with a retrieval-augmented generation system was developed. The RAG system was designed to provide additional information to the user's queries in order to improve the model's answers. Relevant context was retrieved from a curated database of Essence-related documents. This application benefited from various optimisation techniques, such as using an ensemble retriever to search for the most meaningful context.

After finalising the chatbot, an experiment was conducted to evaluate its retrieval capabilities and response quality. The experiment involved asking the chatbot thirty questions, evenly distributed across its three main use cases, and collecting both the retrieved context and the generated answers. The retrieved context was then manually reviewed to assess its relevance, and different metrics were calculated to determine its precision. The responses were evaluated in two ways: calculating their similarity to the ideal answer using BERTScore and conducting a human evaluation. These results were then compared with those achieved by a general-purpose model, GPT-4o.

The results of this experiment show that the application's RAG system was able to

retrieve relevant context over 70% of the time. In addition, the generated responses had an overall precision of 84% compared to the 81% of the general-purpose model. Human evaluation results show a similar trend with the chatbot's accuracy being 81% and GPT-4o's being 68%. In particular, the application excelled at answering general questions, presumably thanks to the information-dense retrieval database, while it struggled more when asked to translate entirely new and very specific practices.

Going back to the original research questions, the following answers are presented:

RQ1: How can a system that leverages large language models integrate and retrieve domain knowledge about Essence?

Answer: A system leveraging large language models can integrate and retrieve Essence domain knowledge through a RAG framework. It processes curated Essence documents into structured chunks, embeds them in a vector database, and retrieves relevant context using an ensemble of vector and keyword search. The retrieved context augments user queries for the LLM, providing context-specific responses.

RQ2: How effective is this new system in providing information related to Essence? In particular, how does it compare to other general-purpose systems?

Answer: The new system proved to be fairly effective in providing domain-specific information related to Essence. When compared to general-purpose models like GPT-4o, BERTScore results showed that Essence Coach had a higher alignment with ideal responses (F1: 0.849 vs. 0.835) as indicated in Table 5.3. Human evaluations further confirmed its advantages, with Essence Coach showing better performance in terms of overall relevance (2.767 vs. 2.5), accuracy (2.433 vs. 2.033) and completeness (2.433 vs. 2.033) (Table 5.4). These results emphasise the value that a RAG system can bring, thanks to its ability to retrieve more precise and contextually relevant information from a custom knowledge base, unlike general models, which can only rely on broader, less detailed knowledge.

In conclusion, this work explores a largely untapped intersection between artificial

intelligence and software engineering, focusing specifically on integrating Essence with LLMs. It highlights the potential of LLM-based applications to assist students in learning software engineering practices and to support development teams in managing them. The presented use case demonstrates promising results, but further testing with real users is necessary to fully understand its limitations and capabilities. This research lays a foundation for future studies on how AI can improve the understanding and implementation of software engineering practices, while also providing an example of a practical use case.

Bibliography

[1] Tyler A. Chang and Benjamin K. Bergen. "Language Model Behavior: A Comprehensive Survey". In: *Computational Linguistics* 50.1 (2024), pp. 293–350. DOI: 10.1162/coli_a_00492. URL: https://doi.org/10.1162/coli_a_00492 (cit. on p. 3).

[2] Muhammad Usman Hadi, Qasem Al Tashi, Rizwan Qureshi, et al. "A Survey on Large Language Models: Applications, Challenges, Limitations, and Practical Usage". In: *TechRxiv* (July 2023). DOI: 10.36227/techrxiv.23589741.v1. URL: https://doi.org/10.36227/techrxiv.23589741.v1 (cit. on pp. 4, 5, 7).

[3] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. *A Comprehensive Overview of Large Language Models*. 2024. arXiv: 2307.06435 [cs.CL]. URL: https://arxiv.org/abs/2307.06435 (cit. on pp. 4, 8, 9, 13, 15).

[4] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. *Large Language Models: A Survey*. 2024. arXiv: 2402.06196 [cs.CL]. URL: https://arxiv.org/abs/2402.06196 (cit. on pp. 5, 6, 10).

[5] Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David R. Mortensen, Noah A. Smith, and Yulia Tsvetkov. *Do All Languages Cost the Same? To-*

kenization in the Era of Commercial Language Models. 2023. arXiv: 2305.13707 [cs.CL]. URL: https://arxiv.org/abs/2305.13707 (cit. on p. 7).

[6] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. "LINE: Large-scale Information Network Embedding". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. International World Wide Web Conferences Steering Committee, May 2015, pp. 1067–1077. DOI: 10.1145/2736277.2741093. URL: http://dx.doi.org/10.1145/2736277.2741093 (cit. on p. 7).

[7] Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. *Randomized Positional Encodings Boost Length Generalization of Transformers*. 2023. arXiv: 2305.16843 [cs.LG]. URL: https://arxiv.org/abs/2305.16843 (cit. on p. 8).

[8] Zichong Wang, Zhibo Chu, Thang Viet Doan, Shiwen Ni, Min Yang, and Wenbin Zhang. *History, Development, and Principles of Large Language Models-An Introductory Survey*. 2024. arXiv: 2402.06853 [cs.CL]. URL: https://arxiv.org/abs/2402.06853 (cit. on p. 8).

[9] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. "On Layer Normalization in the Transformer Architecture". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 10524–10533. URL: https://proceedings.mlr.press/v119/xiong20b.html (cit. on p. 9).

[10] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4 (1993), pp. 185–196. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(93)90006-O. URL: https://www.sciencedirect.com/science/article/pii/092523129390006O (cit. on p. 9).

[11] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. 2022. arXiv: 2204.05862 [cs.CL]. URL: https://arxiv.org/abs/2204.05862 (cit. on p. 10).

[12] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. *LLM Inference Unveiled: Survey and Roofline Model Insights*. 2024. arXiv: 2402.16363 [cs.CL]. URL: https://arxiv.org/abs/2402.16363 (cit. on p. 10).

[13] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. *Extending Context Window of Large Language Models via Positional Interpolation*. 2023. arXiv: 2306.15595 [cs.CL]. URL: https://arxiv.org/abs/2306.15595 (cit. on p. 10).

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762 (cit. on p. 10).

[15] Xianrui Zheng, Chao Zhang, and Philip C. Woodland. "Adapting GPT, GPT-2 and BERT Language Models for Speech Recognition". In: *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. 2021, pp. 162–168. DOI: 10.1109/ASRU51503.2021.9688232 (cit. on p. 11).

[16] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: https://arxiv.org/abs/2312.10997 (cit. on pp. 12, 14, 15).

[17] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685 (cit. on p. 13).

[18] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. *Instruction Tuning for Large Language Models: A Survey*. 2024. arXiv: 2308.10792 [cs.CL]. URL: https://arxiv.org/abs/2308.10792 (cit. on p. 13).

[19] Gabrijela Perković, Antun Drobnjak, and Ivica Botički. "Hallucinations in LLMs: Understanding and Addressing Challenges". In: *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*. 2024, pp. 2084–2088. DOI: 10.1109/MIPRO60963.2024.10569238 (cit. on p. 14).

[20] Ivar Jacobson, Harold "Bud" Lawson, Pan-Wei Ng, Paul E. McMahon, and Michael Goedicke. *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* Association for Computing Machinery and Morgan & Claypool, 2019. ISBN: 9781947487277 (cit. on pp. 17, 18, 22–27).

[21] Debasish Jana and Pinakpani Pal. "ESSENCE Kernel in Overcoming Challenges of Agile Software Development". In: *2020 IEEE 17th India Council International Conference (INDICON)*. 2020, pp. 1–8. DOI: 10.1109/INDICON49873.2020.9342375 (cit. on p. 28).

[22] Teguh Raharjo, Betty Purwandari, Eko K. Budiardjo, and Rina Yuniarti. "The Essence of Software Engineering Framework-based Model for an Agile Software Development Method". In: *International Journal of Advanced Computer Science and Applications* 14.7 (2023). DOI: 10.14569/IJACSA.2023.0140788 (cit. on p. 28).

[23] Paolo Ciancarini and Marcello Missiroli. "Education to Agile: Fostering Team Awareness with Essence". In: *Frontiers in Software Engineering Education*. Ed. by Alfredo Capozucca, Sophie Ebersold, Jean-Michel Bruel, and Bertrand Meyer. Cham: Springer Nature Switzerland, 2023, pp. 69–84. ISBN: 978-3-031-48639-5 (cit. on p. 28).

[24] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. *Large Language Models for Software Engineering: A Systematic Literature Review*. 2024. arXiv: 2308.10620 [cs.SE]. URL: https://arxiv.org/abs/2308.10620 (cit. on p. 30).

[25] Vassilka D. Kirova, Cyril S. Ku, Joseph R. Laracy, and Thomas J. Marlowe. "Software Engineering Education Must Adapt and Evolve for an LLM Environment". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024. Portland, OR, USA: Association for Computing Machinery, 2024, pp. 666–672. ISBN: 9798400704239. DOI: 10.1145/3626252.3630927. URL: https://doi.org/10.1145/3626252.3630927 (cit. on p. 30).

[26] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. "A Deep Learning Model for Estimating Story Points". In: *IEEE Transactions on Software Engineering* 45.7 (2019), pp. 637–656. DOI: 10.1109/TSE.2018.2792473 (cit. on p. 31).

[27] Feng Lin, Dong Jae Kim, and Tse-Husn Chen. "SOEN-101: Code Generation by Emulating Software Process Models Using Large Language Model Agents". In: 2024. URL: https://api.semanticscholar.org/CorpusID:268681456 (cit. on p. 32).

[28] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes de Oliveira Neto. *Beyond Code Generation: An Observational Study of ChatGPT Usage in Software Engineering Practice*. 2024. arXiv: 2404.14901 [cs.SE]. URL: https://arxiv.org/abs/2404.14901 (cit. on p. 32).

[29] Sanka Rasnayaka, Guanlin Wang, Ridwan Shariffdeen, and Ganesh Neelakanta Iyer. *An Empirical Study on Usage and Perceptions of LLMs in a Software Engineering Project*. 2024. arXiv: 2401.16186 [cs.SE]. URL: https://arxiv.org/abs/2401.16186 (cit. on p. 33).

[30] Alireza Salemi, Surya Kallumadi, and Hamed Zamani. "Optimization Methods for Personalizing Large Language Models through Retrieval Augmentation". In: *Proceedings of the 47th International ACM SIGIR Conference on*

Research and Development in Information Retrieval. SIGIR '24. Washington DC, USA: Association for Computing Machinery, 2024, pp. 752–762. ISBN: 9798400704314. DOI: 10.1145/3626772.3657783. URL: https://doi.org/10.1145/3626772.3657783 (cit. on p. 41).

[31] Kush Juvekar and Anupam Purwar. *Introducing a new hyper-parameter for RAG: Context Window Utilization*. 2024. arXiv: 2407.19794 [cs.CL]. URL: https://arxiv.org/abs/2407.19794 (cit. on p. 42).

[32] Alireza Salemi, Surya Kallumadi, and Hamed Zamani. *Optimization Methods for Personalizing Large Language Models through Retrieval Augmentation*. 2024. arXiv: 2404.05970 [cs.CL]. URL: https://arxiv.org/abs/2404.05970 (cit. on p. 42).

[33] Warnia Nengsih. "A Comparative Study on Cosine Similarity Algorithm and Vector Space Model Algorithm on Document Searching". In: *Advanced Science Letters* 21.10 (2015), pp. 3321–3323. DOI: 10.1166/asl.2015.6481 (cit. on p. 42).

[34] Andrew Trotman, Antti Puurula, and Blake Burgess. "Improvements to BM25 and Language Models Examined". In: *Proceedings of the 19th Australasian Document Computing Symposium*. ADCS '14. Melbourne, VIC, Australia: Association for Computing Machinery, 2014, pp. 58–65. ISBN: 9781450330008. DOI: 10.1145/2682862.2682863. URL: https://doi.org/10.1145/2682862.2682863 (cit. on p. 43).

[35] Essify. *Essence Workbench - Empowering Software Engineering Excellence*. Accessed: 2024-11-27. 2024. URL: https://www.essify.com/essence-workbench (cit. on pp. 46, 64).

[36] Sonia Nicoletti. *Essence_LLM*. https://github.com/sonianicoletti/Essence_LLM. GitHub repository. 2024 (cit. on p. 47).

[37] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. *Evaluation of Retrieval-Augmented Generation: A Survey*. 2024. arXiv: 2405.07437 [cs.CL]. URL: https://arxiv.org/abs/2405.07437 (cit. on p. 50).

[38] Alireza Salemi and Hamed Zamani. "Evaluating Retrieval Quality in Retrieval-Augmented Generation". In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '24. Washington DC, USA: Association for Computing Machinery, 2024, pp. 2395–2400. ISBN: 9798400704314. DOI: 10.1145/3626772.3657957. URL: https://doi.org/10.1145/3626772.3657957 (cit. on p. 53).

[39] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL]. URL: https://arxiv.org/abs/1904.09675 (cit. on p. 55).

[40] Ankur Joshi, Saket Kale, Satish Chandel, and D. K. Pal. "Likert Scale: Explored and Explained". In: *Current Journal of Applied Science and Technology* 7.4 (2015), pp. 396–403. DOI: 10.9734/BJAST/2015/14975 (cit. on p. 59).

Acknowledgements

The realisation of this thesis would not have been possible without the support of many amazing individuals.

First, I would like to thank my thesis supervisor, Prof. Paolo Ciancarini. Despite not being one of his previous students, he graciously accepted to work with me, placing his trust in my potential. I particularly appreciated how he transformed our thesis-related meetings into opportunities for open discussion, broadening my perspective on the subject and posing thought-provoking questions.

This thesis, along with my entire academic journey, would not have been possible without the constant support, trust and love from my parents, Nada and Mario. They have always believed in me, encouraging me to pursue any goal I set my mind to. While they may say they are proud of me, I believe I am infinitely prouder of them for everything they have done.

Next, I want to thank my sister Alice, who has always been the pearl to my diamond, the soul silver to my heart gold and the white to my black. She is my rock and shield and I would be forever grateful if I could have even a fraction of her incredible strength.

A heartfelt thank you also goes to my friend and colleague Gianluca, with whom I have walked this path, step by step until today. He made every challenge feel surmountable, filled our days with fun and often politically incorrect jokes, and was always there to lend a helping hand when I needed him.

All my incredible friends have also been a constant source of comfort, support and motivation, especially Isabella and Erika, who never once let me doubt myself and always encouraged me to aim higher. They are the kind of friends that you never truly feel like you deserve, yet you can't imagine your life without them.

Finally, I want to thank my boyfriend Eric. Words cannot describe the millions of ways that he has helped me during these years. He is my north star, the light in darkness that guides me through the undiscovered seas of life. I have learnt so much from him, from knowing how to cook the perfect fried rice to understanding what being happy truly feels like, and my only wish is to never stop learning.

To all of you who are reading this and to all of those who can't, thank you.