
VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
SOFTWARE ENGINEERING STUDY PROGRAMME

# Method of Requirements Compliance Validation based on Graph RAG

# Reikalavimų kokybės atitikties validavimo metodas grindžiamas paieška papildytu generavimu grafo žinių bazėje

Master's Thesis

Prepared by Matas Čenys
Supervisor Doc. Dr. Asta Slotkienė
Reviewer Dr. Gražina Korvel

## Summary

Incorrect or ambiguous software requirements are a well-documented source of wasted effort and delivery failures in software projects, yet automated validation tools that combine methodological rigor with practical applicability remain scarce. This Master's thesis proposes a requirements compliance validation method based on Graph Retrieval-Augmented Generation (Graph RAG), in which ISO/IEC/IEEE 29148:2018-aligned compliance criteria – content, semantic, and data compliance – are evaluated by a Large Language Model (LLM) against a domain knowledge graph constructed from project documentation.

A systematic literature review confirmed that Graph RAG has not previously been applied to requirement compliance validation as a standalone method, establishing a clear research gap addressed by this paper. The proposed method was implemented as a configurable validation tool and evaluated against real-world requirements from a cyber-security project. Experimental results demonstrate that the method is capable of achieving F1 scores of 0.852 (content), 0.974 (semantic), and 0.903 (data). The method exhibits a consistent conservative bias, positioning it as an effective screening tool that surfaces requirements warranting human review rather than a fully autonomous arbiter of requirement quality.

**Keywords:** Graph RAG, requirement compliance, knowledge retrieval

## Santrauka

Netikslūs ar dviprasmiški programinės įrangos reikalavimai yra žinoma projektų nesėkmių priežastis, tačiau automatizuotos validavimo priemonės, derinančios metodologinį griežtumą su praktiniu pritaikomumu, tebėra retos. Šiame magistro darbe siūlomas reikalavimų atitikties validavimo metodas, paremtas grafo prieigos papildytu generavimu (Graph RAG), kuriame suISO/IEC/IEEE 29148:2018 standartu suderinti atitikties kriterijai – turinio, semantinis ir duomenų atitikimas – vertinami dideliu kalbos modeliu (LLM), kuris naudoja domeno žinių grafą, suformuotą iš projekto dokumentacijos.

Sisteminga literatūros apžvalga patvirtino, kad Graph RAG anksčiau nebuvo taikytas reikalavimų atitikties validavimui kaip savarankiškas metodas, todėl egzistuoja aiški tyrimų spraga, sprendžiama su šiuo darbu. Siūlomas metodas buvo įgyvendintas kaip konfigūruojamas validavimo įrankis ir įvertintas naudojant realaus kibernetinio saugumo projekto reikalavimų sąrašą. Eksperimentiniai rezultatai rodo, kad metodas gali pasiekti 0.852 (turinio), 0.974 (semantinio) ir 0.903 (duomenų) F1 įverčius. Metodui būdingas konservatyvus polinkis, todėl jis tinkamiausias kaip pirminės atrankos priemonė, išryškinanti reikalavimus, kuriems būtina papildoma specialisto peržiūra, o ne kaip visiškai autonomiškas reikalavimų atitikties vertintojas.


**Raktiniai žodžiai:** Graph RAG, reikalavimų atitiktis, žinių paieška

## List of Figures

## List of Tables

# Contents

# Introduction

This section explains the relevance and novelty of this research, as well as providing the research aim and objectives.

## Relevance and novelty

Requirements engineering and the quality of requirements are an important part of the project development process. Incorrectly defined functional and non-functional requirements can be the reason why the product provided for the stakeholders of the project does not meet the time and quality expectations that are raised for them, as indicated by the software quality problems, wasted effort and delivery delays that affect such projects [BRB+23]. Research shows that mistakes caused by incorrect requirements related to the software code base can be the reason behind 70% to 85% of the additional costs that had to be invested in order to correct the mistakes in the course of the entire project [AK17].

Considering the costs that incorrect requirements add to project development, it would be expected that this topic would be actively researched; however, a systematic literature mapping of research related to the user story, a popular way of describing functional requirements in Agile software development methodology, shows that there is a lack of validation and evaluation research in this field [AP22]. Even though peer-reviewed publications have been published, they lack testing in laboratory or real-world settings, leading to research being purely theoretical.

A further look into research made on software requirements validation and the tools or techniques already suggested by other researchers indicates that currently trending techniques are those that apply machine learning techniques with knowledge from dictionaries or ontologies – 36.7% of 66 identified primary studies suggest validation methods that adopt knowledge-oriented techniques [ABA+21]. Furthermore, it is visible that there is a lack of research into a validation model, applicable to different software development processes – from the 66 identified primary studies, 74.24% of them did not report the applicability of the suggested validation techniques over specific software process models, whereas requirement validation methods that would be specific to Agile software development methodology, which is the most popular methodology currently used in software development worldwide, are scarce [ABA+21].

To conclude, even though software requirements quality validation using knowledge-oriented tools is a popular research field, there is a lack of proposed solutions that would be tested not only in laboratories, but also in practical environments, while at the same time using advanced machine learning techniques.

## Research aim and objectives

This research aims to propose a requirements compliance validation method using LLMs and Graph RAG.

This aim shall be achieved by performing applied research with the following objectives:

1. Analyze software system requirements compliance validation process and the methods for performing it.
2. Perform a systematic literature review for assessment of LLM and RAG applications for requirements engineering processes.
3. Perform experiments with developed requirement compliance validation method to determine optimal configuration.
4. Propose conclusions and recommendations for further research in this field.

# 1. Requirements engineering process context and goal analysis

The analytical part contains theoretical information, supporting the research topic of this paper, as well as overview of the literature analysis, which has been carried out in order to establish the research gap, which would be the basis for solution proposed during the course of this research.

Firstly, the definition of requirements, their types and quality validation criteria applied to them is provided. Afterwards, the methods for requirement validation are explained, eventually discussing the usefulness of machine learning-enabled models for this process. Furthermore, the paper discusses the usage of Large Language Models (LLMs) for requirements engineering and other tasks, the limitations of these models and techniques, such as augmentation or prompt engineering, to overcome said limitations.

The section ends with a literature review of work related to requirements quality validation using LLMs and Retrieval Augmented Generation (RAG) methods, together with the process of selecting valid research papers, the results of their analysis and the research gap, which will be further explored in this paper.

## 1.1. Requirements and requirements validation

This section provides the definitions of requirement, their types and validation process, while also discussing the different methods available for validating requirements.

### 1.1.1. Requirement and requirement validation process

Every software project begins due to a stakeholder or a group of stakeholders having specific needs that have to be satisfied by the result of the project. However, successfully satisfying these needs is proving challenging, as indicated by the Standish Group's 2020 CHAOS report, which estimates that around 66% of all software fail due to being cancelled or delivered but never used [TSG20]. One of the leading factors for these failures – lack of clarity in requirements defined for the project.

According to IEEE Standard 729 [IEE83], a software requirement is:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in (1) or (2).

Based on this definition we can imply that a requirement is used as a written or modelled representation of a given user need, system specification or other characteristics that a developed

product should have. Based on the type of need that requirements express, they can be categorized into the following types [IEE18]:

1. Stakeholder requirements. This type captures the requirements raised by stakeholders and defines their expectations for system functions, performance and other needs. Stakeholder requirements are used as a foundation in preparing more detailed software requirements.
2. System requirements. This type is derived from stakeholder requirements and describes guides on how the system should be designed as a whole.
3. Software requirements. They describe the functionalities and constraints raised for software components in order to ensure that they align with system requirements.
4. Functional requirements. These define the behavior and functionality that the finished system or software is expected to perform. Functional requirements answer the question "What should the system do?". It is important to note that for a functional requirement to be valuable in the development phase, it must include constraints – a set of limitations imposed for the function explained in the requirement. There are many different ways to categorize functional requirements; however, the most common categorization currently in use is the one proposed by ISO/IEC/IEEE 29148:2018 standard, which groups these requirements according to their system function [IEE18]:
    a. Business rules – constraints derived from organizational policies or regulations. These can be closely related to legal regulations, dependent on the industry and geographical location of where the project is being carried out. Apart from internal policies, these rules can be impacted by GDPR, a regulation applicable in the European Union, HIPAA, an act in the United States, which defines how sensitive patient data should be protected, or other global or regional regulations.
    b. Transaction handling – requirements related to processing sequences, for example, sales transactions.
    c. Authentication and authorization – access control requirements for users and roles.
    d. Data management – creation, reading, updating and deletion of data.
    e. External interfaces – communication with other systems or users.
    f. Reporting – generation of outputs such as logs or reports.
    g. Validation and processing logic – specific logic the system must apply to inputs or internal data.

5. Interface requirements. This type defines the interactions of the developed system or software with external entities, such as hardware, users or even other systems. They are focused on compatibility and integration attributes of the finished product.
6. Design constraints. These impose restrictions on the design solution and can be impacted by regulatory standards, hardware limitations or development tools.
7. Non-Functional requirements. This type of requirement focuses solely on the quality and performance attributes of the system or software in question.

While all of these requirement groups are critical for the success of the project, a greater emphasis has to be put on functional requirements, as these requirements define the core functionality, expected from a developed product. If said requirements are expressed in an ambiguous way, lack key details or do not have constraints attached to them, the developed product may heavily deviate from that expected by the stakeholders of the project. Due to the severe consequences carried by requirements that lack the expected level of quality, a greater emphasis has to be put on the requirements quality validation process.

Requirements quality validation process can be divided into three key activities [IEE18]:

1. Preparation for validation, which involves identifying the scope for validation, inclusion of characteristics of the system or system elements to be validated and the results expected from validation.
2. Performing validation, which focuses on the execution of the validation actions that were identified in the previous stage.
3. Lastly, managing the validation results, which involves properly documenting the validated requirements and the results of their validation in order to maintain traceability.

Due to the complexity of requirements validation activity and the number of traits that have to be validated in order to confirm that requirements are described correctly and uniquely, various requirements validation checklists have been developed. These checklists aim to emphasize which traits of the requirement must be checked in order to confirm it as expressed correctly. Due to this, the traits mentioned in such checklists can be used to determine the most important characteristics that all requirements must have. An overview of traits as part of different developed checklists is provided below (Table 1).

Table 1. Overview of requirement traits validated by different proposed checklists [created by author using IEE18, INC23, IIB06]

| | ISO/IEC/IEEE 29148:2018 | NASA | INCOSE | BABOK |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Correct | + | | + | + |
| Unambiguous | + | + | + | , + |
| Complete | + | + | | + |
| Consistent | + | + | | + |
| Feasible | + | | + | + |
| Verifiable | + | | + | + |
| Necessary | + | | + | |
| Traceable | + | + | + | |
| Implementation-Free | + | + | | |

While the ISO 29148:2018 standard [IEE18] describes traits a good requirement should have, when looking into examples from other sources, it is clear that not all of these traits are employed in different sources. For example, the validation checklist used by NASA [NAS23] follows the traits described by the standard closely, but lacks points to validate requirement feasibility, verifiability, necessity and correctness, whereas INCOSE [INC23], the International Council on Systems Engineering, emphasizes those traits, but decides to leave out completeness, consistency and implementation-free traits. Lastly, BABOK [IIB06], the Business Analysis Body of Knowledge provides the most complete implementation of the traits mentioned in the standard, only leaving out necessity, traceability and implementation-free traits out of their evaluation recommendation.

Based on this analysis, the most important traits of a good requirement can be selected – unambiguity, mentioned by all of the sources, completeness, consistency, correctness and traceability, which are mentioned by the majority of the validation checklist sources, as well as the requirement and its validation process descriptions. These traits can then be mapped to compliance dimensions that must be checked during the validation process:

1. Semantic compliance, validating requirement unambiguity and consistency;
2. Content compliance, validating requirement completeness against the information that it is supposed to support;
3. Data compliance, validating correctness and traceability by evaluating the data and constraints mentioned in the requirement.

In summary, software requirements are an essential part of software projects that define certain traits that are expected from the finished product. Based on the need or constraint these requirements express, they are further categorized into different groups, one of which is the group of functional requirements, which describes the functionality that developed product should have, while also

expressing the constraints provided for said functions. As lack of details in functional requirements can lead to delays of delivery or even functionality being delivered not as expected, a greater focus must be put on the requirement quality validation process, during which a set of requirement traits must be checked. While the checked traits can differ depending on the project at hand and the industry it is meant for, the most important traits can be mapped to three compliance dimensions – semantic, content and data compliance.

### 1.1.2. Methods for requirement validation

While requirements validation checklists help to ensure that important traits of validated requirements are defined, they do not define the methods of how this process should be carried out. Even though the most effective way to carry out the validation process highly depends on the evaluated project, the "Systems and software engineering – Life cycle processes – Requirements engineering" standard [IEE18] mentions these validation techniques:

1. Stakeholder review involves conducting an analysis of the requirement with a group that includes the key stakeholders to determine that the system requirements are complete, correct and consistently reflecting the intent of the stakeholder requirements.
2. Prototyping, which can help in validating the interpretation of the system requirements, clarifying or examining requirements attributes and identifying any omitted requirements. The advantage of prototypes is that they provide a richer context for stakeholder evaluation and input, they can make it easier to interpret the assumptions, and they can provide useful feedback on why they are wrong.
3. Modelling and simulation can be used to assist the stakeholder validation of the requirements. The advantage of modelling and simulations is that they can demonstrate interactions and allow for sensitivity analysis when the results are not what the stakeholders expected.
4. Static conceptual modelling, with the purpose of aiding the understanding of the problem rather than to initiate design of the solution. Due to this, conceptual models comprise models of entities from the problem domain configured to reflect their real-world relationships and dependencies.
5. Lastly, formal modelling that uses notations based upon discrete mathematics and that are traceable to logical reasoning has made an impact in some specialized domains.

While these methods are all effective in uncovering issues related to described requirements, it is important to note that they rely heavily on a human specialist to carry them out and are also time-consuming both to prepare and carry out. Due to this, a significant effort has been put into uncovering

ways on how to optimize the process through the use of automated approaches – a literature review carried out by A. Hussain, covering papers and publications from 2010 to 2022 relating to application of machine learning in requirements engineering, indicates that the interest in this research topic has significantly increased since 2016 and out of 207 papers written in this topic, over 40% of them were written with regard to requirements validation and elicitation topics [CHW+22].

With this in mind, it is important to note that the requirements validation method should be chosen based on the specific needs of the project in question, but increasing interest in automated approaches in requirements engineering processes indicates that such tools are becoming a fundamental part of the requirements engineering process as a whole.

## 1.2. Evolution of automated requirements validation approaches

This section covers the evolution of automated requirement validation approaches, starting with manual and rule-based systems, moving to machine learning techniques and ending with LLM-based validation methods and prompt engineering techniques that can be used to fine-tune them.

### 1.2.1. NLP and rule-based validation

The first approaches aiming to optimize the requirement quality validation process explained in the previous section employed natural language processing (NLP) techniques. NLP techniques are a set of computational methods that enable machines to analyze, understand and process human language text [ZAF+21]. Two such methods are part-of-speech (POS) tagging and named entity recognition (NER), that have proven especially effective for requirements engineering [ZAF+22]. POS tagging is a technique where a sequence of words are processed and attached a part-of-speech tag (noun, verb, adjective and etc.), allowing a machine to better understand the structure of the sentences and find potential inconsistencies – like multiple verbs in a single requirement giving an indication that it could be ambiguous. NER technique on the other hand finds and classifies named entities in the given text, helping the machine to understand their importance in the sentence.

As these techniques cannot perform the requirement compliance validation process on their own, rule-based systems that employ them were developed with hand-crafted linguistic patterns to automatically flag specific defect classes in requirement text. These rules are typically implemented using dedicated text processing frameworks, such as GATE proposed by Rosadini et al. [RFG+17] for the task of evaluating a set of 1866 requirements from a railway signaling manufacturer, showing that structured NLP patterns could identify quality defects at a scale impractical for manual review.

While these tools already improve the effectiveness of the requirement validation process, they still showcase significant gaps:

1. Inability to detect and analyze patterns, that are not explicitly stated in their instructions;
2. Such frameworks are domain-specific and require experts to manually define every rule;
3. They lack semantic understanding of the text beyond the pattern matching as described in their rules.

### 1.2.2. Machine learning and LLM-based validation

To overcome the limitations of rule-based validation frameworks, approaches employing machine learning and later LLM-based validation methods were proposed.

Machine learning can be defined as technology that enables systems to learn from data and improve from experiences without being explicitly programmed [IEE22]. An alternative definition, as proposed by Tom M. Mitchell in his 1997 book "Machine learning" states that "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [MMC97]. This definition breaks down a system based on machine learning into three fundamental parts:

1. Experience (E), which stands for the data or interactions from which the system learns. An example of such experience could be a list of requirements that are categorized in advance for the program to have context of what is considered a complete requirement, and what is considered ambiguous or incomplete.
2. Tasks (T), that define the problem that the program is solving. An example would be the problem of performing requirements validation on a large set of requirements.
3. Performance (P), which provides a quantifiable measure that can be used to evaluate how well the proposed system can solve a given problem. An example of this would be a metric indicating how many requirements the system is capable of classifying correctly.

Delving deeper into how a machine learning model works, it is important to mention that the learning step can be implemented via a list of various techniques. The most fundamental techniques that can be used in this step are supervised, unsupervised and reinforcement learning [MMC97]. A system is considered as using supervised learning techniques if it learns to perform a given task by first training on a labeled dataset, which already has all of the correct answers assigned to each entry in the dataset, usually by a person with expert knowledge in the given task. A system is considered as using unsupervised learning, if it is learning from an unlabeled dataset. Such a system learns to perform a given task by learning the patterns between data entries in the unlabeled dataset. Another learning method would be reinforced learning, where the system learns based on a reward and penalty mechanism, where reward is given for a correctly performed action and penalty for incorrect one.

Another machine learning model learning technique is called self-supervised learning. This technique is a representation learning approach that creates surrogate labels from unlabeled data and trains models to predict those labels [LZH+21]. This is an especially useful learning technique for large language models as it enables the use of massive amounts of unlabeled text data, which is essential for learning general purpose representations of language.

While machine learning models already significantly improve over their rule-based predecessors for the validation task by learning patterns from data rather than relying on hand-crafted rules, they still require large labeled datasets for training and need to be trained separately for each task, which keeps them domain-specific. These limitations were addressed through the emergence of large language models (LLMs).

Large Language Models have emerged as cutting-edge artificial intelligence systems that can process and generate text with coherent communication and generalize to multiple tasks [ARC22]. Their efficiency at performing these tasks comes from three major factors – transformer block, massive computational power and a large-scale unlabeled dataset (Figure 1).

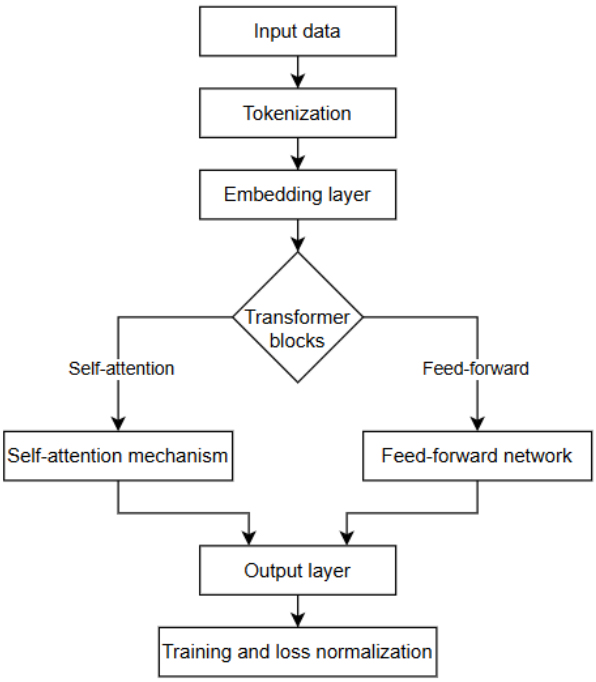


Figure 1. Large Language Model architecture

The transformer block is a processing unit that uses a self-attention mechanism to weigh the relevance of every word in the input against every other word simultaneously, allowing the model to capture long-range dependencies and contextual relationships in text regardless of how far apart the relevant words appear in the sentence.

Before the input can be processed by the transformer blocks, however, it first has to be split into tokens, which represent parts of a word or punctuation marks, which are then converted into numerical vectors that the model can process mathematically. Each LLM has a context window, which allows it to be more productive in a conversation setting or when needing to process a large block of text, like for example requirements specification, together with the regulatory documents that support it.

Research demonstrates that LLMs applied for requirement quality validation can provide binary evaluations per quality characteristic – including completeness, singularity and verifiability, generate rationales for their judgments and propose actionable improvements, with the recall metric being particularly high and a combined human-LLM assessment workflow producing stronger reviewer agreement than either approach alone [LFT+24].

To summarize, the usage of machine learning models and the newer LLMs for requirement quality validation task can further optimize the process by creating a method, that can adapt to the requested domain, without requiring a large labeled dataset for training.

### 1.2.3. Prompt engineering techniques

LLMs can employ prompt engineering – the process of designing input prompts to elicit the desired behavior from language models [WWS+22] – to further enhance the accuracy and robustness of the generated answers.

All of the different prompt engineering techniques work around the same LLM prompt structure, just focusing on improving different aspects of it (Figure 2). The prompt structure should include the following parts:

1. Inputs, that provide the data, which should be used for performing the task.
2. Task-relevant defines the task that the model is expected to perform.
3. Output is the structured output defined as an object.
4. Structure-relevant defines the instructions of how the output should be structured and formatted.

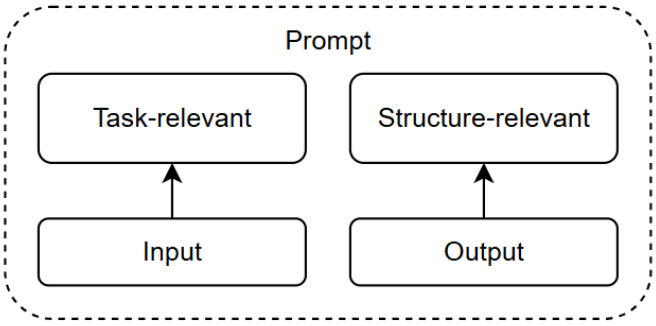


Figure 2. The structure of LLM prompt

Using this prompt structure, some techniques, like zero-shot and few-shot learning, focus on improving task-relevant part of the prompt to improve results, while others, like Chain of Thought and Tree-of-Thought focus on structure-relevant part, while still keeping the focus on task-relevant information:

1. Zero-shot learning is a 1-step technique, where only the task is provided for the model to solve. An example of such a prompt would be a simple question of "Can you tell me if requirement X is valid?". While an LLM will be able to answer such a query (and use the information learned from it in subsequent requests in the same context), the answer will be crafted according to the knowledge that the model has, which might not be the answer expected by the requester.
2. Few-shot learning is a 2-step technique, which provides both the task and the answer expected from the model. Expectations for the answer are expressed as examples of how the answer structure should look or by providing additional details for the model to take into account when preparing the answer [NKQ+23]. To use few-shot learning, the initial prompt should be updated in order to include examples of the expected answer – "Evaluate the validity of the following requirements. Requirement Y is valid, Requirement Z is invalid, Requirement Q is valid, evaluate validity of requirement X". In order to reach the full potential of few-shot learning, the examples given should be diverse for the model to learn the pattern between them, it should follow a consistent format and be concise not to use up too much of the context that model can handle at the same time.
3. Chain of Thought prompting technique encourages the model to generate intermediate reasoning steps by providing it with instructions on how the answer should be determined, which significantly improves the model's ability to solve arithmetic, common sense, and symbolic reasoning tasks [WWS+22]. This enhances model's logical reasoning and multi-step problem solving, while making output more transparent and easier to audit by a specialist, if a particularly sensitive topic to accuracy is handled by the model. An example of using chain of thought technique would be to provide model with a requirement to evaluate, while also asking it to give reasoning for every decision that it makes.
4. Lastly, Tree of Thought technique generalizes Chain of Thought prompting by allowing the model to explore multiple reasoning possibilities and backtrack when necessary [YYZ+23]. The key features of this technique are:

a. Each potential reasoning is explored as a node, where the LLM explores all these reasonings as multiple branches.
b. The model does not commit to one solution path – it evaluates every branch separately, backtracks if a particular branch does not lead to the desired answer and then chooses the most promising one of them all.
c. Symbolic search algorithms are combined with natural language reasoning in order to prepare the answer to the initial prompt.

These techniques allow Large Language Models to solve harder reasoning related tasks, while at the same time allowing them to provide more robust answers and even provide comparisons between alternative reasonings, further enhancing the effectiveness of the model.

## 1.3. LLM augmentation for domain-specific validation

### 1.3.1. LLM limitations

While LLMs provide unparalleled capabilities when it comes to text generation, classification and similar tasks thanks to the large amount of data they are trained on and the number of parameters that they have, these models are not without limitations [NKQ+23].

Firstly, LLMs can showcase the hallucination phenomenon, where the generated content is not supported by the input or factual world [JLF+22]. In such a situation, an LLM does answer the query passed to it in a fluent and plausible way; however, a specialist person with knowledge related to the field of the query will be able to tell that the answer is factually incorrect or in some cases even nonsensical. These hallucinations can be classified into four general categories:

1. Factual hallucination, where the answer provided by the model is not backed by any real-world data.
2. Intrinsic hallucination, where the output given by the model contradicts or misrepresents the input data. An example of this would be a summarization task of “requirement X is valid” as “requirement Y is valid”.
3. Extrinsic hallucination, where the model adds additional and plausible information, which is not supported by the initial query. An example of such a hallucination would be after passing a requirements specification for the model to validate, its summary of the document would include information that was not present in the requirements specification.
4. Confabulation, where the model creates made up named entities, dates or citations.

Due to their nature, these hallucinations can be caused by different reasons, like using unverified

internet data for model training [BEN12] or due to the initial prompt being of poor structure or too ambiguous to be interpreted correctly by the model.

However, the leading reason for such hallucinations, which can be considered a separate limitation of Large Language Models, would be a lack of grounding in real-world and real-time data, due to the models not being connected to a source of truth and therefore being unable to verify the factual correctness of the generated content [BMR+20]. The main reason why this happens stems from LLMs' roots as a machine learning model – it answers using the information on which it was initially trained, with the added improvement of also learning from the input that is provided for it in the form of input queries. This means that the latest information available to the model will be the one that was available at the time of training it, which leads to the model quickly becoming obsolete if a newer version of it is not trained and released, in order to update the general knowledge that the model has. Learning from the information provided by its users can be a valid approach to minimize this issue, but it carries its own drawbacks, as user input cannot be validated and could by itself become the reason for the model hallucinating.

Various different techniques have been suggested in order to overcome these issues – already discussed prompt engineering allows to minimize the risk of the model hallucinating and ensuring that it cannot misinterpret the given input, while data augmentation techniques have been proposed and developed in order to inject models with up-to-date information and improve the correctness and accuracy of the answer, according to the needs of the crafted task.

### 1.3.2. Retrieval augmented generation types

In the context of LLMs, augmentation is the process of providing additional information for the model to use, which was not present during the training of the model. Two key methods are available for augmenting the model's knowledge to improve the quality of the output generated by it – external search-based augmentation and retrieval-augmented generation (RAG).

The core idea behind external search-based augmentation is to give an LLM live access to a web (or enterprise) search engine and let it browse pages during inference, synthesizing the answer after the step is performed and often quoting or citing the pages found during its search [NHB+21]. However, a model that uses this type of augmentation has some key drawbacks stemming from its conceptual use of a search engine – it requires network calls and HTML parsing in order to get and read the pages it found during the search, which translates into a higher latency and cost of maintaining such a setup. Furthermore, models are susceptible to Search Engine Optimization (SEO) noise, paywalls, malicious content or generally incorrect or unverified content that is ever-present on the Internet. While these drawbacks can be managed by fine-tuning the LLM or integrating a rejection

sampling step in answer generation, which would be responsible for removing false or unverified sources before they are used by the model, they still remain an inherent weakness of such a setup, when compared to alternative solutions, like retrieval augmented generation.

Retrieval augmented generation combines a parametric language model with a non-parametric memory of documents that can be queried at inference time. By augmenting the generator with an external knowledge store, the system can incorporate up-to-date and factual information without retraining the model [LPP+20]. The main idea of this augmentation method is to fuse a neural retriever with a seq-to-seq generator, which is a neural architecture that maps an input sequence of tokens directly to an output sequence of tokens, inside a single model. At inference the retriever pulls top-k passages from a fixed vector index and the generator then conditions on those passages to produce the answer to a given query. This technique allows all parts of the model to run locally, reducing latency and computational power required to prepare an answer, however, this means that in order to update the knowledge used by the model, re-embedding and re-indexing the memory of documents is required. Even having this in mind, retrieval-augmented generation can provide more accurate responses for answers requiring information from highly specialized domains, by providing the specialist responsible for the model to extend its capabilities by providing up-to-date and domain-specific knowledge for it to use while answering inquiries, as explained by Patrick Lewis in his paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" [LPP+20].

There are five distinct sub-classes of retrieval augmented generation, all differing in their core traits and parametrization options (Table 2) – Naïve RAG, Advanced RAG, Modular RAG, Graph RAG and Agentic RAG.

Naïve RAG [GXG+24] represents the foundational implementation of retrieval-augmented generation as illustrated by the figure below (Figure 3) [NBG24].

Such a system relies on simple keyword-based retrieval techniques for fetching documents from static databases, that are then used to augment the LLMs generative capabilities. While simple and easy to implement, such systems suffer from several limitations:

- Scalability issues, due to the system struggling with large datasets and experiencing difficulties when trying to retrieve the most relevant information from them.
- Systems lack contextual awareness, since retrieved documents can fail to capture the semantic nuances of the query due to reliance on lexical matching rather than semantic understanding of the query.

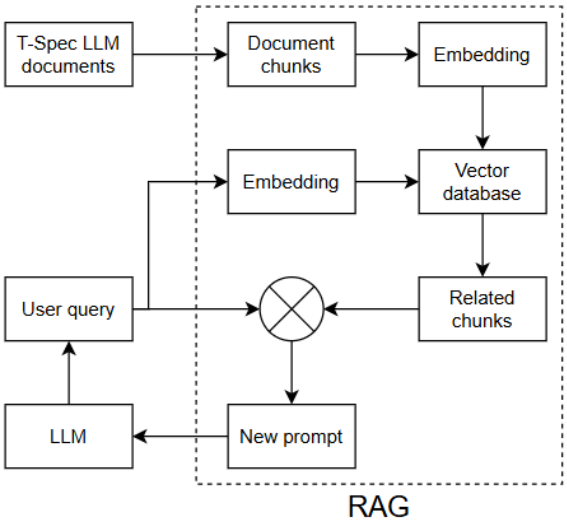


Figure 3. Overview of Naïve RAG

In order to fight said limitations, Advanced RAG has been developed to incorporate semantic understanding and enhanced retrieval techniques [GXG+24]. Such systems leverage dense retrieval models, that produce dense vectors for questions and passages and perform retrieval in the embedding space using inner product search [KOM+20], and neural ranking algorithms to improve retrieval precision. Furthermore, such systems have iterative retrieval capabilities, which enables reasoning across multiple documents for complex queries. While Advanced RAG greatly improves high precision and nuanced understanding of RAG, it still suffers from issues like computational overhead and limited scalability, especially when dealing with large datasets or multi-step queries.

Modular RAG is another evolution of the RAG paradigm, focusing on flexibility and customization by decomposing the retrieval and generation pipeline into independent, reusable components, enabling domain-specific optimization and task adaptability [GXG+24]. The core innovations introduced with Modular RAG:

1. Hybrid retrieval strategies, that combine sparse retrieval methods together with dense retrieval in order to maximize accuracy across different queries [KOM+20].
2. Composable pipelines, that enables retrievers, generators, and other components to be replaced, improved or reconfigured in order to adapt the system to the needs of specific use cases.

Another variation of the RAG paradigm is Graph RAG (GRAG) [PZL+24], which extends RAG systems by integrating a graph-based data structure as indicated in the figure below (Figure 4). This enables the system to generate richer and more accurate outputs, particularly in tasks that require relational understanding.

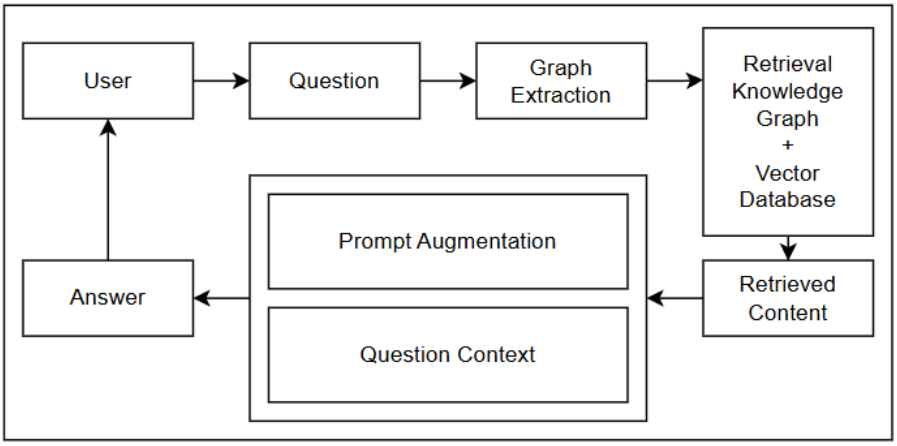


Figure 4. Overview of GRAG

The graph data structure can be adapted according to the needs of the project; however, some of the more popular structure diagrams are:

1. Lexical graph with hierarchical structure, in which the document is layered into a multi-level hierarchy of documents, sections, subsections and chunks. Such a structure is useful when document structure carries meaning and traceability of the information is the primary concern.
2. Memory graph, which stores conversational or session-level context as graph nodes, which makes the retrieval aware of prior interactions rather than treating each query independently
3. Lexical graph with extracted entities, which divides each document into chunks of text, as well as extracting the entities that are mentioned in said chunks. This allows the retrieval to traverse both the text structure and the semantic connections between entities, providing richer context than a simple vector search and is the most suited structure for a domain-specific system, where entity relationships can have an impact on retrieval quality.

Key characteristics of the GRAG system are:

4. Node connectivity, which allows it to capture the relationship between different documents or entities mentioned in said documents.
5. Hierarchical knowledge management, that allows the system to handle both structured and unstructured data through graph-based hierarchies.
6. Context enrichment, which adds relational understanding by leveraging graph-based pathways.

While capable of providing more accurate outputs, combined with relation understanding,

GRAG still suffers from limited scalability due to reliance on graph structures and high-quality graph data, combined with more complex integration effort with unstructured retrieval systems.

The latest advancement in this field is called Agentic RAG, which introduces autonomous agents capable of dynamic decision-making and workflow optimization [RSR24] (Figure 5).

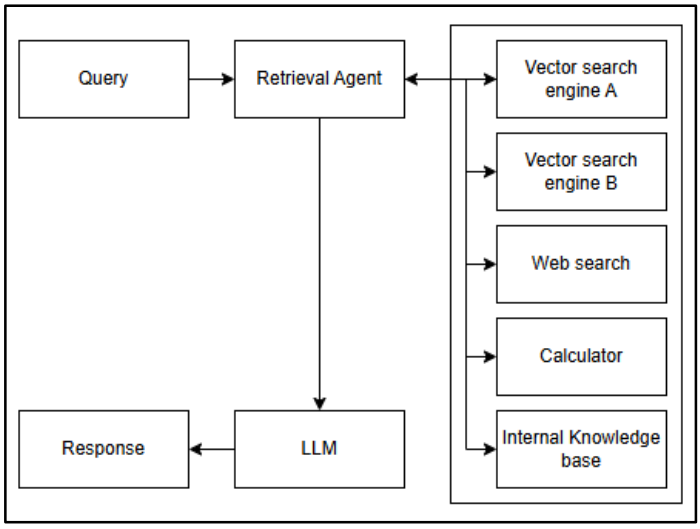


Figure 5. Overview of Agentic RAG

Unlike static systems, Agentic RAG employs iterative refinement and adaptive retrieval strategies to address complex, real-time, and multi-domain queries. It is characterized by:

1. Autonomous decision-making, due to each agent having the ability to independently evaluate and manage retrieval strategies based on query complexity.
2. Iterative refinement, by incorporating feedback loops to improve retrieval accuracy and response relevance.
3. Workflow optimization, by being able to dynamically orchestrate tasks, which enhances its efficiency in real-time applications.

While Agentic RAG is plagued by the same issues as GRAG, the improved adaptability and added ability of solving queries relating to different or compound domains makes Agentic RAG the most advanced and capable augmentation technique, that can be used together with LLMs to solve various text processing related tasks, such as requirements validation.

A summary of all the discussed RAG types is presented below (Table 3), together with their core traits and use-cases related to the requirements engineering field. It is important to note that all of these types can be used for different tasks in the requirements engineering field; however, the accuracy of provided results would be impacted by the selected type (more advanced being better) as well as the documents that the model will be able to use for query augmentation.

Table 2. Different RAG types, their traits, use-cases and parametrization

| **RAG type** | **Core traits** | **Use-case** | **Parametrization** |
|---|---|---|---|
| Naïve | Simple retrieval with no feedback loop<br>Single-pass query and generation | Requirements traceability validation | Top k – number of retrieved documents<br>Similarity metric – cosine, dot product<br>Retriever type – sparse or dense<br>Temperature and max tokens for generation |
| Advanced | Contextualization and scoring<br>Uses dense retrievers | Refined requirements justification retrieval | All Naïve RAG parameters<br>Hybrid retriever<br>Reranker – cross-encoder or scoring model<br>Context window |
| Modular | Decoupled components<br>Easier optimization and extension<br>Support pipeline tuning | Component-level requirements linking | All Advanced RAG parameters<br>Component swapping<br>Document chunking – size and overlap<br>Query expansion – paraphrasing and rewriting |
| Graph | Graph-based knowledge representation<br>Retrieval guided by semantic links<br>Strong at relational understanding | Dependency and impact analysis | All Modular RAG parameters<br>Edge weighting<br>Graph traversal strategy<br>Multi-hop – depth limit and aggregation method<br>Knowledge linking |
| Agentic | LLM agents that plan, retrieve and reason iteratively<br>Workflow optimizations | Interactive requirements elicitation and validation | All Graph RAG parameters<br>Tool use<br>Planning strategy – Chain-of-Thought, subtask decomposition<br>Feedback loop – self critique |

While Agentic RAG type can be viewed as the most advanced and hence most well suited for the topic of requirement quality validation, a GRAG can also be considered as a strong contender here as the compliance validation task has a fixed domain knowledge base, with predetermined compliance dimensions, with every query following the same structured pattern. These task traits mean that Agentic RAG's capability to dynamically plan multi-step retrieval across unknown or changing

information landscapes simply introduces unnecessary complexity and unpredictability for the task, while a GRAG system built on top of a lexical graph with extracted entities would allow for a deterministic, structured retrieval over a prebuilt domain graph, offering the relational context and traceability that compliance validation requires without the overhead and non-determinism of agent orchestration.

## 1.4. Systematic literature review on LLM-based requirements validation

As discussed in the previous sections, issues related to requirements engineering are a leading factor for project failures. According to the 2020 CHAOS report, 66% of all software projects are considered as failed due to resulting software getting cancelled or not being used [TSG20]. Such statistics indicate a dire need for an efficient method to aid in the requirements validation process and LLMs provide an appealing way to fully or partially automate the requirements validation process, especially in cases where a project consists of a large quantity of requirements, which would be time-consuming or problematic for a human specialist to go through. Furthermore, the usage of the RAG paradigm could further enhance such a tool, by allowing an LLM to gain up-to-date information and to be able to handle requirements specifications from projects tailored for highly specialized and regulated domains, such as medicine or law.

This section explores literature, which analyzes how LLM and RAG systems can be used in order to aid projects or specialists working in highly specialized domains and provides insights on research gaps existing in this field.

### 1.4.1. Literature review methodology

The review methodology was developed and executed according to the guidelines provided by Kitchenham and Brereton [KBB+09]. It consists of five steps – preparation for review, identification, screening, eligibility and analysis (Figure 6).

In the first step, research questions were raised, which would cover the main research question – *How transformer models are applied to aid in requirements quality validation?*

RQ1: What LLMs are applied in requirements engineering processes?

RQ2: What kind of RAG techniques are proposed for requirements engineering processes?

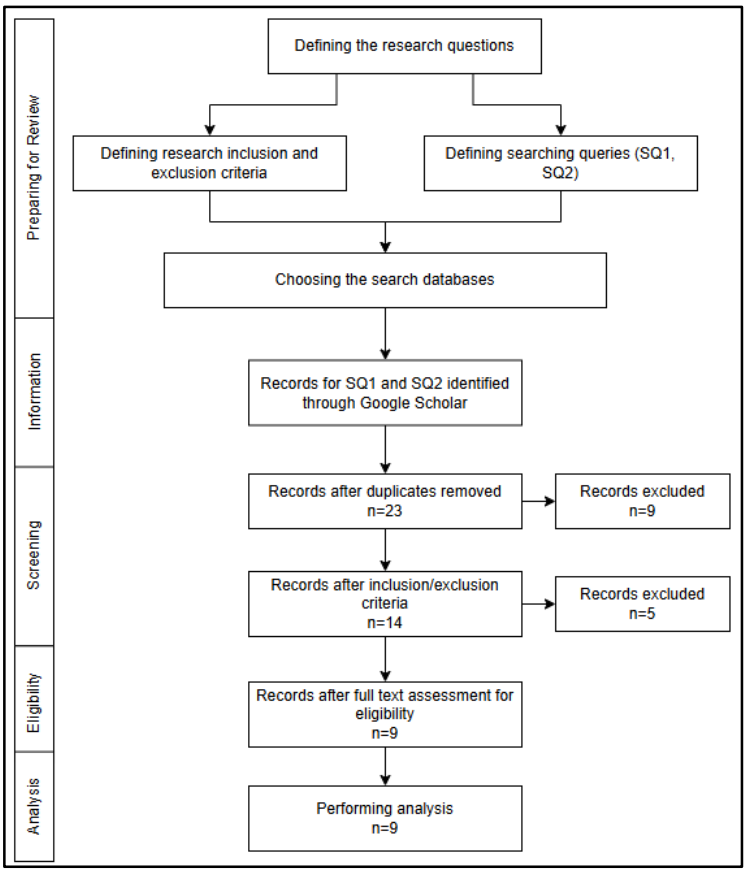


Figure 6. Flow diagram of the literature review process

To increase the accuracy of papers selected for literature review, inclusion and exclusion criteria were determined (Table 3).

Table 3. Study inclusion/exclusion criteria

| **Studies Inclusion Criteria (IC)** | **Studies Exclusion Criteria (EC)** |
|---|---|
| IC1. Studies were published after 2022 | EC1. Studies not accessible in full text |
| IC2. The publication must be written in English or Lithuanian | EC2. Publication is not written in English or Lithuanian |
| IC3. Studies that discuss LLM or LLM and RAG applications | EC3. Studies that are duplicated |
| | EC4. Studies do not relate to LLM or LLM and RAG applications |

The search strategy is prepared to include the main terms relating to the research topic of this paper in order to find articles on Google Scholar database:

1. SQ1 was developed to detect papers discussing the use of LLMs for requirements engineering purposes.

SQ1: *(LLM OR Large Language Model) AND (Requirement* engineering OR Requirement* validat*)*

2. SQ2 was developed to detect articles relating to the usage of RAG in tools proposed for requirements engineering processes.

SQ2: (*RAG OR Retrieval?augmented generation OR Knowledge Graph) AND (Requirement* engineering OR Requirement* validat*)*

The identification step is then performed by applying defined search queries in the mentioned scientific databases, also applying the defined exclusion criteria. Afterwards screening of found articles was performed in order to remove duplicates (no articles were removed), ensure that all remaining studies conform with inclusion and exclusion criteria (9 articles were removed), while eligibility was checked by reading all of the remaining articles (5 articles were removed), which were then reviewed again in order to answer the previously mentioned research questions.

### 1.4.2. Results of research questions

After performing the literature review methodology explained previously, 9 articles were selected as relevant for the research topic.

Out of the mentioned articles 5 directly answer RQ1 (Table 4). The LLM focused on by all of the researchers is the famous GPT model developed by OpenAI – two different versions of it commonly appear in research papers: a slightly older GPT-3.5, employed by Sami et al. for the requirements prioritization task [SAM24], and state-of-the-art GPT-4o. GPT-4o model is used for two main requirements engineering processes – verification, researched by Masoudifard et al. [MMM+24] and validation, which is analyzed by three different papers [KGV24][RSW+24][SNV+24]. While other LLMs are not researched as heavily, some are still mentioned in the scope of requirements validation:

1. Krishna et al. compared CodeLlama model against GPT-4o in order to evaluate the performance difference between the two models [KGV24].
2. Reinpold et al. decides to compare Claude 3.5 Sonnet against GPT-4o in the same requirements validation task and using identical datasets [RSW+24].
3. Lastly, Saleem et al. compared GPT model against latest Gemini model, 1.5 Pro [SNV+24].

Regarding the research question RQ2, more focus was put on understanding what RAG techniques and for what use-cases are proposed by different research. The findings of this review can be found on the table below (Table 4).

Table 4. Summary of RAG techniques and their applications for requirements engineering process

[created by author using MMM+24, LZL25, JYH+24, LCM+24, JCH+24]

| Research | Research goal for RE process | RAG technique | Results |
|---|---|---|---|
| Masoudifard et al. 2024 | Requirements verification | Graph RAG | Graph-based data retrieval tool with advanced prompting capabilities |
| Luo et al. 2025 | Domain specific question answering | Graph RAG | Causal Graph tool with cause-effect filtering. |
| Jin et al. 2024 | Domain specific question answering | Naïve RAG | LLMs with long-context window initially improves results, later showing worse results due to "hard negatives" |
| Li et al. 2024 | Domain specific question answering | Naïve RAG | Comparison against long context LLM, showcasing that some questions can only be answered with RAG |
| Jiao et al. 2024 | Domain specific question answering | Advanced RAG | Question answering tool with two-fold improvement in performance compared to Llama 2-7B model using 0-shot prompts |

As indicated above, the majority of the research has been focused on proposed RAG-enabled tools tailored for domain-specific question answering, a task for which RAG is most beneficial due to its underlying design. More recent papers investigate the use of Graph RAG for requirements verification [MMM+24] and question answering topics [LZL25], further enhancing the results via the use of cause-effect filtering or advanced prompting techniques, such as Chain-of-Thought and Tree-of-Thought.

The majority of remaining papers focus on the usage of Naïve RAG technique for question answering task, though providing slightly different insights:

1. Jin et al. propose a tool using an LLM with a large context window paired together with Naïve RAG technique, which showcases initially better results when compared to a simple LLM, but eventually showing worse results as more retrieved passages are introduced due to the issue of "hard negatives" – which is a negative example that is difficult for a model to distinguish from a positive one [JYH+24].

2. Li et al. instead pits a Naïve RAG-enabled tool against an LLM with a large context window [LCM+24]. This research reveals that while a large context window can be more beneficial initially, it cannot defeat a RAG-enabled model in cases where highly domain-specific information is required.

Lastly, Jiao et al. propose a question answering tool using a modified version of Advanced RAG called DuetRAG, which showcases two times better results when compared to Llama-7B and GPT-3.5 LLMs on open-domain question-answering dataset called HotpotQA [JCH+24].

To summarize, existing research focuses mainly on analyzing the capabilities of LLMs such as GPT-4o, one of the latest models developed by OpenAI, while the core focus remains on the requirements validation topic. When it comes to RAG techniques, most recent research focuses more on Graph-RAG techniques, exploring requirements verification task or domain-specific question answering, which is a commonly researched task in this field.

### 1.4.3. Conclusions and identified research gap

According to the analyzed sources, the requirement's quality criteria that avoid misunderstanding are unambiguousness, completeness and correctness. While other quality criteria are present in analyzed sources, they appear to be complementary depending on the field for which validation criteria are prepared. Due to this, the research performed in this thesis will focus on validating functional requirements completeness and correctness.

Furthermore, based on the systematic literature review, research related to LLM usage mostly focuses on the requirements validation process, while RAG usage for the same use case remains under-researched. What is more, there is a clear gap in research focusing on the impact that different parameters or prompting techniques have on the results of such methods, as well as evaluating the performance of the proposed method in real-world applications.

According to the literature analysis, the research will focus on proposing a method using LLM and GRAG method with configurable parameters for the requirement compliance validation process for a more precise and unambiguous validation of the mentioned requirement compliance criteria.

## 1.5. Evaluation framework for graphRAG-based validation

In order to validate the performance of the proposed method, the evaluation categories, as well as the metrics themselves need to be determined. This section is structured in subsections based on the evaluation categories, firstly discussing the metrics that will be used to evaluate RAG implementation, then discussing metrics for graph quality, finalizing with the gold standard comparison metrics, that evaluate the performance of the validation method as a whole.

### 1.5.1. RAG evaluation metrics

As there is no currently standardized method for evaluating performance of RAG techniques, Es et al. [EJE+25] has proposed a novel evaluation framework – RAGAS – for assessing the quality of a RAG system, which has been empirically validated with human judgment indicating a high agreement with human judgment on the three metrics proposed in the paper – faithfulness, answer relevance, context relevance (Table 5).

Table 5. RAGAS metric agreement with human judgment [created by author using EJE+25]

| Metric | Agreement with Human Judgment |
|---|---|
| Faithfulness | 95% |
| Answer Relevance | 78% |
| Context Relevance | 70% |

The faithfulness metric is used to determine how many claims in the generated answer were supported by information retrieved from documents available to the system (Table 6).

Table 6. Faithfulness metric [created by author using EJE+25]

| | |
|---|---|
| Definition: | Measures how well the generated answer is grounded in the retrieved context. |
| Goal: | Avoid hallucinations by ensuring all answer claims are supported by the context. |
| Method: | • Break the answer into atomic statements using an LLM.<br>• For each statement, use an LLM to verify whether it can be inferred from the context.<br>• Ratio of supported statements to total statements. |

This metric (F) is a number gained by dividing number of atomic statements extracted from the answer that are verified as supported by the context (V) from total number such statements (S) (Equation 1):

$$F = \frac{V}{S} \tag{1}$$

The Answer Relevance (AR) metric is used to check how closely the generated answer aligns with the original question. This is achieved by generating additional plausible questions from the answer using an LLM and then using embeddings to compute the similarity between these generated questions and the original question (Table 7).

Table 7. Answer Relevance metric [created by author using EJE+25]

| | |
|---|---|
| Definition: | Assesses whether the generated answer actually addresses the given question. |

| | |
|---|---|
| Goal: | Penalize incomplete, irrelevant, or overly verbose answers. |
| Method: | • Generate multiple plausible questions from the answer using an LLM.<br>• Use embeddings to compute the similarity between these generated questions and the original question.<br>• Calculate average cosine similarity score between the generated and original questions. |

The metric is a result of an average cosine similarity score between generated and original questions, as indicated by this formula, where $q$ is original question, $q_i$ is generated question and $n$ is the number of generated questions (Equation 2):

$$AR = \frac{1}{n}\sum_{i=1}^{n} \sin(q, q_i) \tag{2}$$

Lastly, Context Relevance (CR) measures how much of the provided context is actually useful for answering the question. This is performed by using an LLM to extract only those sentences from the context, that were needed to answer the given question and then calculating the ratio of extracted sentences against all sentences generated by the model (Table 8).

Table 8. Context Relevance metric [created by author using EJE+25]

| | |
|---|---|
| Definition: | Measures the focus of the retrieved context – how much of it is directly useful for answering the question. |
| Goal: | Avoid bloated or noisy context that increases LLM inference cost and reduces effectiveness. |
| Method: | • Use an LLM to extract only the sentences from the context needed to answer the question.<br>• Calculate the ratio of extracted relevant sentences to total context sentences. |

The formula for calculating Context Relevance is provided below (Equation 3):

$$CR = \frac{number\ of\ extracted\ relevant\ sentences}{total\ number\ of\ sentences\ in\ the\ context} \tag{3}$$

With the help of these metrics, researchers can evaluate the performance of a developed RAG enabled tool, by ensuring that all generated answers are supported by the context, penalizing incomplete, irrelevant or overly verbose answers, and by avoiding bloated or noisy context, which reduces the effectiveness of proposed model.

### 1.5.2. Graph quality metrics

While RAG metrics can evaluate the performance of the RAG implementation, the metrics

explicitly showcasing the performance of the graph data structure can also provide valuable insights into understanding how the RAG implementation is performing.

Firstly, the average Node Degree (ND) indicates how richly connected the entities are within the graph on average. This metric allows to spot whether missing relationships between entities directly limit the graph's ability to support downstream reasoning tasks, indicating that entity extractions from the provided documents may have been incomplete. The formula to calculate Node Degree is provided below (Equation 4), where E stands for number of edges (relationships) and V is the number of nodes (entities) in the graph:

$$ND = \frac{2 \times E}{V} \tag{4}$$

The Graph Density (GD) metric measures how many edges exist in a graph compared to the maximum number of edges that could exist if every node was connected to every other node. If density is closer to 0, most entities on the graph do not have relationships to other entities, indicating that entity extraction process pulled out domain concepts but failed to capture how they are related to each other. However, a dense graph, where Graph Density is close to 1, signals noisy extraction, where relationships were over-extracted, connecting entities that have no meaningful domain relationship, this way degrading retrieval precision. Graph Density formula is provided below (Equation 5), where E is the number of edges and V is the number of nodes:

$$GD = \frac{E}{V \times (V-1)} \tag{5}$$

Lastly, Graph Connectivity (GC) measures the proportion of domain entity graph that is reachable from any given entry point, aiming to evaluate properties that directly affect the usability of the graph for downstream tasks – isolated nodes and disconnected subgraphs represent structural deficiencies that then limit the graph's utility. Graph Connectivity formula is provided below (Equation 6), where LLC stands for largest connected component and V is the total number of nodes:

$$GC = \frac{LLC}{V} \tag{6}$$

Using these metrics, it is possible to determine the performance of the graph data structure against the RAG implementation metrics discussed in the previous section to better understand the performance of the method as a whole, when it comes to retrieving relevant context for the requirement quality validation.

### 1.5.3. Gold standard comparison metrics

The gold standard comparison method evaluates the results of an automated system against the results generated by domain-experts when performing the task [PS12]. In the case of this paper, all

requirements used in the research were evaluated by human specialists that provided scoring for all three compliance categories:

1. For content compliance three scoring options were available – passed, for full compliance, intermediate, in case a partial compliance is determined and failed.
2. For data compliance the same three scoring options were available – passed, intermediate and failed.
3. Due to the nature of semantic compliance, only two scoring options are available – passed and failed.

Evaluating the results generated by the proposed method against expert scoring, three metrics can be calculated to indicate the overall performance metrics of the proposed method – precision, recall and F1 score.

The precision metric measures how many evaluations of the method can be confirmed by expert judgement. A low scoring is an indication that the method has tendency to raise false alarms or evaluated the requirements too strictly. The formula for Precision is provided below (Equation 7), where TP stands for true positives and FP stands for false positives:

$$P = \frac{TP}{TP+FP} \tag{7}$$

Recall (R) metric focuses on measuring the proportion of actual non-compliant requirements that the LLM successfully detects. A low recall means that tool misses real compliance issues and has a tendency for too lenient evaluation of the provided requirements. Formula for Recall metric is provided below (Equation 8), where TP stands for true positives and FN stands for false negatives:

$$R = \frac{TP}{TP+FN} \tag{8}$$

Lastly, the F1 score is a harmonic mean of precision and recall, penalizing extreme values of either metric. F1 score ensures that the evaluation reflects performance on both outcomes rather than being dominated by the majority class. The formula for F1 score is provided below (Equation 9), where TP stands for true positives, FP stands for false positives, FN stands for false negatives and TN stands for true negatives:

$$F1 = \frac{2\times TP}{2\times TP+FP+FN} \tag{9}$$

It is also important to note that all of these metrics must be calculated for each compliance category separately, giving detailed insight as to how well the method is capable of handling each of the evaluated compliance categories.

# 2. Research methodology

This section covers the implementation of the LLM-based and Graph RAG enabled requirement quality validation method and its overall results. Section firstly discusses the research plan, then covers each step of the research plan in detail, finalizing with the experiment execution and result analysis subsections.

## 2.1. Research plan

The focus of this research is to propose an automated requirement quality validation method, which would validate requirements based on three categories – semantic, data and content compliance. The method shall be enabled through an LLM, paired with a Graph RAG knowledge base implementation.

Research plan consists of four steps in total – analysis of the subject area, preparation of the research environment, execution of the experiment and analysis of the gathered results in order to present final conclusions and recommendations for further work (Figure 7). The method shall be implemented through the course of this research using the Python programming language, due to it being the native programming language used in the LLM ecosystem.

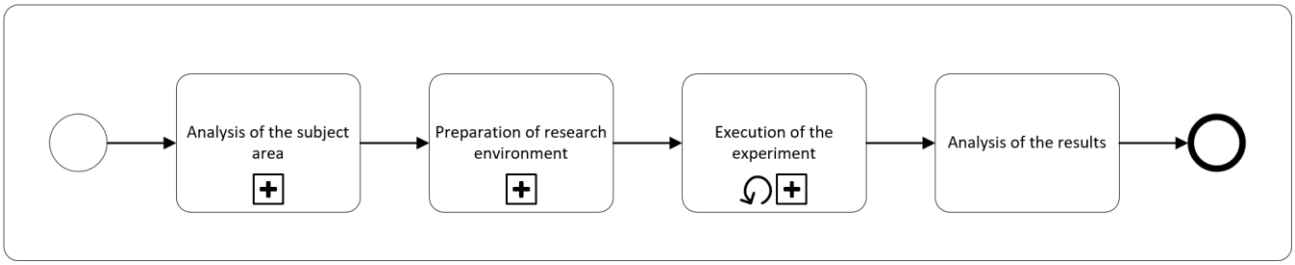


Figure 7. High-level research plan

## 2.2. Analysis of the subject area

In order to carry out the research, a subject area first had to be analyzed (Figure 8). This step is essential for the research in order to carry out meaningful experiments with requirements, written by human specialists with the goal of implementing real business needs, supported by documents.

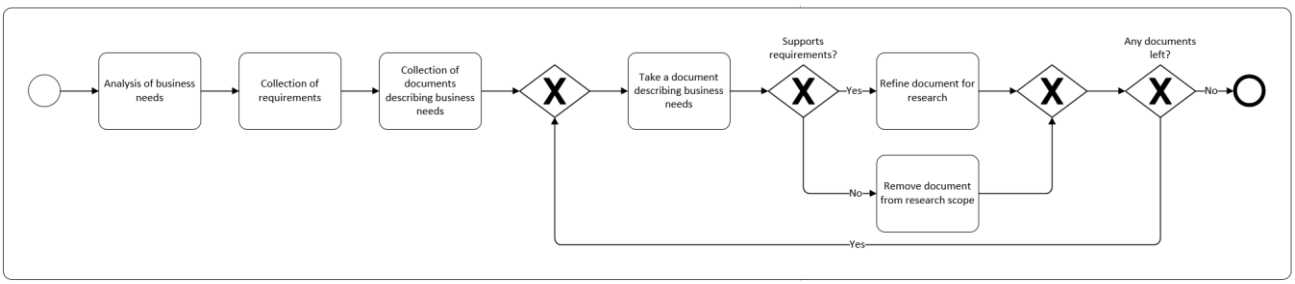


Figure 8. Process for analysis of the subject area

The requirements and supporting documents used in this research have been provided by a company, which creates a Virtual Private Network (VPN) solution to be used on Windows devices of B2C type customers.

### 2.2.1. Analysis of business needs

As the literature review has identified a lack of research when it comes to advanced RAG solutions, like Graph RAG, business needs provided by the company were analyzed to determine that the implementation of the need would be a sufficiently large project, to generate the required number of requirements and to have extensive supporting documentation.

Furthermore, a single business need had to be selected for the preparation of RAG knowledge base, to ensure the model cannot hallucinate by using supporting information of one business need to evaluate the requirement of a different one.

With this information in mind, the business area selected for the research was the introduction of Wireguard VPN protocol to be used by customers of the company, as such a business need requires:

1. Changes on the VPN server and its configuration in order to support the new protocol.
2. Client side implementation for the device to be able to establish a connection to the VPN server.
3. Changes to the Windows application to expose options for user to make a connection with the new protocol.

Documents supporting all of these steps have to be connected together in order for the project to succeed, making it the perfect business need for this experiment.

### 2.2.2. Collection of requirements

With business need now selected for research, the requirements that were used for implementation had to be selected. Requirements selected for the research had to be:

1. Written by a human specialist, with experience of working in the business area.
2. Supported by company documentation – technical or business-oriented.

The collected requirements were written by multiple specialists, with different amounts of experience working on the subject area (Table 9).

Table 9. Specialists, responsible for requirement creation.

| | Specialist 1 | Specialist 2 |
|---|---|---|
| Occupation | Product Owner | Engineering Manager |
| Experience | 2 Years | 4 Years |
| Business area | Client-facing application | Server infrastructure |

The first specialist was a person with two years in the Product Owner role and responsible for the preparation of business-oriented requirements for the development team. The person had less experience on the product, as well as less technical knowledge about VPN infrastructure and implementation.

The second specialist was an Engineering Manager, focusing significantly on the technical side and responsible for the definition of technical details of the implementation. This person had 4 years of experience in the position and was previously working on the server infrastructure in a hands-on position, leading to extensive technical knowledge, required for the implementation of the business need.

All collected requirements were formatted into a block of text, connected into a single list and then saved in JSON format.

#### 2.2.3. Supporting document preparation

In order to implement RAG knowledge base to be used for collected requirement quality validation, all of the documents describing Wireguard implementation have been provided by the company. This documentation was of two types:

1. Documentation prepared by company specialists, explaining the details of the implementation in their product.
2. External documentation from the official Wireguard documentation, that was used for the research into the potential implementation of the technology in the product.

Each of the collected documents then had to be analyzed further to determine whether it is supporting one of the collected requirements, or was simply part of the research process, but was no longer relevant in the implementation phase. All irrelevant documents were discarded, while the relevant ones were further refined to:

1. Remove tables, equations and pictures from the text.
2. Remove headings, footers and other formatting, that was applied to the documents.

Once the documents were refined to only have the text information, they were added into the document list to be used in the experiment. It is important to note that larger documents were also divided into smaller ones without losing their meaning in order to make the processing easier during the experiment phase.

## 2.3. Preparation of research environment

Once the subject area analysis has been completed, the next step was to prepare the environment,

in which the research will be carried out (Figure 9). This entails the selection of a database provider for Graph RAG implementation as well as setting the parameters for the knowledge graph, selection of an LLM provider for the validator part of the method, analysis and implementation of evaluation metrics, as well as the implementation of the method itself.

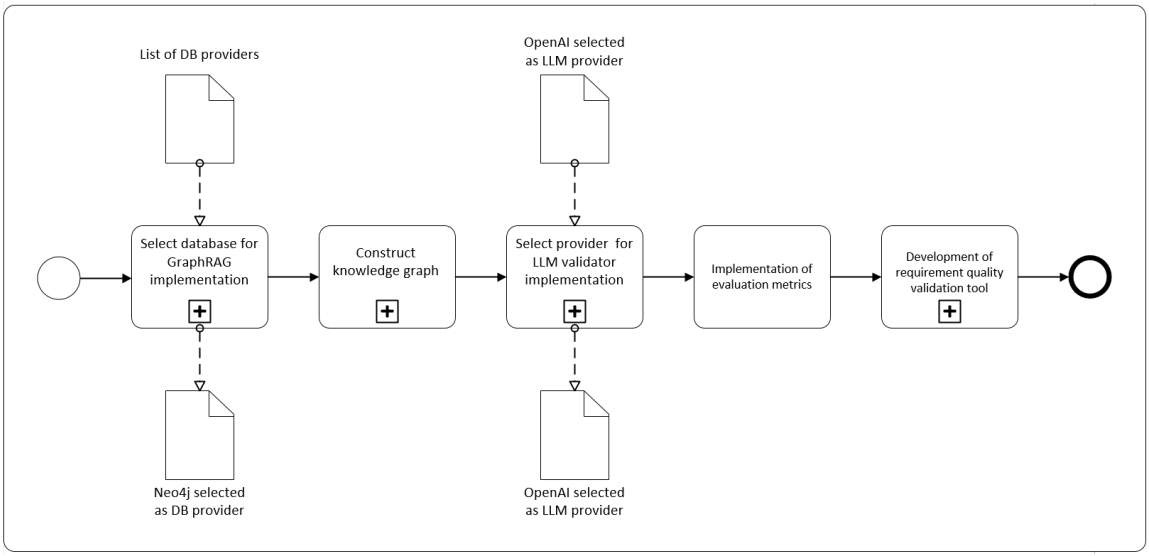


Figure 9. Process for preparation of research environment

### 2.3.1. Knowledge graph construction

The core part of the requirement quality validation method is the knowledge graph used by the Graph RAG implementation to provide requirement context for the validator, allowing to make a more precise and grounded decision on requirement quality. The key part of this knowledge graph is the graph database, which can be implemented via a list of potential database providers. Due to this reason, the first step of preparing research environment was to choose a knowledge base database provider and write the code to set it up for use in the research.

A list of potential graph database providers was collected, which was then analyzed based on two criteria (Table 10):

1. Is there extensive support for the use of the database in Python programming language?
2. Is there support for hosting and accessing the database on a local device (via CLI or client application)?

Table 10. Comparison table of analyzed database providers for Graph RAG knowledge base

| Provider name | Short description | Python support | Options for accessing the database |
|---|---|---|---|
| Neo4j | Popular property-graph database using Cypher, widely adopted for | Official Python driver | Official drivers, CLI, Browser UI, Bolt/HTTP |

| | | | |
|---|---|---|---|
| | knowledge graphs and Graph RAG. | | APIs |
| Memgraph | High-performance in-memory/disk property-graph DB with Cypher compatibility. | Python via GQLAlchemy | Official drivers, CLI, Memgraph Lab UI |
| ArangoDB | Open-source multi-model database (document, graph, key-value). | Supported through python-arango package | Web UI, CLI, HTTP/REST APIs |
| JanusGraph | Open-source distributed graph DB built on TinkerPop/Gremlin. | Supported through Gremlin-Python package | Gremlin Server, drivers, CLI tools |
| Dgraph | Distributed graph database with GraphQL and DQL query support. | Supported through pydgraph package | GraphQL API, CLI |

After analyzing the potential providers, Neo4j was selected as the one to be used in the research due to it being the most mature and with comprehensive support in Python ecosystem, with official drivers and integrations existing with popular RAG frameworks. Furthermore, it is purpose-built for knowledge graph implementation, while its property-graph model and Cypher query language make it an especially effective solution for entity-centric retrieval, relationship traversal and context expansion. Lastly, it also supports hybrid retrieval solution, that combines native graph traversal with built-in vector search.

In order to construct the knowledge graph using Neo4j, the structure of the graph had to be determined based on the parameters of the task (Figure 10). As already mentioned in the retrieval augmented generation types section of this paper, the lexical graph with extracted entities was selected, as it allows to maintain the structure of the document, while also allowing to connect different, but related documents through the extracted entity relation.

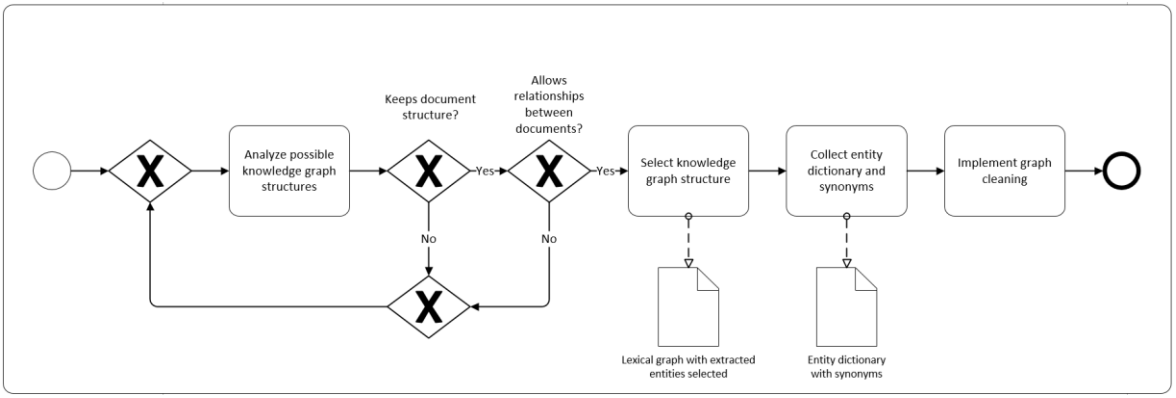


Figure 10. Process for construction of the knowledge graph

Using this structure, entities can be extracted manually by a trained specialist, automatically by an LLM tool or semi-automatically, where both approaches are used in the process. For this experiment, the semi-automatic approach was selected, where entity information was provided by the company, supplying the business needs and requirements, which was then enriched by synonyms and additional entities that the LLM was able to detect in the knowledge base documents.

As such an approach can lead to the extraction of less relevant entities, additional focus was put into developing the ability to remove entities that occur only once in the knowledge graph database.

### 2.3.2. Selection of LLM models

The LLM in this research will have to be responsible for two actions in total:

1. Entity extraction in preparation of the knowledge graph.
2. Validation of requirement quality.

As the two actions differ in complexity, separate models can be used to improve efficiency and cost-effectiveness. In preparation for the research, first a list of LLM providers and their models was selected and each of them analyzed to determine whether they are applicable for use in the experiment by checking Python support and availability of the model for non-enterprise usage (Table 11).

Table 11. List of analyzed LLM providers

| Provider | Python support | Model availability |
|---|---|---|
| OpenAI | Official openai SDK | Access to GPT-5, GPT-4, GPT-3, as well as embedding models |
| Anthropic | Official anthropic SDK | Access to the latest Opus 4.7 and 4.6, Sonnet 4.6 and Haiku 4.5 models |
| Google | Official unified google-genai SDK | Access to Gemini 3.x family, as well as Gemini 2.5 Pro, Flash, Flash-lie and |

| | | |
|---|---|---|
| | | image models |

Due to the popularity of LLM usage in Python programming language, all of the analyzed tools had official support via a provider-made SDK and allowed registration for API usage in self-made tools. Having this in mind, greater focus was put into analyzing the available models for use as well as taking into account which providers are more focused in existing research. Consequently, OpenAI was selected for use in the experiment due to it providing access to more models, including models designed explicitly for embedding task, as well as this family of models being more heavily focused by existing researchers and having better understood capabilities, which helps the research to focus more heavily on experimentation with the Graph RAG implementation.

### 2.3.3. Evaluation metrics

In order to evaluate the performance of proposed method, three evaluation categories were used – RAG, graph quality and gold standard comparison.

Gold standard comparison compares the results of the method against the results generated by a human specialist at the same task, with the same available context information.

RAG metrics evaluate the performance of the Graph RAG implementation when performing the task. These metrics are evaluated both for cases when the model successfully evaluates the requirement and for cases when it does so unsuccessfully, in order to determine patterns that lead to model‘s decision.

Lastly, the graph quality metrics – node degree, graph density and graph connectivity – showcase how well the graph database was constructed in order to perform the requirement quality validation task.

### 2.3.4. Development of requirement compliance validation tool

Once the needed parts of the tool – knowledge graph for Graph RAG implementation, LLM and evaluation metrics – were selected, the development of the tool that will be used in experiments was started (Figure 11).

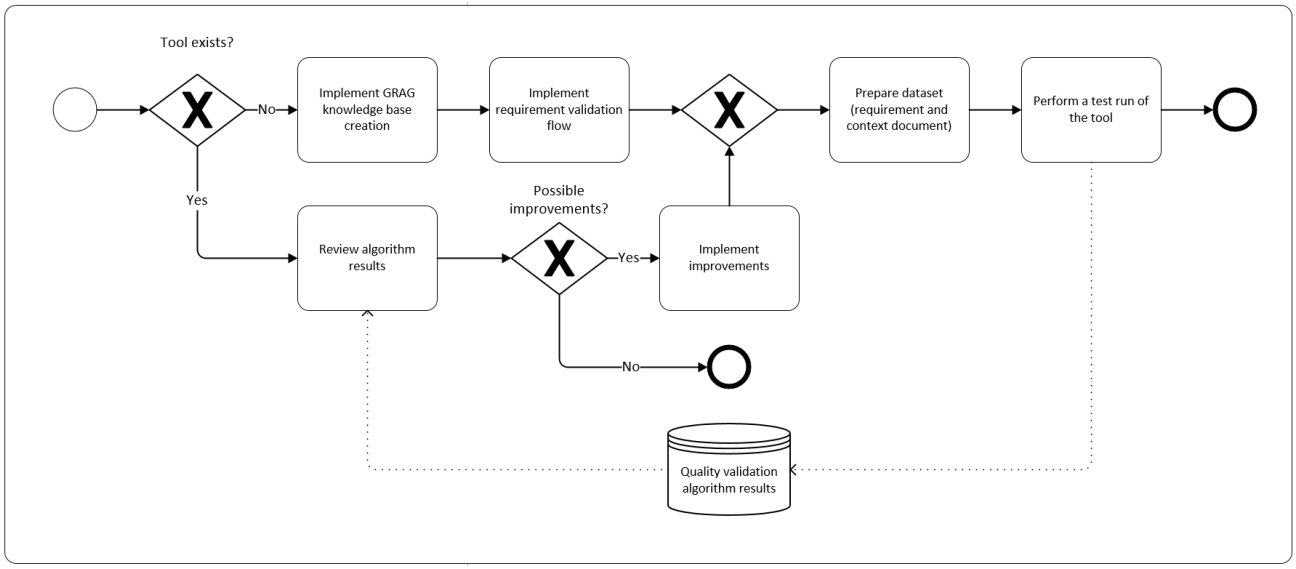


Figure 11. Process of developing requirement quality validation tool

First of all, the Graph RAG knowledge base was implemented, which took the provided supporting document, divided into chunks and built the knowledge base out of this information. These chunks were then transformed into graph database nodes, connecting them to one another, and with the original document node. Once these initial nodes were created, the automatic entity extraction process started, which uses Cypher query language to query existing text chunk nodes and uses an LLM to extract entities mentioned in them. Afterwards, extracted entities were validated against the provided dictionary of entities and their synonyms before the complete list was embedded into the database as Entity type nodes, connected with Chunk nodes, that mentioned them (Table 12).

Table 12. Graph database nodes and relationships

| Node type | Properties | Relationships |
|---|---|---|
| Document | Name | PART_OF_DOCUMENT with Chunk nodes |
| Chunk | Embedding, text | PART_OF_DOCUMENT with Document node, NEXT_CHUNK with other Chunk nodes, HAS_ENTITY with Entity nodes. |
| Entity | Text, variants (of the text expression) | HAS_ENTITY with Chunk nodes |

With these steps performed, a graph database was created, allowing to search for supporting text chunks by both directly traversing the graph database, or by using mentioned entities to find distant text chunks, that would also support the requirement. Furthermore, embedding expressions were calculated for each text chunk to allow for vector index search, next to the already available full text search. For context retrieval, a Hybrid Retriever was used, which combines traditional keyword search

with vector search, using the embedding calculated for each Chunk type node.

Once knowledge base setup was implemented, initial prompt technique templates were prepared to be used in the experiment. The prompting techniques defined for this task were:

1. Zero-shot.
2. Few-shot.
3. Chain-of-Thought.
4. Chain-of-Thought few-shot.
5. Tree-of-thought.

The validation tool is programmed in such a way, that it first accepts documents for the preparation of knowledge base, using the previously explained logic. Once knowledge base is created, it accepts one or multiple requirements in JSON format, requests context to be retrieved from the knowledge base based on the requirement text, applies a pre-defined prompting technique, combining context, requirement and instructions for the LLM in order to prepare quality assessment result. Once the result is generated, evaluation metrics are calculated and tool begins the assessment of another requirement, if more than one was passed for assessment. Each result consists of the following information in JSON format:

1. Experiment run name.
2. Experiment configuration, explaining the setting of each configurable parameter.
3. Setup time for the database in seconds.
4. Evaluation time of all requirements in seconds.
5. Total experiment run time in seconds.
6. An Array of results for each individual requirement, noting the requirement, evaluated criteria, evaluation reasoning, final score, context used to deliver the score and evaluation metrics together with the evaluation time for this requirement (in seconds, combining evaluation time for all three criteria).

## 2.4. Execution of the experiment

Execution of the experiment uses the previously described setup of the requirement validation tool together with the collected requirements and supporting documents dataset. Using this information, a set of experiment runs was prepared, each time evaluating the impact of a selected parameter on the results of the tool (Figure 12). The results of each run were saved in separate JSON files and analyzed after each run, comparing with previous test results to determine the impact of the changed parameter.

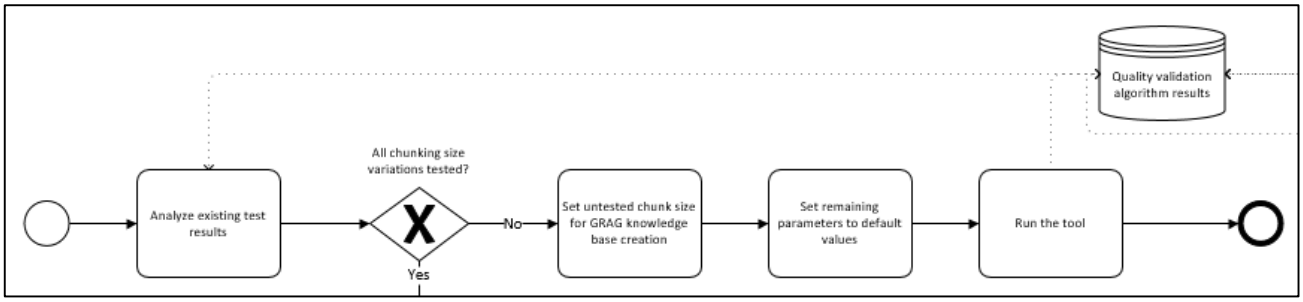


Figure 12. Part of the experiment execution process

The tool allows for configuration of knowledge base creation parameters, as well as the parameters of how the LLM prepares the responses (Table 13).

Table 13. A list of configurable parameters in the requirement quality validation tool

| Configurable parameter | Description | Baseline value | Available values |
|---|---|---|---|
| Chunk size | The maximum amount of characters that will be allowed in a single chunk of text, extracted from the document. Due to the chunk size limit, new sentences won't be included into the chunk if adding it will exceed the limit. | 1000 | 250, 300, 350, 400, 450, 500, 600, 700, 800, 900, 1000 characters |
| Overlap | The amount of characters taken from the end of the previous chunk and added to the next one. | 200 | 80, 100, 150, 200 characters |
| new_after_n_chars | Sets a preferred maximum limit to the text chunk. With this setting enabled, method attempts to create text chunk closer to the given value, instead of defined chunk size. | 800 | 600, 800, 1000 characters |
| top_k | Indicates the amount of context chunks that will be returned by the method. If top_k parameter exceeds the amount of context found, only the relevant number of found context will be returned. | 5 | 3, 4, 5, 6, 7, 8, 10 |
| Depth | Defines the depth of relationships that | 1 | 1, 2, 3, 4 |

| | | | |
|---|---|---|---|
| | the method will explore when traversing graph database in order to collect relevant context. | | |
| Prompt technique | Defines the instructions provided to the LLM model that control how LLM will decide on the final answer and how the answer will be structured. | Tree of Thought | Few shot, Chain of Thought, Tree of Thought |
| Dictionary | Tells method whether to use an in advanced created dictionary file with synonyms instead of only using entities model is able to find in the context documents. | Path to dictionary provided | path to the dictionary file or empty |
| Single occurring entities | Defines whether method removes entities, that have connection to only a single text chunk or not. | false | false, true |
| Extraction of parameter values | Defines whether method uses LLM to extract parameter and value pairs from the context documents to be included in the graph database | false | false, true |

A single experiment run is a case, where a single parameter is changed to the next untested value, while all other parameters are kept as baseline in order to evaluate the impact of the change. Once all options of the parameter are tested, the best performing option is selected for the following runs and later experiment runs test the impact of other, yet untested parameters to the results. This iteration of the process is performed until all of the available experiment setups are tested and final results analysis can be carried out in order to determine the best performing configuration for requirements quality validation task.

## 2.5. Results analysis

The collection of requirements used in the experiment consists of 31 requirements in total (Table 14), all saved in JSON format, which is used to pass them to the developed validation method. These requirements are supported by 5 context documents, which have all been formatted to exclude illustrations, tables or other objects, that are not currently supported by the method. Additionally, a dictionary file provided together with the documents has been formatted into a JSON format,

containing the concept and synonyms for it, as provided by the company.

Table 14. REQ-5 requirement used in the validation method experiments

| Requirement ID | Requirement |
|---|---|
| REQ-5 | The server private key is generated using Curve25519 and store it in server.key path. The file permissions should allow only the owner to read or modify the file, while other users should have no access to the key for security. The corresponding public key should be exposed for client provisioning and configuration files are readable not only from root user. |

As answers provided by the LLM can differ even while all of the parameters in experiment iterations are identical, each iteration was repeated twice and results were analyzed taking the cumulative results of both runs into consideration.

The baseline iteration required 73 seconds in order to set up the database and 213 seconds to carry out the validation of every requirement in the dataset, with graph structure presenting a fully connected, but sparse graph, where each node is connected to roughly two other nodes, leading to a chain-like structure of the graph, expected for focused technical documentation (Table 15).

Table 15. Baseline iteration graph connectivity metrics

| Nodes | Edges | Average Degree | Density | Connectivity | Setup (s) |
|---|---|---|---|---|---|
| 821 | 1636 | 1.99 | 0.005 | 1 | 73 |

Afterwards, model results were evaluated against the gold standard metrics – precision (Equation 7), recall (Equation 8) and F1 score (Equation 9).

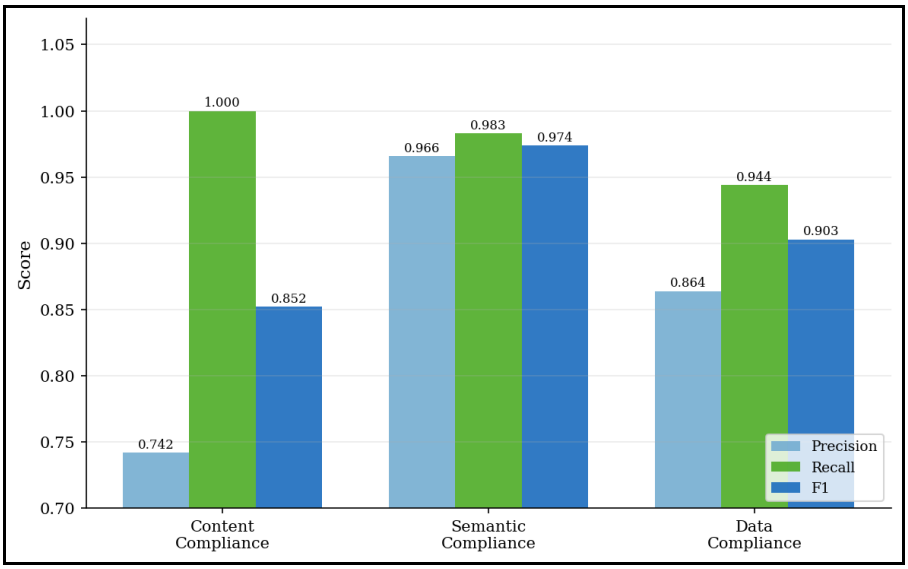


Figure 13. Baseline iteration impact on gold standard metrics

These results indicate the model's tendency towards stricter evaluation (Figure 13). For

example, the recall score for content compliance, most reliant on the Graph RAG implementation, was 1, meaning that the model never showcased a false negative – falsely evaluating a requirement as compliant – in the entire iteration. As this behavior was observed in other iterations, further analysis was performed focusing on the F1 Score, which provides a better insight into both false negative and false positive results of the model, hence providing a better overview.

Lastly, the RAG quality metrics – Faithfulness (Equation 1), Answer Relevance (Equation 2) and Context Relevance (Equation 3) – indicate that around 72% of all reasoning claims are grounded in retrieved context, but other metrics indicate the method‘s tendency to retrieve topically related, but not always tightly scoped content, which combined with verbose Tree of Thought technique produces broad, rather than requirement-specific evaluations (Table 16).

Table 16. Baseline iteration RAG quality metrics

| Faithfulness | Retrieval Quality | Answer Relevance | Context Relevance |
|---|---|---|---|
| 0.718 | 0.281 | 0.170 | 0.119 |

Firstly, iterations were carried out with the varying chunk size parameter. Evaluation of the F1 score indicates that these iterations do not have a significant impact on the results of the method – content compliance remains unchanged in each iteration, while data and semantic compliance fluctuate, but within a margin of error (Figure 14).

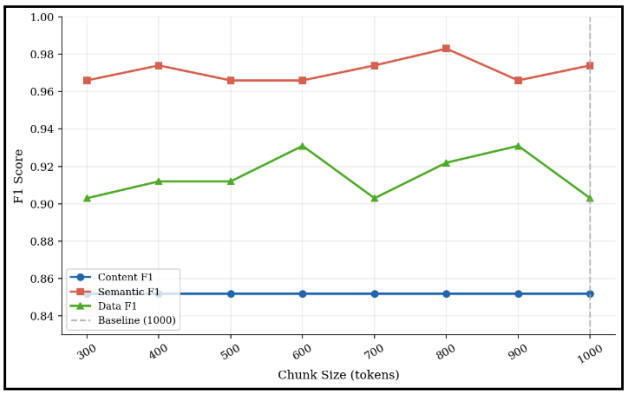


Figure 14. Chunk size impact on F1 score

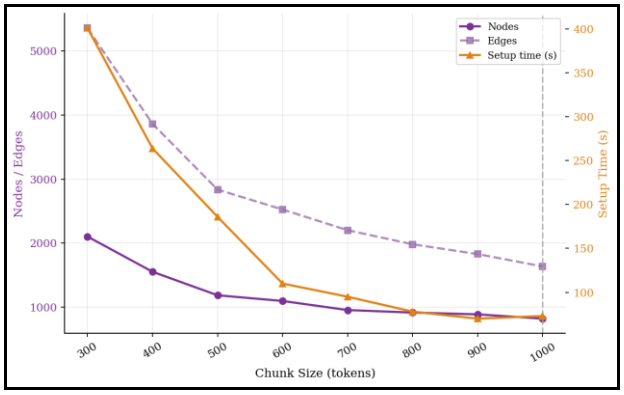


Figure 15. Chunk size impact on graph construction

However, these iterations have a significant impact when it comes to graph construction performance (Figure 15) – reducing chunk size exponentially increases graph setup time and also leads the graph to be less connected, in the case of 300 characters chunk size even leading to some nodes being entirely unreachable due to how disconnected the graph becomes.

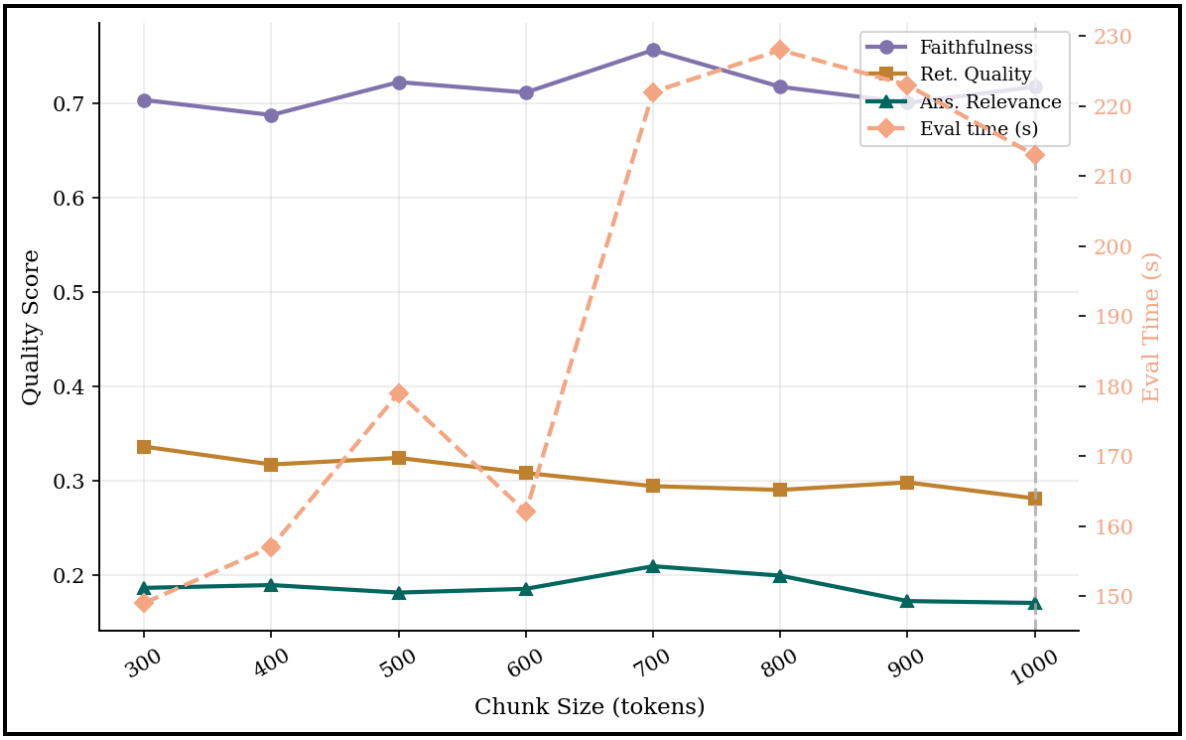


Figure 16. Chunk size impact on RAG quality metrics

However, during the same iterations, as chunk size decreases, the retrieval quality metric improves, but this does not translate into a better F1 score for the method. Faithfulness follows a different trend (Figure 16), reaching its overall best performance at chunk size of 700, suggesting that this size produces nodes, whose content most cleanly maps to reasoning statements, even if not translating to better F1 score and a similar insight can be made when it comes to answer relevance.

Overlap iterations show a significant improvement when it comes to semantic and data compliance F1 score, with 150 characters overlap showcasing the overall best results (Figure 17). Content compliance score again remains flat. Further investigation into the method indicates that this is caused by the model's strict evaluation criteria, which require the requirement to be fully supported by the corpus, leading the method to never score a requirement as fully content compliant (due to even minor differences from the context information). Analyzing the results further, the retrieval quality metrics indicate a minor or no improvement when compared to the benchmark (Figure 18).

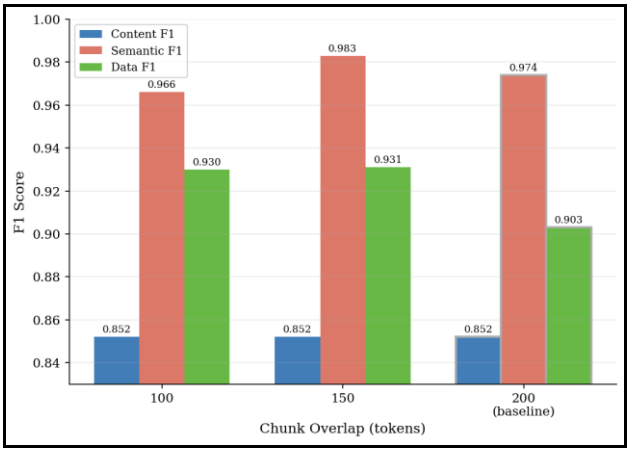


Figure 17. Overlap impact on F1 score

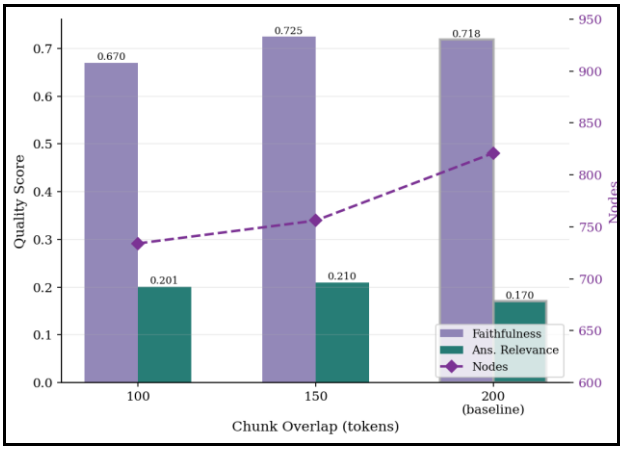


Figure 18. Overlap impact on RAG quality metrics

While new_after_n_characters iterations do not showcase a significant improvement over the baseline results in any metrics category, this change appears to have a significant impact on evaluation time, significantly increasing when the parameter is reduced (Figure 19). This observation can be explained by reduced chunk size in the case of this experiment, which leads the method to pull more neighbor chunks during a context search query, in turn leading to longer evaluation times.

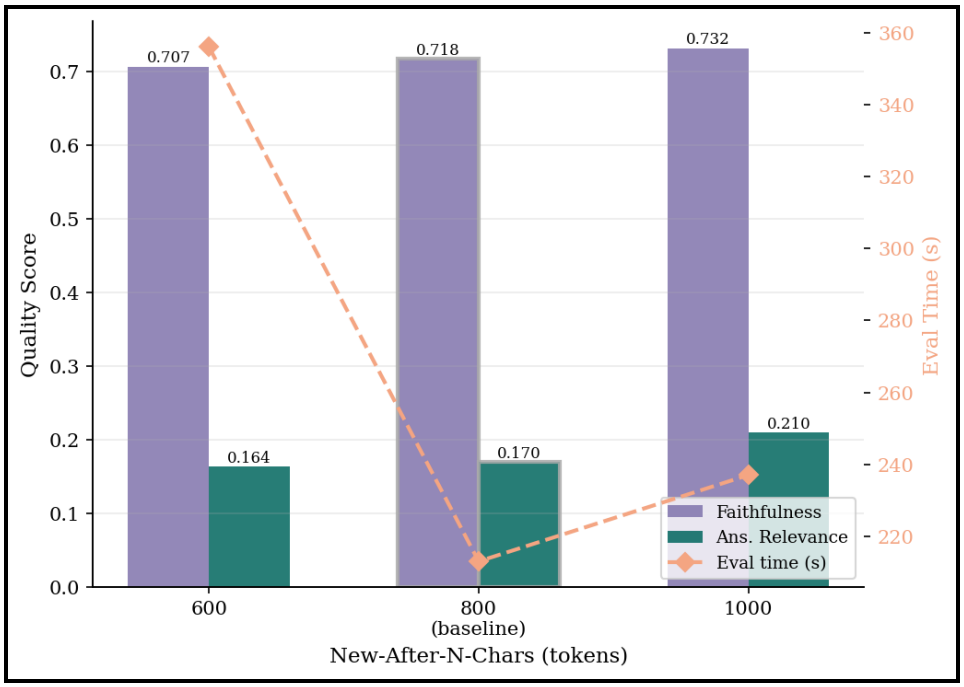


Figure 19. new_after_n_characters impact on evaluation time

Afterwards, graph construction iterations – expanded synonym file, single occurring entities removal and parameter extraction from context – were carried out. Parameter value extraction iteration showcases the best improvement to data and content compliance F1 scores and is the first test iteration

to push content compliance above 0.852 score reached in the baseline experiment, with a marginal trade-off in answer and context relevance metrics (Figure 20).

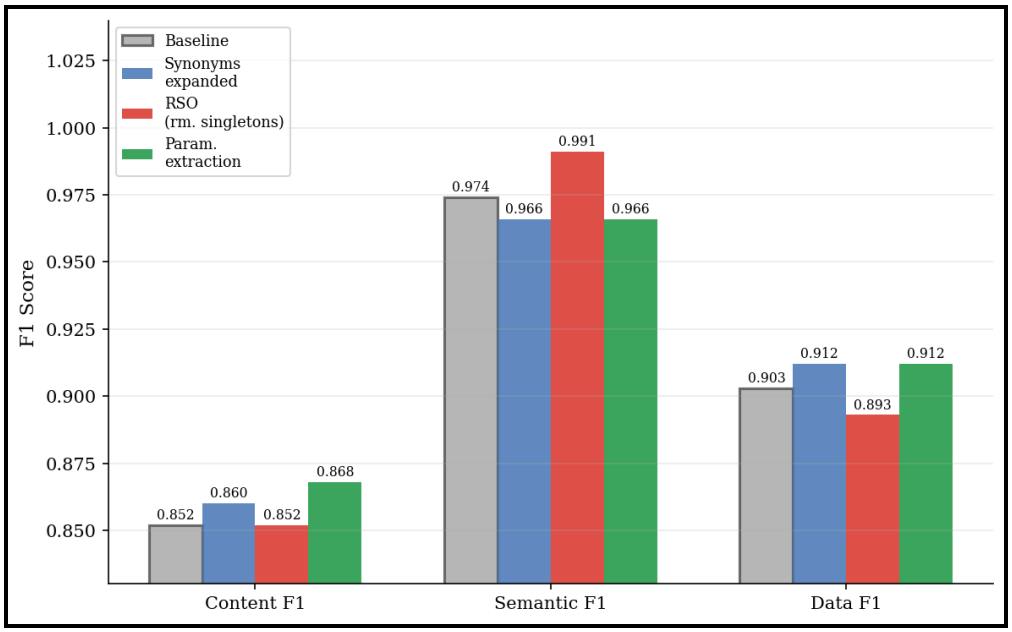


Figure 20. Impact of graph construction parameters on F1 score

Single-occurring entities removal is the iteration that has the biggest impact on semantic compliance F1 score, but worse results in other metrics, leading to a highly concentrated overall impact. Lastly, the expanded synonyms file, in which additional entities were manually added after a separate review of context documents, showcases a modest improvement in content compliance, without any meaningful trade-offs as indicated by other observed metrics – presenting this parameter as a safe, but low-impact improvement.

Afterwards, the experiment focus shifted towards analyzing the impact of context retrieval metrics – search depth and top-k. Search depth metric showed a visible improvement to method accuracy until depth of 3 layers into the graph database (Figure 21). Increasing the metric further reduces the accuracy, leading to the conclusion that further increases in depth only move the method away from relevant content. While RAG quality metrics do not show any significant changes during these iterations (Figure 22), evaluation time showed a significant increase as the search depth increased, indicating excess time spent in the context retrieval step.

Top-k metric showed best performance according to F1 score when set to 4 (Figure 23), where improvement to semantic compliance was observed with a slight decrease in data compliance. In the case of all other parameter values either negative or neutral results were observed – RAG quality metrics indicated marginal difference (Figure 24), while evaluation time steadily increased, reaching the highest point with top-k set to 7, where requirement evaluation took 340 seconds, compared to

230 seconds in top-k setting of 4.

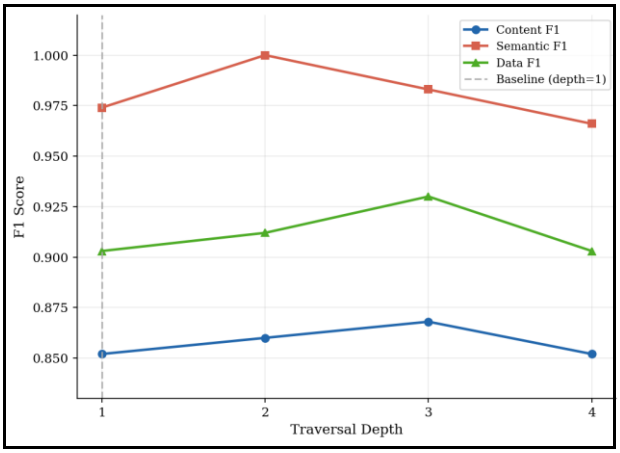


Figure 21. Impact of search depth on F1 score

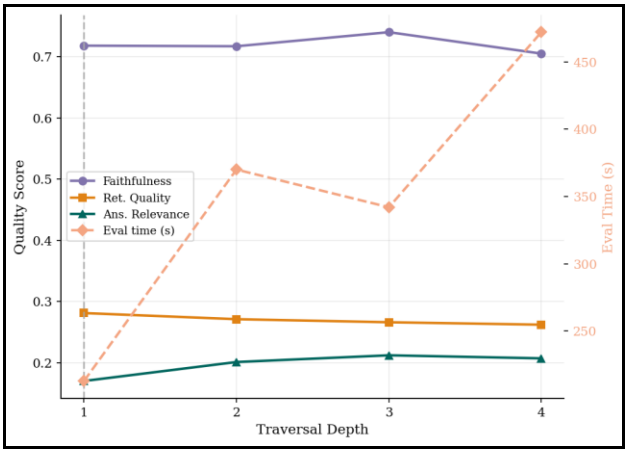


Figure 22. Impact of search depth on RAG quality metrics

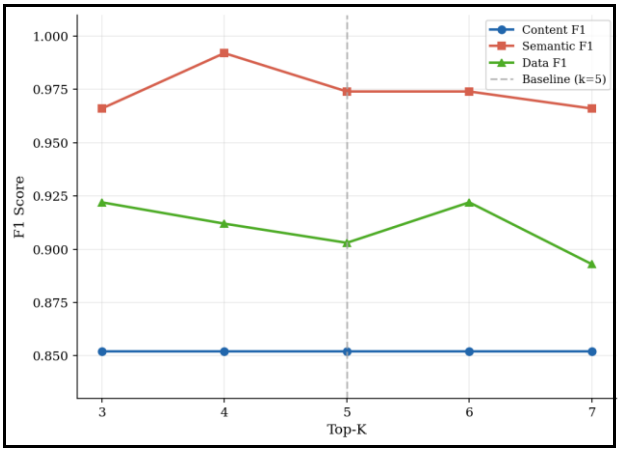


Figure 23. Impact of top-k on F1 score

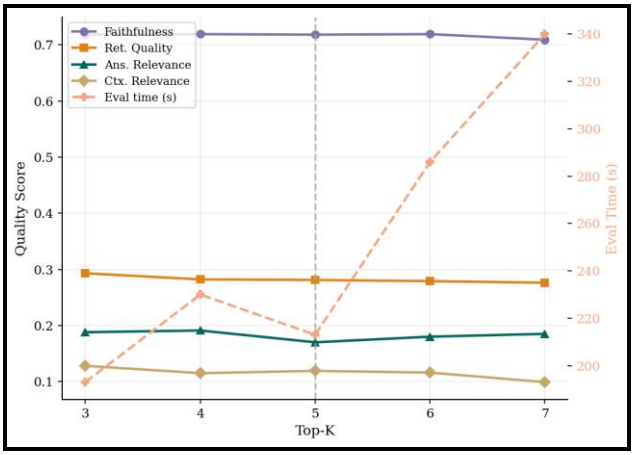


Figure 24. Impact of top-k on RAG quality metrics

Lastly, the impact of different prompting techniques was analyzed. During this iteration, the few-shot technique managed to reach a perfect F1 score in semantic compliance and above-average results for content and data compliance, while the zero-shot technique showed the best data compliance results (Figure 25), which indicates that more complex prompting techniques are pushing the model even further towards stricter evaluation and potential for false positive evaluations.

After evaluating these results, a new iteration was prepared and carried out with a method setup that appeared as the potentially optimal configuration. The optimal parameter values for this configuration were selected based on the best results showcased in all of the previously discussed experiment iterations, choosing values that improve the most underperforming F1 scores, deliberately putting less focus into evaluation and database build times. Because of this, the iteration changed chunk size to 700, top-k to 6, search depth to 3 and the prompting technique to zero-shot, while other

parameters were left in their baseline configuration.

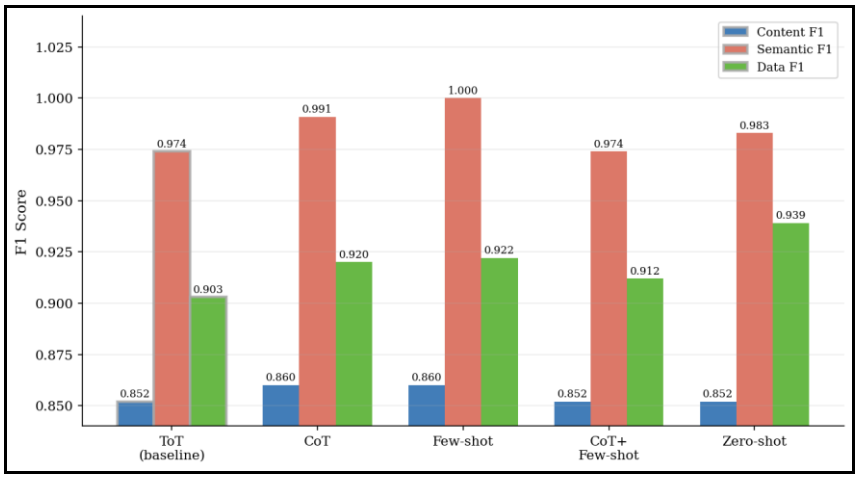


Figure 25. Impact of prompting techniques on F1 score

In this setup, the model sacrifices faithfulness and context relevance in order to gain improvements in answer relevance, semantic, and data compliance F1 scores (Figure 26). Considering these results, a conclusion can be made that the configuration allows the method to use broader context effectively for requirement quality validation, while also grounding its reasoning less tightly in specific retrieved passages.

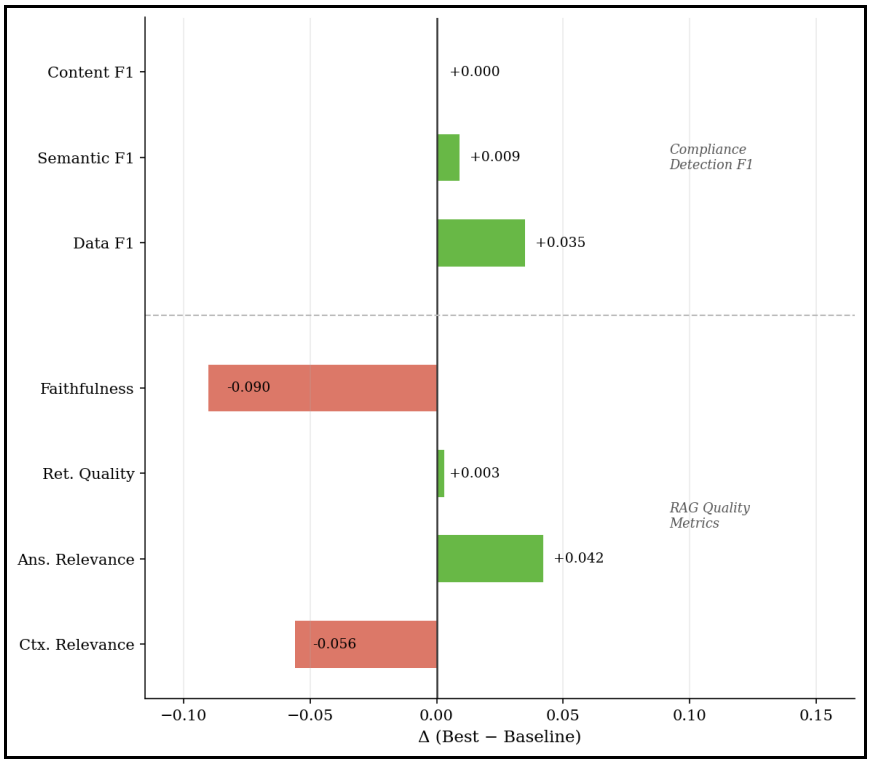


Figure 26. Winning configuration results comparison against baseline configuration

## Results

This section provides a summary of all the experiment iteration results, discussed in detail in the previous sub section, as well as conclusions and proposal for future work.

## Results summary

Summarizing the results of the proposed Graph RAG-based requirements compliance validation method, it is important to note that the content compliance F1 score is the least affected by all of the iterations, showing improvements only in Chain-of-Thought and Few-shot prompting techniques, and the expanded synonyms list iterations.

A similar observation can be made for all graph construction metrics, which were only impacted by the reduction of chunk size to 400 and 300 characters, in both of which cases the graph became disconnected.

With this in mind, key improvements on the method's performance can be observed in semantic and data compliance evaluation metrics, as well as faithfulness and answer relevance retrieval metrics (Table 17). Semantic compliance evaluation is impacted the most by Few-shot prompting technique, top-k value of 4 and removal of single occurring entities, while data compliance performs the best with Zero-shot prompting technique and new_after_n_characters parameter set to 1000 characters. While faithfulness and answer relevance metrics do not translate to direct improvement in semantic or data compliance evaluation, improvements in these metrics allow the method to provide better reasoned and grounded answering. The winning configuration, as mentioned in the previous sub-section, is not a sum of individual improvements seen across other experiment iterations, but rather a compromise, allowing to maximize improvements to semantic and data compliance, at the cost of faithfulness metrics.

Table 17. Results summary of key experiment iterations.

| Configuration | Semantic Compliance F1 | Data Compliance F1 | Faithfulness | Answer Relevance |
|---|---|---|---|---|
| Baseline | 0.9744 | 0.9027 | 0.7177 | 0.1701 |
| chunk=300 | 0.9655 | 0.9027 | 0.7035 | 0.1856 |
| chunk=700 | 0.9739 | 0.9027 | 0.7566 | 0.2085 |
| overlap=150 | 0.9825 | 0.9310 | 0.7254 | 0.2098 |
| nac=600 | 0.9825 | 0.9123 | 0.7073 | 0.1641 |
| nac=1000 | 0.9831 | 0.9217 | 0.7320 | 0.2104 |
| Synonyms expanded | 0.9655 | 0.9123 | 0.7111 | 0.1882 |

| | | | | |
|---|---|---|---|---|
| Removal of single occurring entities | 0.9913 | 0.8929 | 0.6919 | 0.1971 |
| Parameter value extraction | 0.9655 | 0.9123 | 0.7230 | 0.1880 |
| depth=3 | 0.9825 | 0.9298 | 0.7403 | 0.2121 |
| top-k=4 | 0.9915 | 0.9123 | 0.7194 | 0.1912 |
| Few-shot prompting technique | 1.0000 | 0.9217 | 0.6390 | 0.1683 |
| Zero-shot prompting technique | 0.9831 | 0.9391 | 0.6931 | 0.2300 |
| Winning configuration | 0.9831 | 0.9381 | 0.6281 | 0.2124 |

## Conclusions and future work

The proposed Graph RAG-based requirements compliance validation method achieves F1 scores of up to 0.852 (content), 0.974 (semantic), and 0.912 (data) in its optimal configuration, demonstrating its viability as a practical tool for automated requirements screening. Across all experiment iterations, the method consistently exhibited a conservative evaluation bias, marking borderline-passing requirements as intermediate or failing rather than as fully compliant. This positions the approach as a screening tool that surfaces requirements warranting further human review, rather than as a fully autonomous arbiter of requirement compliance.

The optimal configuration – chunk size 700, top-k 6, search depth 3, zero-shot prompting – was selected as the setup that achieved the best balance across all three compliance dimensions simultaneously. This configuration represents a deliberate trade-off: the method sacrifices faithfulness and context relevance to gain improvements in answer relevance and stronger semantic and data compliance F1 scores, indicating that broader context utilization yields more reliable compliance judgments than tightly scoped retrieval. Content compliance remained stable at approximately 0.852 across all iterations regardless of parameter changes, and was therefore not a differentiating factor in configuration selection.

More broadly, the experiments revealed that the method is highly sensitive to context retrieval and prompting parameters, while remaining relatively robust to graph construction parameters such as chunk size and overlap. The most significant improvements were achieved by enabling parameter value extraction during graph construction, setting search depth to 3 layers, and applying the zero-shot prompting technique. More complex prompting techniques pushed the model towards even stricter evaluation and increased the potential for false positive results.

These findings point to several directions for future research. The method's sensitivity to retrieval parameters suggests that automated configuration tuning could replace manual iteration when applying the method to new datasets. Additionally, evaluating the approach across a broader range of domains and requirement representation styles would strengthen generalizability claims, while directly addressing the conservative bias would be a necessary step towards presenting the method as a fully autonomous validator of requirement compliance.

## References and sources

[IIB06] analysis, International institute of business. 2006. *A guide to the business analysis body of knowledge, version 1.6.* 2006.

[AP22] Anis R. Amna, Geert Poels. 2022. *Systematic Literature Mapping of User Story Research.* 2022.

[ARC22] Arcas, B. A. y. 2022. Do large language models understand us? *Do large language models understand us?* s.l. : Daedalus , 2022, pp. 183–197.

[MMM+24] Arsalan Masoudifard, Mohammad Mowlavi Sorond, Moein Madadi, Mohammad Sabokrou, Elahe Habibi. 2024. *Leveraging Graph-RAG and Prompt Engineering to Enhance LLM-Based Automated Requirement Traceability and Compliance Checks.* 2024.

[KBB+09] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, Stephen Linkman. 2009. *Systematic literature reviews in software engineering – A systematic literature review.* s.l. : ICSE04, 2009.

[BEN12] Bender, Emily M. 2012. *On the Dangers of stochastic parrots: can language models be too big?* 2012.

[PZL+24] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, Siliang Tang. 2024. *Graph Retrieval-Augmented Generation: A Survey.* 2024.

[JYH+24] Bowen Jin, Jinsung Yoon, Jiawei Han, Sercan O. Arik. 2024. *Long-Context LLMs Meet RAG: Overcoming Challenges for Long Inputs in RAG.* 2024.

[RSR24] Chidaksh Ravuru, Sagar Srinivas Sakhinana, Venkataramana Runkana. 2024. *Agentic Retrieval-Augmented Generation for Time Series Analysis.* 2024.

[AK17] *Comparison of Stochastic and Rule-Based POS Tagging on.* Kalaiarasi Sonai Muthu Anbananthen, Jaya Kumar Krishnan,. 2017. 2017, American Journal of Applied Sciences.

[JCH+24] Dian Jiao, Li Cai, Jingsheng Huang, Wenqiao Zhang, Siliang Tang, Yueting Zhuang. 2024. *DuetRAG: Collaborative Retrieval-Augmented Generation.* 2024.

[BRB+23] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, Robert Feldt. 2023. *Challenges and Practices in Aligning Requirements with Verification and Validation: A Case Study of Six Companies.* 2023.

[TSG20] Group, The Standish. 2020. *CHAOS Report Beyond Infinity (digital version).* Boston : s.n., 2020.

[LZL25] Hang Luo, Jian Zhang, Chujun Li. 2025. *Causal Graphs Meet Thoughts: Enhancing*

*Complex Reasoning in Graph-Augmented LLMs.* 2025.

[NKQ+23] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, Ajmal Mian. 2023. *A Comprehensive Overview of Large Language Models.* s.l. : Cornell University, 2023.

[IEE83] IEEE. 1983. *IEEE 729-1983.* s.l. : IEEE, 1983.

[IEE22] —. 2022. *ISO/IEC 22989:2022.* s.l. : IEEE, 2022.

[IEE18] —. 2018. *ISO/IEC/IEEE 29148:2018.* s.l. : IEEE, 2018.

[INC23] INCOSE. 2023. *Guide to Writing Requirements.* 2023.

[ABA+21] Issa Atoum, Mahmoud Baklizi, Izzat Alsmadi, Ahmed Ali Otoom, Taha Alhersh, Jafar Mohammad ali Ababneh, Jameel Almalki, Saeed Alshahrani. 2021. *Challenges of Software Requirements Quality Assurance and Validation: A Systematic Literature Review.* s.l. : IEEE, 2021.

[PS12] James Pustejovsky, Amber Stubbs. 2012. *Natural Language Annotation for Machine Learning.* Sebastopol : O'Reilly Media, Inc., 2012.

[WWS+22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, Denny Zhou. 2022. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.* s.l. : Cornell University, 2022.

[CHW+22] Ken Cheligeer, Jingwei Huang, Guosong Wu, Nadia Bhuiyan, Yuan Xu, Yong Zeng. 2022. *Machine learning in requirements elicitation: a literature review.* 2022.

[RSW+24] Lasse M. Reinpold, Marvin Schieseck, Lukas P. Wagner, Felix Gehlhoff, Alexander Fay. 2024. *Exploring LLMs for Verifying Technical System Specifications Against Requirements.* 2024.

[ZAF+22] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo. 2022. *Classification of Natural Language Processing Techniques for Requirements Engineering.* s.l. : Cornell University, 2022.

[KGV24] Madhava Krishna, Bhagesh Gaur, Arsh Verma, Pankaj Jalote. 2024. *Using LLMs in Software Requirements Specifications: An Empirical Evaluation.* 2024.

[NAS23] NASA. 2023. *Appendix C: How To Write a Good Requirement.* 2023.

[ZAF+21] *Natural Language Processing for Requirements Engineering: A Systematic Mapping Study.* Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo, Muideen A. Ajagbe, Erol-Valeriu Chioasca, Riza T. Batista-Navarro. 2021. 2021, ACM Computing Surveys, pp. 1-41.

[LPP+20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela. 2020. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* s.l. : New

York University, 2020.

[NBG24] Rasoul Nikbakht, Mohamed Benzaghta, Giovanni Geraci. 2024. *TSpec-LLM: An Open-source Dataset for LLM Understanding of 3GPP Specifications.* 2024.

[NHB+21] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess. 2021. *WebGPT: Browser-assisted question-answering with human feedback.* 2021.

[SAM24] Sami, Malik Abdul. 2024. *Prioritizing Software requirements using large language models.* 2024.

[LFT+24] Sebastian Lubos, Alexander Felfernig, Thi Ngoc Trang Tran, Damian Garber, Merfat El Mansi, Seda Polat Erdeniz, Viet-Man Le. 2024. *Leveraging LLMs for the Quality Assurance of Software Requirements.* s.l. : Cornell University, 2024.

[EJE+25] Shahul Es, Jithin James, Luis Espinosa-Anke, Steven Schockaert. 2025. *Ragas: Automated Evaluation of Retrieval Augmented Generation.* 2025.

[YYZ+23] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, Karthik Narasimhan. 2023. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models.* 2023.

[SNV+24] Summra Saleem, Muhammad Nabeel Asim, Ludger Van Elst, Andreas Dengel. 2024. *Generative Language Models Potential for Requirement Engineering Applications: Insights into Current Strengths and Limitations.* 2024.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh. 2020. *Language Models are Few-Shot Learners.* s.l. : Cornell University, 2020.

[MMC97] Tom M. Mitchell, Ryszard S. Michalski, Jaime Carbonell. 1997. *Machine learning.* s.l. : Mc Graw Hill India, 1997.

[RFG+17] *Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain.* Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, Stefano Bacherini. 2017. 2017, International Working Conference on Requirements Engineering: Foundation for Software Quality.

[KOM+20] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, Wen-tau Yih. 2020. *Dense Passage Retrieval for Open-Domain Question Answering.* 2020.

[LZH+21] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, Jie Tang. 2021. *Self-supervised Learning: Generative or Contrastive.* 2021.

[LCM+24] Xinze Li, Yixin Cao, Yubo Ma, Aixin Sun. 2024. *Long Context vs. RAG for LLMs: An Evaluation and Revisits.* 2024.

[GXG+24] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, Haofen Wang. 2024. *Retrieval-Augmented Generation for Large Language Models: A Survey.* 2024.

[JLF+22] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Delong Chen, Wenliang Dai, Ho Shu Chan, Andrea Madotto, Pascale Fung. 2022. *Survey of Hallucination in Natural Language Generation.* 2022.

## Abbreviations

The abbreviations used in this paper are:

| | |
|---|---|
| LLM | *Large Language Model* |
| RAG | *Retrieval Augmented Generation* |
| GDPR | *General Data Protection Regulation* |
| HIPAA | *Health Insurance Portability and Accountability Act* |
| GRAG | *Graph Retrieval Augmented Generation* |
| NLP | *Natural Language Processing* |
| POS | *Part-of-Speech Tagging* |
| NER | *Named Entity Recognition* |
| API | *Application Programming Interface* |
| RAGAS | *Retrieval Augmented Generation Assessment* |
| VPN | *Virtual Private Network* |
| SEO | *Search Engine Optimization* |
| B2C | *Business to Customer* |