CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Vectorising Tabular Data for RAG and Vector Database Validation

A study of tabular data retrieval and anomaly detection methods against poisoning attacks in RAG

Master's thesis in Computer science and engineering

Eric Källman, Max Sjödahl

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Vectorising Tabular Data for RAG and Vector Database Validation

A study of tabular data retrieval and anomaly detection methods against poisoning attacks in RAG

Eric Källman
Max Sjödahl

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Vectorising Tabular Data for RAG and Vector Database Validation
A study of tabular data retrieval and anomaly detection methods against poisoning attacks in RAG
Eric Källman, Max Sjödahl

Supervisor: Mehrdad Farahani, MPDSC
Advisor: Adam Ek, AI Sweden
Examiner: Romaric Duvignau, MPCSN

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Vectorising Tabular Data for RAG and Vector Database Validation
A study of tabular data retrieval and anomaly detection methods against poisoning attacks in RAG
Eric Källman, Max Sjödahl
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Current Retrieval-Augmented Generation (RAG) systems are primarily optimised for unstructured text and lack mechanisms for ensuring data integrity in the knowledge base. This thesis investigates two challenges in the context of SVEA, AI Sweden's digital assistant for the Swedish public sector. Firstly, how structured tabular data should be represented for vector retrieval. Secondly, how a validation layer can protect the vector database against data poisoning attacks. The two attacks tested are prompt injections and hate speech.

Four table representation methods are evaluated on a corpus of 23,638 chunks from Swedish government and municipality documents. The methods are the following: Unmodified Tabular Extraction (UTE), Metadata-Enriched Representation (MER), Natural Language Summarization (NLS), and Summary-Augmented Hybrid Representation (SAHR). The results indicate that SAHR achieves the highest table *Recall*@1, *Recall*@3 and MRR (0.362, 0.496 and 0.445), while UTE achieves the highest *Recall*@10 (0.630). This shows a trade-off between ranking quality and retrieval coverage. In addition, the cost in terms of latency of generating summaries with an LLM must be considered.

Unsupervised anomaly detection using HDBSCAN and Isolation Forest is combined with pre-trained classifiers for prompt injection and hate speech detection as a potential validation layer. Using only classifiers achieves the highest $F1$-score (0.592 for prompt injection on Regeringen), while hybrid two-stage approaches reduce latency at the cost of reduced detection performance.

Keywords: Computer Science, RAG, Security, Information Retrieval, Performance, Classification, Machine Learning.

Acknowledgements

We would like to thank Adam Ek, our internal supervisor at AI Sweden, who continuously gave us support and guidance throughout the development of this thesis. Furthermore, we thank Mehrdad Farahani for helping with the writing of the report and supervising this thesis.

Eric Källman and Max Sjödahl, Gothenburg, 2026-08-07

Contents

List of Figures

List of Tables

1

Introduction

1.1 Introduction and Background

Large language models supported by Retrieval-Augmented Generation (RAG) provide new opportunities for data-grounded decision support, yet current retrieval pipelines are primarily optimised for unstructured text. In real-world deployments, information is also stored in complex tables that span multiple pages, mix textual and numeric fields, and rely on header continuity for interpretation. Regardless of how the data are represented in the documents, it may be a threat to the overall system. In a scenario where an attacker has gained access to upload documents to the system, it may try to attack the system by uploading malicious information that affects the retrieval and may represent dangerous data for the end-user. Therefore, it is crucial that the vector database that stores documents in an RAG system is properly validated before retrieving documents.

This thesis addresses these challenges by investigating how structured tables can be represented for vectorisation while preserving its semantic value, and how correctness and privacy can be strengthened through validation mechanisms applied directly at the level of the vector database. The work is motivated by the needs of SVEA, AI Sweden's digital assistant for the public sector. SVEA aims to enable public-sector workers to reliably search and converse over both public sources (e.g., laws, government reports) and internal sources (e.g., municipal guidelines, HR documentation). Achieving this requires not only an RAG system capable of handling complex tabular data, but also one that ensures that content is verified as secure.

Current RAG systems typically comprise three essential components that are listed here and can be seen in Figure 1.1:

- **Vector database/store**: stores embeddings of the documents or table segments that form the retrieval corpus.
- **Retriever**: performs similarity search between the user query embedding and stored embeddings to select relevant content.
- **Generator**: a large language model that synthesises the final answer based on the user query and retrieved evidence.

RAG systems offer two core benefits: the base model does not need to be retrained on domain-specific data, and grounding the generation process in real documents

reduces hallucinations and increases factual reliability.

However, deploying such systems in the public sector introduces complexities across the entire computing stack. At the software layer, the work involves embedding models, vector indexing strategies, table extraction routines, and metadata frameworks. At the middleware layer, the system must support multiple organisations simultaneously and enforce separation between user groups that share the same infrastructure. At the hardware layer, RAG must run efficiently on GPU-backed inference resources provided by SVEA. The resulting architecture is therefore a multi-tenant, distributed AI system whose correctness and security depend on careful systems-level design.

This thesis will investigate how different segmentation and representation strategies preserve table semantics during embedding and retrieval. Beyond representation, it introduces a second line of inquiry: validating the data's contents directly within the vector database. Instead of attempting to sanitise the data post-retrieval, the goal is to validate the data before it becomes retrievable offline, and thus not adding latency that affects the end-user's experience. This approach provides a practical and interpretable security layer that complements existing defensive methods and validates the data before and after retrieval. Furthermore, vector databases used in RAG pipelines are vulnerable to accidental data contamination as well as intentional poisoning attacks, making robust validation mechanisms essential for maintaining retrieval integrity and preventing leakage. Data poisoning and anomaly detection are both mentioned in the OWASP Top 10 for Agentic Applications 2026 [1], respectively, as a potential attacking threat and mitigation.

To summarise, we will explore how retrieval quality interacts with table structure and assess the effectiveness of embedding-space validation as a safeguard against data poisoning. Together, these contributions aim to improve the robustness, reliability, and security of RAG systems used in multi-tenant public-sector environments.

1.1.1 Information Retrieval Tabular Data with RAG

Recent work has begun to address the challenge of integrating structured, tabular data into RAG systems, yet the scientific and engineering problems central to this thesis remain unsolved. For example, Tabular Embedding Model (TEM), a fine-tuned embedding model designed to better encode tabular information for retrieval tasks [3], has been proposed. Although their approach demonstrates that general-purpose embeddings struggle with structured data, the scope of their contribution is limited to model-level improvements. TEM assumes a fixed representation of tables, performs retrieval only at the file level, and does not investigate how tables should be chunked, indexed, or integrated with textual retrieval in a full RAG pipeline.

Beyond embedding strategies, recent architectural frameworks have attempted to solve tabular reasoning through symbolic execution and routing. For example, TableRAG proposes a hybrid approach in which the model generates SQL queries to interact with tabular data, significantly improving reasoning capabilities over heterogeneous documents [4]. Similarly, ERRATA introduces an "Extreme RAG"

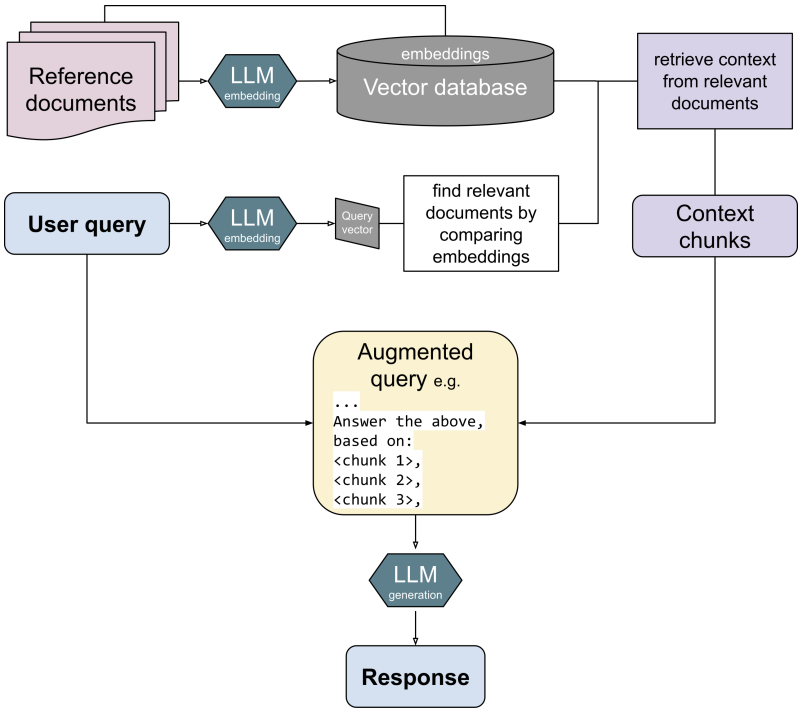

Figure 1.1: A common RAG-system architecture, figure taken from Wikipedia([2])

architecture that uses explicit routing to direct user queries to appropriate data tables based on high-level metadata [5]. However, both approaches rely heavily on the assumption that the underlying data is already structured or easily mappable to a clean database schema. In the public sector context, where data is often locked in "messy" unstructured formats like PDF-embedded tables without defined schemas, these SQL-reliant and explicit routing methods are not applicable. They do not address the fundamental engineering challenge of semantic vector retrieval for chunked schema-less table segments.

1.1.2 The Need for Pre-Retrieval Validation

RAG systems present a new attack surface for adversaries in the form of poisoning the evidence base with the goal of influencing the generator to produce malicious answers. An attack can, for instance, be to poison documents with previously correct content with hate speech or profanity. If the document is retrieved, the generator can answer the question with an answer that involves the poisoned content. Hate speech is a common threat on the internet, and there are multiple mitigations against it[6]. Another threat is prompt injections. The ultimate goal of the attack is to upload a poisoned document that involves content that overrides the generator's prompt instructions. As an effect, the generator may generate a malicious response to the user that could cause data leakage or a malicious generated response[7].

A similar type of threat model has been explored in the PoisonRAG project[8], which is referenced as a data poisoning attack in the OWASP Top 10 for Agentic Applications[1]. The project evaluates their attack against previously discovered defence mechanisms, such as paraphrasing and the perplexity of search queries.

However, they do not assess how well anomaly detection can capture the existence of poisoned vectors.

A second body of related work concerns defences against poisoning attacks in RAG systems. Many mitigations suggest sanitising the content before and after retrieval. An approach in this regard suggests analysing the embeddings before ingestion into the vector database. After retrieval, guardrails should be implemented in the LLM to ensure that the context is handled in a safe manner[9]. In addition to the guardrailing, it proposes multi-stage verification with consistency checking and also the use of an LLM, pre-trained on malicious content, to verify that all of the retrieved context is safe. This approach has been proven to lead to the detection of malicious content to a greater extent[9]. However, post-retrieval verification adds additional latency to retrieve the generated answer to the end-user since it has to be done online. Although the computation overhead reported for this method is small in comparison with the generation of the final answer, it still has an effect[9].

An additional approach to verifying the content post-retrieval is RAGDefender[10]. It uses machine learning algorithms and clustering mechanisms to try to detect malicious retrieved passages. Their core idea is that poisoned passages will form dense clusters in the embedding-space, which was empirically found[10]. Although effective, this approach also relies on online anomaly detection, which causes latency overhead while generating the answer to the end-user.

To mitigate the threat of RAG poisoning, this thesis proposes a *pre-retrieval validation mechanism*: using anomaly detection in combination with pre-trained classifiers in embedding space to verify whether uploaded vectors are considered malicious or not. Such validation is particularly important in public-sector deployments like SVEA, where data is ingested from multiple sources, and one unauthorised user that has managed to get access to upload documents may poison the whole vector database. By investigating how embedding-space validation can be used to enforce anomaly detection, this thesis addresses a relevant problem in RAG security and retrieval research.

1.2 Research Questions

The aim of this thesis is to discover how an RAG system for Swedish public-sector documents can reliably retrieve complex tabular data while simultaneously defending against vector database poisoning. Furthermore, this can be reduced to two core research questions:

1.2.1 RQ1: What strategies constitute a principled approach for representing and segmenting tables for vector retrieval?

This question investigates how different methods of segmenting and enriching tabular data, from Swedish public sector, affect the ability of a vector retrieval system to locate relevant table chunks in response to natural language queries. The methods

will be compared with respect to not only the resulting retrieval accuracy but also the latency each method of table representation adds to the system. This is of great importance, as an efficient system is a requirement.

1.2.2 RQ2: To what extent can automated detection mechanisms ensure data integrity of a vector database and how does it affect the system's performance?

This question investigates whether poisoned documents injected into a vector database can be reliably detected. This component will analyse the embedding distribution to identify anomalous documents that may have been produced by a poisoning attack. The system will experiment with unsupervised machine learning algorithms and classifiers to assess how effectively they can serve as a safeguard against data contamination. However, the system requires this additional operation with validation to be efficient. If the validation is not optimised, it results in reduced performance in terms of latency in the overall system. Thus, it is important to find an adequate balance between security and latency performance.

2

Theory

The aim of this chapter is to present the theoretical foundations necessary to understand the RAG system built in this thesis. It establishes the underlying principles of both the information retrieval pipeline and the unsupervised machine learning techniques used for anomaly detection and classification.

2.1 Embedding

An embedding is a dense, low-dimensional vector representation of data that aims to capture semantic meaning[11]. In this thesis, the data will be related to text, words, and tables. The representation is a dense vector of real numbers and is presented in a high-dimensional space. In contrast to traditional keyword-matching and encoding-methods, such as One-Hot Encoding or Bag-of-Words (BoW), which results in huge sparse vectors, the embeddings are more compact where every dimension includes information about the data[12]. In addition, embeddings offer semantic similarity and can automatically detect relationships between words and synonyms. This is possible due to the usage of trained embedding models that have learnt the context of words and how they have been used in the training data[11].

2.2 Retrieval

The retrieval stage is a fundamental part of an RAG pipeline. Its purpose is to, given the vectors of every chunk that is stored in the vector database and a query, find the most relevant vectors related to the query. The measurement of relevancy of the vectors is done by comparing the similarity of a vector with the embedding of the query, using the same embedding model. Because the query and the documents are embedded independently before this comparison, this approach serves as a bi-encoder architecture[13]. The similarity score can be achieved in different ways, depending on the search mechanism and distance metric used. One common distance metric that can be used for this purpose is cosine similarity[14]. Given the embedded query, which is represented in the same dimension as the vectors in the vector database, it inserts the query vector into the vector space and finds the vectors that result in the smallest angle in degrees from the query vector[14]. An illustration of this operation is illustrated in Figure 2.1. The number of vectors and a similarity threshold can be configured to control the number of vectors retrieved.

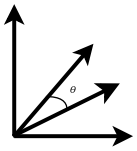

Figure 2.1: Geometric representation of vector embeddings, where θ denotes the angle between two vectors used to calculate cosine similarity.

2.2.1 Reranker

The retrieval step is necessary for efficient filtering and retrieval of possible candidate vectors. However, independent embedding of the query and chunks before the calculation of the similarity metric might struggle to find more nuanced contextual relationships[13]. As a result, the retrieval stage may be accompanied by a reranker that, for instance, uses a cross-encoder architecture to re-evaluate the relevancy of the retrieved vectors further. In contrast to the bi-encoder independent evaluation, the cross-encoder evaluates them by combining the query with every vector. It then forces the neural network to assess every vector with the query in more depth, and thus results in a more in-depth investigation of their relevancy. The cross-encoder architecture has been shown to improve the correct ranking of relevant documents compared to the bi-encoders[15]. However, this process is much more demanding compared to the bi-encoder and thus is often only used on the subset of vectors that the bi-encoder outputs[13].

2.2.2 Information Retrieval Metrics

Two well-known and industry-standard metrics for evaluating retrieval are Recall@k and Mean Reciprocal Rank (MRR)[16].

Recall@k is a metric that indicates at what rate relevant chunks appear at positions from one to k. It is calculated with the following formula[16]:

$$\text{Recall@}k = \frac{|R \cap \hat{R}_k|}{|R|} \tag{2.1}$$

where R and $\hat{R}_k$ represent the relevant documents and the top-k retrieved documents, respectively.

If the evaluation dataset consists of queries that map to exactly one document ($|R| = 1$), the metric is simplified: 1 if the target chunk is present in the top k results, and 0 otherwise. The overall system retrieval performance is the Mean Recall@k across all N queries:

$$\text{Mean Recall@}k = \frac{1}{N}\sum_{i=1}^{N}\text{Recall@}k_i \tag{2.2}$$

In addition to the Recall@k metric, the performance of a retrieval system can be evaluated regarding the MRR. It penalizes the system if the target chunk is not among the most relevant. The calculation is done with the following formula[16]:

For a single query, the Reciprocal Rank (RR) is calculated as the multiplicative inverse of the rank of the first correct answer:

$$\text{RR} = \frac{1}{\text{rank}_i} \tag{2.3}$$

where rank_i is the position of the target chunk in the ranked list of retrieved results. If the target chunk is not retrieved at all, the Reciprocal Rank is evaluated as 0.

To determine the overall ranking quality of the retrieval pipeline, the MRR is calculated taking the average of the reciprocal ranks throughout the entire set of N evaluated queries:

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\text{RR} \tag{2.4}$$

A MRR and Mean Recall@k score closer to 1.0 indicates that the system consistently places the most relevant chunk at the very top of the retrieved results.

2.2.3 Contextual Retrieval

Contextual retrieval is a topic within the field of RAG that tries to solve the issues with loss of context when chunking documents[17]. It proposes an approach that with the interaction of an LLM to replace the original chunk with text that not only is the chunk itself but also appended information about the context from the whole document. This is done by handing both the chunk and the document it originates from and asking for a "succinct context" that can be appended to the original chunk. This has been proven to achieve greater retrieval performance compared to passing the chunk itself to the embedding model[17]. However, this improvement comes with a cost in terms of additional overhead and latency, since this requires a call to an LLM for every chunk that is embedded. However, this computational overhead is not done online and thus does not affect the user experience

2.2.4 Step-back Prompting

Step-back prompting is a prompting technique that improves the reasoning capabilities of a large language model by using an abstraction step before the final response generation[18]. Prior to the response generation, the model is prompted to conceptualise the original question or generalise the underlying principle, this is

the step-back abstraction. Using that abstraction as a contextual focal point for the reasoning, the model generates the final response[18]. This two-step process of abstraction followed by abstraction-based reasoning is motivated by the observation that directly reasoning over questions containing many specific details increases the risk of errors in intermediate reasoning steps. By stepping back to a broader level of abstraction, the model retrieves more relevant background knowledge and is less likely to deviate from the correct reasoning path. This approach demonstrates a substantial improvement in reasoning capabilities in domains such as STEM and knowledge-based question-answering[18].

2.3 Validation

In this section, the theory behind the methodologies chosen to perform the validation of the vector database will be introduced.

2.3.1 Classifiers

Text classification is the task of assigning predefined labels to an input by extracting features. The classifier assigns a probability to each label for each sample in the input. The probability assigned to the labels is used to decide which label a particular sample should be assigned[11]. Text classification covers a wide selection of problems, such as language identification or sentiment analysis.

A common approach to classification is the use of supervised machine learning; during training, the input consists of the observation and an observation signal (the label)[11]. One such approach is to fine-tune a pre-trained language model on a specific classification task[19]. The pre-trained language model is augmented with a classification head and by using supervision on a smaller, task-specific dataset the model is fine-tuned on the nuances of the target classification task[19].

2.3.2 Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)

The unsupervised machine learning algorithm HDBSCAN is designed to overcome the primary limitation of its predecessor, Density-Based Spatial Clustering of Applications with Noise (DBSCAN)[20]. Traditional density-based algorithms are based on a global density threshold to identify clusters[20]. However, this approach may struggle with complex and more realistic data where clusters have significantly varying densities. If the global threshold is set too high, the sparse clusters are incorrectly labelled as noise. In the case of the threshold being too low, dense clusters may merge into a single massive group[20]. To resolve this, HDBSCAN abandons the concept of a single threshold. Instead, it generates a comprehensive clustering hierarchy that accounts for all possible density levels.[20]. Hierarchial algorithms, such as HDBSCAN, are effective in clustering and work well with clusters that have various shapes[21].

2.3.3 Global-Local Outlier Scores from Hierarchies (GLOSH)

GLOSH can be used in combination with clustering algorithms based on hierarchies creation. It is an efficient algorithm that can detect both global and local outliers and can be accompanied by HDBSCAN to detect anomalies that differ from its local region, but according to HDBSCAN's initial detection, it still belongs to a cluster[22][23]. GLOSH can be used on all types of distribution, regardless of the cluster's form since it is based on a non-parametric model[22].

2.3.4 Isolation Forest

This is an unsupervised anomaly detection algorithm that specifies anomalies by isolation, compared to most other approaches that construct a profile of what constitutes a normal instance and label anomalies as those that do not conform to the profile[24]. The Isolation Forest uses isolation instances depending on properties. The algorithm works by creating a tree and branches on the instances' attributes. Taking the assumption that the anomalous instances are fewer and differ from normal instances, the algorithm labels anomalies as the instances with the shorter paths to the root. Whereas the normal instances are deeper in the isolation tree[24].

2.3.5 Dimensionality Reduction with Uniform Manifold Approximation and Projection (UMAP)

UMAP is used to reduce the dimensionality of vectors and embeddings while still preserving its local connectivity[25]. UMAP has been proven to improve outlier detection performance by first reducing vectors to lower dimensions and then combining it with another outlier detection algorithm[26]. In addition, it is also one of the most scalable options in terms of performance[25].

UMAP captures local relationships by constructing k-nearest neighbour (kNN) graph from the data[25]. It uses this graph structure to learn a lower-dimensional representation of the high-dimensional data and does this by penalizing the learning if two points in close proximity in the high-dimensional data end up being too far apart in the projection. The mechanism for penalization is cross-entropy loss that tries to align the high-dimensional relationships with the projection. Furthermore, the kNN construction enables UMAP to optimize the projection, not only in regards to local neighbourhoods, but to the global topology of the high-dimensional data[25].

2.3.6 Classification Metrics

Classification metrics are used to evaluate the results of classifiers and these involve precision, recall, area under the precision-recall curve (PR-AUC) and the Fβ-measure[27][28][29]. These metrics are derived from the confusion matrix, which categorises predictions into four outcomes: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), where a positive instance corresponds to a poisoned node.

	Predicted Positive	**Predicted Negative**	**Total**
Actual Positive (P)	TP	FN	$P = TP + FN$
Actual Negative (N)	FP	TN	$N = FP + TN$

Table 2.1: Confusion matrix for binary classification, where positive instances correspond to poisoned nodes, P denotes the total number of true positives and N the total number of true negatives.

The precision metric displays the positive predictive value of the classifier and measures the proportion of flagged instances that are true positives:

$$precision = \frac{TP}{TP + FP} \tag{2.5}$$

Compared to recall as an IR-metric (see Section 2.2.2), classification recall measures the sensitivity, or the true positive rate:

$$recall = \frac{TP}{P} \tag{2.6}$$

The Fβ-score is a metric that provides a single metric that weighs the relative importance of recall and precision. The weight β is the relative importance of the recall over precision[27]:

$$F\beta = \frac{(\beta^2 + 1.0) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{2.7}$$

if recall and precision are of equal importance $\beta = 1.0$, which gives the F1-score. If recall is twice as important, then $\beta = 2.0$ which gives the F2-score.

The Area Under the Precision-Recall Curve (PR-AUC) is a metric that summarises the trade-off between precision and recall across different decision thresholds. This metric gives a more comprehensive analysis of an algorithm's performance when evaluation is done on heavily imbalanced datasets, where one label represents the majority of the data points[29].

To compute PR-AUC from discrete classification thresholds, the trapezoidal rule can be applied[29]. The formula for calculating this is:

$$PR\text{-}AUC = \sum_{i=1}^{n-1} \frac{precision_i + precision_{i+1}}{2} \cdot (recall_{i+1} - recall_i) \tag{2.8}$$

where $precision_i$ and $recall_i$ correspond to the values achieved at the i-th threshold operating point.

The FPR metric measures the proportion of actual negative instances that are incorrectly flagged as positive by the classifier[30]:

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N} \tag{2.9}$$

3

Technology Stack

This chapter presents the technologies used in the development of the system. The technologies are grouped into three categories: databases, infrastructure & services, and libraries & frameworks. Each section describes the technology and motivates why it was chosen for this project.

3.1 PostgreSQL

PostgreSQL is an object-relational database management system (ORDBMS)[31]. A relational database stores data in a structured format using rows and columns, and the PostgreSQL ORDBMS extends the SQL-language to manage the database using queries[31]. For this project, PostgreSQL was chosen because it being open-source and has mature documentation, making it easy to adopt.

3.2 Qdrant

Qdrant is a vector database and similarity search engine that makes it possible to manage vectors of high dimensions[32]. In addition to vectors, it is possible to store the corresponding metadata as a payload in the database. The payload complements the vector to provide more nuanced search capabilities in the database and a way to present the entry to the user[32]. A vector database is designed to efficiently store and query high-dimensional data points in what is called a Collection[32]. To optimise the vector database for efficient storing and querying, the database uses indexing techniques, such as Hierarchical Navigable Small World (HNSW). HNSW is a method that implements Approximate Nearest Neighbour search, making it possible to use similarity metrics such as cosine similarity to query vectors, depending on the similarity score, in a given collection[32]. Qdrant was chosen for this project due to its ability to deploy via Docker.

3.3 Docker

Docker is a platform that provides the deployment of services in reproducible and isolated environments using containerisation. A container is a running instance of an image. An image is a unit of software that packages an application together with its

dependencies. This ensures that the application runs consistently regardless of the host system[33]. Compared to Virtual Machines (VMs) which are an abstraction of physical hardware, effectively turning one server into multiple, which contain their copy of an operating system and the application, a container is an abstraction of the application layer[33]. Containers run as isolated processes in user space but share the same operating system kernel among all running containers. This makes it possible for much smaller images and less resource requirements[33]. In this project, Docker is used to deploy Qdrant, vLLM and TEI, and was chosen because of the facts that containers give a reproducible and consistent environment for development.

3.4 vLLM

vLLM is an inference and serving engine for large language models. It is designed to maximise throughput and resource use during generation workloads. It uses continuous batching and a memory-efficient key–value cache management system (PagedAttention), allowing multiple requests to be processed concurrently with minimal overhead [34]. This makes vLLM particularly well-suited for high-demand scenarios where many queries must be handled in parallel. In this project, vLLM was chosen due to its support for dynamic batching and high throughput, enabling faster response times and more efficient use of available GPU resources[34].

3.5 LLM: Qwen3-4B

Qwen3-4B is part of the latest iteration of the Qwen family of models. It has a higher multilingual support of 119 languages and dialects, compared to its predecessor Qwen2.5 with 29[35]. The Qwen3 series offers both a thinking mode and a non-thinking mode, and by utilising a thinking budget the user can specify a certain amount of context the model has to reason about the task [35]. This makes it possible for the user to change the performance cost at inference time to maximise utility. For smaller models in the series, such as the 4 billion parameter variant, strong-to-weak distillation from larger teacher models is employed. This significantly outperforms reinforcement learning in both performance and training efficiency, making compact variants highly capable relative to their parameter count [35].

3.6 Ablated LLM: Qwen2.5-3B-Instruct-abliterated

The standard versions of LLMs are trained to understand the difference between an adverse and a benign query. The most common procedure is to reject the query if it is considered harmful. In order to overcome this behaviour, abliterated models of already established LLMs have been developed. Abliterated models are created by rejecting the refusal vector during activation and at every token position[36]. The Qwen2.5-3B-Instruct-abliterated model is one such type and thus can be used to generate adverse responses[37].

3.7 Text Embedding Inference (TEI)

Text Embedding Inference (TEI) is a toolkit designed for efficient deployment and serving of open-source text embedding models. It provides a standardised service for generating vector representations of text[38]. TEI is optimised for production environments through features such as token-based dynamic batching, which groups requests to improve throughput. It also leverages Flash Attention to optimise inference[38]. Additionally, its lightweight design with small Docker images and fast startup times makes it well-suited for scalable and reproducible deployments. In this project, TEI is used to serve the embedding model and generate vector representations of text efficiently, which benefits from its batching capabilities and overall high-throughput performance[38].

3.8 Embedding model: multilingual-e5-large-instruct

Multilingual-e5-large-instruct is an embedding model that supports multiple different languages, including Swedish. It has been trained through weakly supervised learning on roughly one billion text pairs and ultimately fine-tuned[39]. It has been shown to be one of the best performing embedding models for multilingual semantic search tasks currently available[40]. However, a limitation of this model is that it can only handle a maximum input size of 512 tokens[39].

3.9 Reranker model: bge-reranker-v2-m3

Bge-reranker-v2-m3 is a reranker model that is efficient and built on 258 million parameters. It offers fast inference and is multilingual[41][42]. The model has been finetuned from the BGE-M3 embedding[43]. The core benefits of the BGE-M3 embedding is that they support over 100 languages, where Swedish is one of them. It has been trained on 1.2 billion text pairs of raw data for training and then with labelled data and synthetic data for fine-tuning[43].

3.10 Prompt Injection Classifier: deberta-v3-base-prompt-injection-v2

Deberta-v3-base-prompt-injection is a classifier, see Section 2.3.1, based on Microsoft's microsoft/deberta-v3-base model and has been fine-tuned for detecting prompt injections in English text[44]. It is trained on a variety of different datasets that contain multiple types of prompt injections and has shown great results in detecting prompt injections. However, it is only suitable for the English language[44].

3.11 Hate-Speech Classifier: unitary/toxic-bert

The TOXIC-BERT classification, see Section 2.3.1, model has been developed by Unitary with the purpose of detecting different genres of hate speech. This includes swear words and profanity[45]. The model has been trained on datasets that include toxic comments from social media of various types of toxicity. It offers support for some languages other than English but not Swedish[45].

3.12 LlamaIndex

LLAMAINDEX is an open-source framework for context-augmented LLM applications. Context-augmentation makes it possible to use custom data with LLMs. LLAMAINDEX does this by providing functionality for data-ingestion, indexing, and retrieval orchestration[46]. This means that it is possible to provide the LLM with specific data, which the model has not been trained on, during inference[46]. In this project, LLAMAINDEX was chosen due to its native integration with QDRANT as a vector store and its flexible abstractions for building retrieval pipelines.

3.13 Mistletoe

MISTLETOE is a Markdown parser that adheres to the COMMONMARK specifications[47]. Once parsed, an Abstract Syntax Tree (AST) is created. The AST can be accessed as a python object and uses custom tokens, which makes it easy to work with. In this project, MISTLETOE was chosen because it follows the COMMONMARK specifications and ease-of-use.

4

Methods

4.1 Problem Formulation and Methodology

The methodological approach of this thesis combines theoretical analysis with the design, implementation, and evaluation of a retrieval prototype. The work follows the two central research directions defined in the aims of this project: (1) developing different methods of representing tables for vector-based retrieval, and (2) designing and evaluating a validation mechanism that ensures the correctness and integrity of the evidence store, by reducing the risk of malicious data poisoning.

4.2 The Dataset

The dataset was provided by AI Sweden and consists of public documents gathered from publicly available records and converted to Markdown format. The documents originate from a broad range of sources, including different government agencies and municipalities. From this dataset, a subset of documents was sampled, of which all contained at least one table. This decision was made due to the focus of this thesis on table representations and how they affect retrieval performance (see Section 1.2.1). However, the ratio of text chunks to table chunks remains heavily skewed towards text, with tables accounting for only 7.35% of all chunks in the total corpus, see Table 4.1. This still constitutes a realistic representation of the distribution found in the full dataset.

Each Markdown file contains different elements such as headers, text paragraphs, and tables. After chunking, the total corpus consists of 23,638 chunks across 997 unique source documents, of which 21,901 are text chunks and 1,737 are table chunks, as summarised in Table 4.1. The mean chunk length is 208.55 tokens for the total corpus, with the table chunks on average longer (255.29 tokens) than the text chunks (204.84 tokens), as reported in Table 4.1. As illustrated in Figure A.2 and Figure A.3, the token and character length distributions are notably well-behaved regarding the 512 token limit imposed by the embedding model, only 169 chunks (0.7 %) across the entire corpus exceed 512 tokens, see Table 4.1. The corpus spans contributions from several government agencies and municipalities, of which Regeringen and Mediemyndigheten are the two largest sources. Regeringen comprises 16,828 chunks from 86 unique sources with a table density of 5.82 % and a mean chunk length of 239.94 tokens (Table 4.2). Mediemyndigheten contains 5,537 chunks

from 692 unique sources, has a higher table density of 8.63 % and a lower mean chunk length of 129.88 tokens (Table 4.3), reflecting the shorter and more uniform structure of its source documents, also visible in Figure A.2 and A.3. The remaining sources are considerably smaller in size and are therefore not evaluated in isolation. Experiments are conducted on either the full corpus or on one of the two largest sources individually.

Corpus Statistics	
Total chunks	23,638
Text	21,901
Table	1,737
Unique sources	997
Table density	7.35%

	Tokens	**Characters**
Mean	208.55	830.06
Median	206.00	811.00
Std	131.72	536.98
Min	6	8
Max	871	3,067
> 512 tokens	169 (0.7 %)	—
Avg (text)	204.84	840.98
Avg (table)	255.29	692.34

Table 4.1: Statistics and chunk length distribution for the total corpus.

Collection Statistics	
Total chunks	16,828
Text	15,849
Table	979
Unique sources	86
Table density	5.82%

	Tokens	**Characters**
Mean	239.94	976.52
Median	282.00	1,131.00
Std	123.33	505.36
Min	6	8
Max	871	3,067
> 512 tokens	112 (0.7 %)	—
Avg (text)	237.17	987.44
Avg (table)	284.68	799.69

Table 4.2: Statistics and chunk length distribution for Regeringen.

The relationship between chunk length measured in characters and tokens is expected to have a linear relationship. That is, the token count should grow linearly with the character count. This relationship is illustrated in Figure A.1. The distribution shows a strong positive linear correlation, with a Pearson correlation coefficient of 0.954, indicating that the number of characters is highly predictive of the corresponding token count, which verifies our assumption. As the character length of a chunk increases, the token length generally increases proportionally.

Most chunks are concentrated below approximately 1,500 characters and 400 tokens, although several outliers with larger token counts can be observed. The increasing spread for longer chunks suggests a higher variation in tokenization density for larger text segments, likely caused by differences in formatting, punctuation, tables, or

Collection Statistics	
Total chunks	5,537
Text	5,059
Table	478
Unique sources	692
Table density	8.63 %

	Tokens	Characters
Mean	129.88	468.04
Median	86.00	340.00
Std	117.52	426.64
Min	12	20
Max	535	1,732
$>$ 512 tokens	39 (0.7 %)	—
Avg (text)	119.07	453.18
Avg (table)	244.23	625.30

Table 4.3: Statistics and chunk length distribution for Mediemyndigheten.

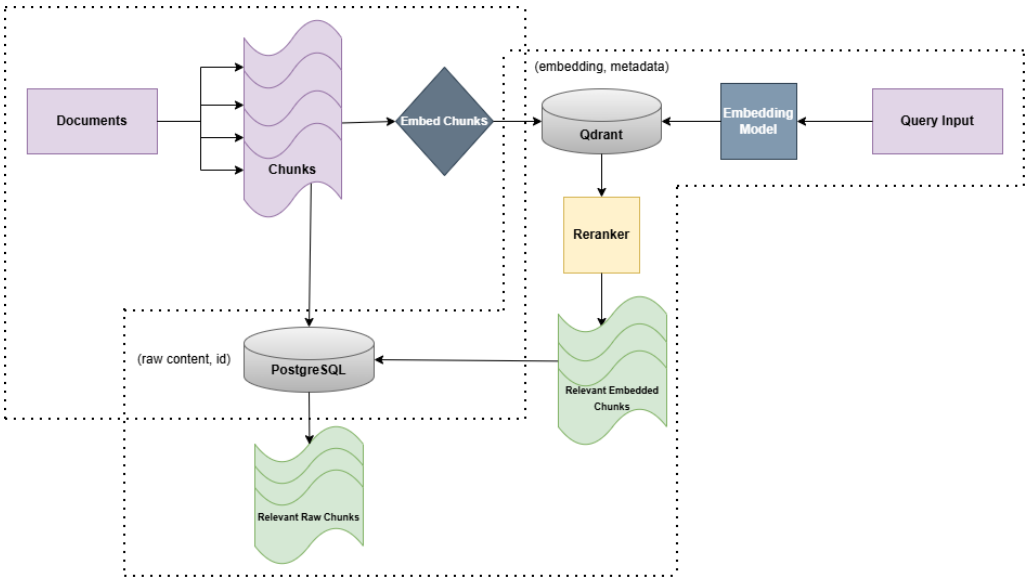

Figure 4.1: An overview of the end product of the RAG system

word composition. Despite these variations, the fitted linear trend demonstrates that character length provides a reliable approximation of token length across the corpus.

Furthermore, these example documents from public-sector sources will be examined to identify recurring patterns to aid in table reconstruction. These insights will guide the development of candidate segmentation strategies and representation formats.

4.3 RAG Design and Implementation

To realise the theoretical findings, a prototype retrieval pipeline will be developed, the work will be based on already established skeleton code provided by AI Sweden. The implementation will consist of three main components.

4.3.1 Ingestion: Extraction and Reconstruction of Tables

In order to handle the markdown files in a structured manner, we took advantage of the libraries that Python provides. The Mistletoe library was used to break down every individual file into an AST. The AST was then traversed to find every table that exists in the file.

When a table was encountered, it was preprocessed with custom methods that involve merging and cleaning the tables. The cleaning removes unnecessary white-spaces and empty columns. It is necessary to merge tables since when they are parsed from their original format (such as PDF) and the tables span multiple pages, they are split up as different tables in markdown with a newline separating them. Thus, if we find two tables separated with only a newline and it has the same number of columns as the previous table, it is merged with the previous table.

During preprocessing, metadata of the tables is extracted. The metadata contains information about the structure of the table, such as size, column names, and types. Every table is also wrapped with custom in-line tags that will later be used when the files are chunked to enforce rule-based chunking of the tables. After the files are preprocessed, the AST is rendered back to markdown format and stored in memory.

4.3.2 Segmentation: Embedding and Indexing

To ease the process of segmenting and chunking the markdown files before embedding, the Llamaindex MarkdownNodeParser was utilised. In order to enforce rule-based chunking of tables, a custom parser, called CustomNodeParser, was created. Whenever the CustomNodeParser encounters the custom table tags, it can chunk the table separate from other text paragraphs. Consequently, the table chunks are not diluted with other data, and thus keeping the table specific metadata available for all the subsequent chunks. The chunks, regardless whether it is table or text, are chunked with Llamaindex SentenceSplitter with a token limit of 512 tokens.

For every chunk, an id is created and is appended to its metadata. This node id is built to be stable across different runs of the system. As a result, the data will be consistent and can be re-used without having to parse the whole knowledge base and assigning new ids again. This id is also what ensures that the nodes are interconnected across the different components in the pipeline.

After parsing and extracting the nodes, they are inserted into a PostgreSQL database that serves as a middleman between the RAG generation and vector database. The PostgreSQL database stores the raw text and metadata for each chunk. This is of great importance, since some of the later proposed methodologies involve modification of the raw text. With the usage of PostgreSQL, the original text can be preserved to enable correct response generation to the end-user.

4.3.3 Retrieval

An overview of the developed RAG system and the retrieval process can be seen in Figure 4.1. The first step is taking a text input from the user query. The pipeline

then starts by using the embedding model, the same as for the document chunks, to embed the query into the same vector space. Then this query-embedding is passed to the Qdrant retriever that uses similarity search to match the query to the top k vectors. This value was chosen to be 30. The retrieved vectors are then passed to the reranker that further investigates the similarity and potentially changes similarity scores, see Theory 2.2. From this we take the IDs of the vectors and query the relational database, PostgreSQL, to get the plain-text representation of the vectors. These could later be passed on to the RAG-LLM that generates the final output to the user. However, to calculate the retrieval statistics needed for comparing our embedding approaches, we are not interested in the final output of the LLM. Hence, the pipeline ends before querying the RAG-LLM to generate the end-result to the user.

4.3.4 Deployment

All inference services in the pipeline are deployed as Docker containers to ensure reproducibility and dependency isolation. The language model is served via vLLM, which enables efficient handling of concurrent requests through batching. To avoid memory exhaustion, the context window is limited per request. The embedding model and the reranker are served via TEI, for which batching is also used to increase throughput. The evaluation loop is implemented asynchronously with a concurrency limit of five simultaneous retrievals, while reranking is serialised using a mutex to prevent concurrent GPU calls that would otherwise cause memory conflicts on the shared GPU. Since the LLM, embedding model, and reranker are hosted in separate containers, it is possible to be more flexible with resource allocation throughout the lifetime of the pipeline. Since summaries and query generation are only needed after ingestion, it is possible to start that container only in its stage of the pipeline. Similarly, the embedding model and reranker are only required during the indexing and evaluation phase, the containers are started and stopped sequentially to maximise available GPU memory in each phase and prevent them from competing for the same VRAM.

4.3.5 Table Representation Methodologies

To evaluate the impact of table formatting on retrieval performance, four different representation methodologies were applied to tabular data. The methods chosen are the following:

Unmodified Tabular Extraction (UTE) This method is the simplest representation format since it is solely that the table is parsed and embedded as-is with no further modification. All the metadata are excluded in the embedding and thus only the raw table text is embedded.

Metadata-Enriched Representation (MER) This method enriches the UTE representation by making the metadata, explained in Section 4.3.1, visible for the embedding model. As a result, both the metadata and the raw table text will be available in the retrieval process.

Natural Language Summarization (NLS) Instead of preserving the raw tabular structure, this approach completely replaces the raw table text with a summary generated by an LLM. The method of this approach was explained in more detail in Section 4.3.2.

Summary-Augmented Hybrid Representation (SAHR) Building upon the summarization approach, this final method uses the same summary but instead concatenates it with the raw table text.

All methods treat tables as structurally extracted chunks, meaning that Unmodified Tabular Extraction (UTE) serves as the internal baseline for comparison. A naïve text-only chunking approach was deemed not suitable as the baseline. This because it would not be directly comparable as table and text retrieval cannot be discerned and a separate query dataset would be required, thus it is not used as a baseline.

4.3.6 Generation of Summaries

In order to test the performance of the proposed methods utilising summaries, its raw content and metadata were passed to an LLM. The LLM is instructed with a prompt that can be seen in Appendix A.2.1. The LLM is given three attempts to create a summary. If a summary was successfully created, it was inserted and embedded in the Qdrant vector database and connected to the id that the original table had. If not, the raw content of the table was embedded instead. The chunks containing only texts are always embedded without any further modification of their content.

The chunks of tables and text are embedded with different settings depending on the different runs. The text has almost all metadata, except for organisation, id, and source-file included and used as parameters when embedded. However, when the tables have been modified and have either been entirely replaced by the summary or combines the raw content with the summary, all metadata is excluded since it should be included in the summary produced by the LLM.

4.4 Validation

A validation layer was implemented that inspects the ingested vectors to assess whether there are any malicious document chunks in the vector database. Different techniques were applied to identify content whose embedding does not match the distribution of its assigned organisation. The different approaches combine the unsupervised machine learning algorithms HDBSCAN, Isolation Forest and supervised classifiers. The objective is to detect and flag potential poisoning attempts before the data becomes available for retrieval.

4.4.1 Hyper-parameter Optimization

The unsupervised algorithms have the possibility to be tuned by changing the algorithms hyper-parameters to better suit the data. This was achieved by executing

a hyper-parameter search over 50 trials and finding the optimisation that resulted in the best performing parameters combinations. In the detection pipeline, this hyper-parameter optimisation is leveraged to find the best values and combinations to maximise the F2-score. The F2-score, in particular, makes it possible to tune the algorithms to limit the amount of false negatives as much as possible. This is because recall is prioritised over precision and thus the false positives increase and the false negative decrease. This priority is chosen since it may result in a potential increase in F1-score once the cross encoders get the results of the anomaly detection algorithms. However, in order not to tune the algorithms to flag all entries, the tuning is also balanced as the weighted sum of PR-AUC and the F2-score. The method can be treated as exploratory tuning, since it is based on the pre-knowledge of what embeddings are poisoned and not. In the context of this thesis the results of the IsolationForest and HDBSCAN can therefore not be treated as generalised estimates.

4.4.2 Anomaly Detection with Unsupervised Machine Learning Algorithms

After the tuning of hyper-parameters, the respective values for each argument are used to run the anomaly detection of HDBSCAN and Isolation Forest, respectively. The algorithms run independently on reduced dimensionality vectors that have been reduced by UMAP. The purpose of the algorithms is to detect whether there are any vectors that act as outliers. Since the algorithms originates from two distinctive and different families, they process and flag anomalies differently.

HDBSCAN assumes that normal organisational data will form dense clusters that semantically represent identical text. A vector will be flagged as noise if it does not meet a certain density threshold that has been defined by the tuning of the hyper-parameters. Along with this threshold, the generated clusters are also analysed with the help of GLOSH to find anomalies on the edges of valid clusters.

In contrast to HDBSCAN, Isolation Forest assumes that malicious vectors will be found in isolated regions. Its ensemble of randomised decision trees is used to assign an anomaly score to each individual chunk. The tuned hyper-parameter *contamination* is used by the algorithm to define a strict decision boundary, flagging only a specific percentage of the most isolated points.

4.4.3 Semantic Detection with Classifiers

Since HDBSCAN and Isolation Forest both deduce anomalies from the surrounding vectors in the vector space, it is impossible for them to find semantic outliers if the embedding is still similar to other vectors. Classifiers, on the other hand, evaluate the content and return a score on the level of certainty the model labels the entry. With the use of classifiers pre-trained on prompt injections and hate speech, it is possible to flag these even if their vectors closely resemble benign entries.

4.4.4 Validation Methodologies

In this thesis, different validation methodologies will be evaluated and compared on how successful they are at detecting prompt injections and hate-speech. They are the following:

Full Scan with Classifiers - Every entry in the vector database is passed to the classifiers, and given the entry's raw text, the classifiers output whether it believes if the entry is benign or malicious.

Anomaly Detection with Unsupervised Machine Learning Algorithms - In this mode, the pipeline strictly relies on unsupervised dimensionality reduction by UMAP and the machine learning algorithms (HDBSCAN and Isolation Forest) to flag anomalies. The detection is done independently by both HDBSCAN and Isolation Forest with the use of tuned hyperparameters.

Two-Stage Hybrid - In addition to the individual testing of the two different algorithms and cross-encoders, a hybrid approach will be evaluated. It is based on individual combining of machine learning algorithms with classifiers. The algorithms will be used to flag potential outliers and pass these on to the classifiers, which can then classify whether they believe that the flagged vectors are anomalies or not.

Two-Stage Hybrid with Sampling - Additionally, with the usage of HDBSCAN and its formation of clusters, an approach that not only passes the outliers, but also a small sample of vectors from every cluster will be passed forward to the classifiers. If a sampled vector is identified as an anomaly, the classifiers will evaluate the whole cluster. This approach is motivated by the idea that malicious vectors may form their own clusters and thus the whole cluster should be treated as possible malicious vectors.

4.5 Evaluation of Tabular Segmentation and Embedding

The system's retrieval performance is done by generating queries and assessing the amount of correct retrieved documents with retrieval performance metrics and the latency of the system.

4.5.1 Query Generation

In order to evaluate the system without the need of manually creating queries for every chunk that is parsed, the usage of an LLM is introduced. Given every chunk and its id, every chunk's content is used to let the LLM formulate one or multiple queries that are mapped to the id. The prompt with which the LLM is instructed is produced by AI Sweden and can be seen in Appendix A.2.2.

The questions that are being produced are then once again handed to an LLM, with the purpose of creating a more general query. This is based on the concept of Step-back prompting to try to achieve a more generic question, rather than key-word

specific. The queries generated by the step-back method are stored in a dataset and are used as user inputs to the system to retrieve documents that the system evaluates as related to the input.

When utilising an LLM to generate queries, we took extra steps to ensure that all chunks would get an assigned query by limiting the impact of failed generation attempts. This was done by retrying a maximum of 3 attempts. This resulted in a total of 31 chunks that did not receive an assigned query and were therefore filtered out of the test data.

4.5.2 Retrieval Assessment

To be able to assess the system's ability to retrieve appropriate chunks given a query, the metrics introduced in Section 2.2.2 are used. The metrics are split up to show results of the retrieval for table and text independently, as well as the overall performance regarding Recall@k and MRR. The values for k were chosen to be 1, 3 and 10 to give an overview of the recall across different positions in the retrieval. The results are produced by querying the system with every generated query in our dataset and evaluating at what position the target chunk ends up in, or if not at all. In addition to the retrieval performance, the different runs with different table representation methodologies are assessed by measuring the latency of the whole system.

4.6 Evaluation of Anomaly Detection

The evaluation of the performance of anomaly detection relies on investigating how the different methodologies perform by analysing different anomaly detection metrics as well as the system's latency.

4.6.1 Generating Synthetic Poisoned Data

In order to assess the performance, synthetic poisoned data must be generated and ingested into the vector database. The generation of synthetic prompt injection is made by sampling random entries from a dataset consisting of different prompt injections developed by NeuralAlchemy[48]. Meanwhile, the generation of hate-speech was chosen to be performed by an ablated LLM (see Section 3.6) with the prompt shown in Appendix A.2.4.

The chosen chunks passed to the LLM for hate-speech were randomly sampled from the targeted organisation. The transformed entries were then stored in a dataset to ensure consistency between different runs of anomaly detection.

4.6.2 Anomaly Detection Assessment

The purpose of this evaluation is to assess the capability of different validation methodologies to identify anomalous vectors and their performance in terms of latency. The experiments simulate targeted poisoning attempts with the synthetic data. The

results will be highlighted with the help of the classification metrics mentioned in Section 2.3.6. The detection performance for each proposed methodology will be evaluated on the two organisations Regeringen and Mediemyndigheten that was presented in Section 4.2.

4.7 Limitations

The design and evaluation framework presented in the previous sections aims to provide a rigorous comparison of the evaluation methods presented. However, there are limitations that can make the results, given in Chapter 5, less generalisable. The following limitations should be considered when interpreting the results.

4.7.1 Heuristic Table Parsing

Table extraction is the foundation of table representation methods. However, the parsing relies on heuristic pattern matching to determine if a page spanning table or two separate tables have been encountered during the parsing. This could lead to complications if two tables that do not belong together are accidentally merged. This includes, but not limited to, the header of the second table being treated as a data entry in the final table chunk and, subsequently, the actual data will not correspond to the new table's column headers.

4.7.2 Hardware Constraints

All experiments were conducted on a single NVIDIA Tesla T4 GPU with 16GB of VRAM, shared between the embedding model, the reranker and the language model. The Tesla T4 does not support bfloat16 precision, which is the default precision format for many modern large language models. As a consequence, models requiring bfloat16 are automatically cast to float32, doubling their memory footprint and making it infeasible to load more capable models within the available VRAM budget. This constrained the selection of language models to smaller variants compatible with float16 or float32 within 16GB, and required sequential scheduling of inference services to avoid memory conflicts between the embedding model, reranker, and language model running concurrently on the same device.

4.7.3 Model Selection

The retrieval pipeline relies on fixed models for embedding, reranking, and generation. The retrieval results are directly correlated with the choice of models, hence the results may be drastically different if other models are used, for the evaluation, on the same input.

4.7.4 Query Quality

The queries used to evaluate the retrieval are generated using a large language model and not real user queries. Therefore, queries may not reflect the diverse or organic

query scenarios experienced in a real system deployment, which could limit the external validity of the results.

4.7.5 Summary Quality

Table context summaries may not accurately capture the semantic meaning of the table and its metadata. Errors, hallucinations, or incomplete descriptions when generating the response would propagate into the index and degrade the retrieval results. Difficulties in distinguishing bad generation could limit the evaluation of the table representation methods using the summaries.

4.7.6 Anomaly Classifier Language Coverage

The classifiers used in this thesis were trained on English and Chinese. Since the dataset, see Section 4.2, consists of documents from the Swedish government and municipalities, the classifiers would perform worse due to the language mismatch. To mitigate this, we translate all the chunks from Swedish to English using Helsinki-NLP's translator model opus-mt-sv-en[49]. The translation might still produce text that does not fully encapsulate the original semantics and is thus worth to be considered when comparing the results.

4.7.7 Attack-Specific Classifiers

The proposed method of validation requires the use of pre-trained classifiers for each attack vector. In this thesis, we have chosen to focus on two such attacks, namely, prompt injection and hate speech. The need for specific classifiers necessitates the expansion of more classifiers if extended capabilities are needed.

5

Results

5.1 Retrieval

In this section, we will highlight the results of the different table representation methodologies. The methods are tested on the combined documents from the entire sampled corpus. The results show the recall at 1, 3 and 10, and the performance in terms of latency of the test run of each method.

5.1.1 Unmodified Tabular Extraction (UTE)

The structural table extraction achieved an *Recall*@10 of 0.7701 with an MRR of 0.6185 overall, see Table 5.1. The table versus text retrieval shows a notable difference between *Recall*@10 (0.6301 *vs.* 0.7812) and MRR of (0.4212 *vs.* 0.6342). The total retrieval and evaluation runtime was measured at 25978s, see Table 5.2.

Table 5.1: Results from structurally extracting tables as their chunks (UTE).

Queries	**Recall@1**	**Recall@3**	**Recall@10**	**MRR**
Overall (23607)	0.5311	0.6835	0.7701	0.6185
Table (1731)	0.3241	0.4633	0.6303	0.4212
Text (21876)	0.5474	0.7009	0.7812	0.6342

Table 5.2: Latency analysis of structurally extracting tables (UTE).

Checkpoint	**Elapsed (s)**	**Delta (s)**
`ingest_done`	92	92
`store_postgres_done`	94	2
`vllm_started`	196	102
`summaries_done`	196	0
`golden_dataset_done`	18591	18394
`vllm_stopped`	18592	2
`inference_services_started`	18611	19
`index_built`	19233	622
`evaluation_done`	45211	25978
`inference_services_stopped`	45212	2
`total_done`	45212	0

5.1.2 Metadata-Enriched Representation (MER)

Adding metadata enrichment to the table chunks resulted in no noticeable difference between overall *Recall*@10 (0.7702 *vs.* 0.7701) nor the MRR (0.6178 *vs.* 0.6185) compared to the internal baseline, while table retrieval performance shows an absolute change in *Recall*@10 of -0.0445 and a change in MRR of -0.0335, see Table 5.3. The retrieval and evaluation runtime was 25614s, see Table 5.4.

Table 5.3: Results from adding metadata to the table embedding (MER).

Queries	**Recall@1**	**Recall@3**	**Recall@10**	**MRR**
Overall (23607)	0.5305	0.6831	0.7702	0.6178
Table (1731)	0.2987	0.4281	0.5858	0.3877
Text (21876)	0.5489	0.7032	0.7847	0.6360

Table 5.4: Latency analysis from adding metadata to table embedding (MER).

Checkpoint	**Elapsed (s)**	**Delta (s)**
`ingest_done`	92	92
`store_postgres_done`	94	2
`inference_services_started`	104	11
`index_built`	736	632
`evaluation_done`	26349	25614
`inference_services_stopped`	26351	2
`total_done`	26351	0

5.1.3 Natural Language Summarization (NLS)

By replacing the raw table texts with LLM-generated summaries, we can note that the table retrieval performance when averaging the results across three different summaries for each table text is worse across all retrieval metrics compared to the internal baseline, see Table 5.5. However, marginal improvements can be seen in text retrieval performance. The runtime when averaging the retrieval and evaluation across three runs was 22442s, see Table 5.6.

Table 5.5: Results from replacing the raw table text with a summarization of the table content (NLS).

Queries	**Recall@1**	**Recall@3**	**Recall@10**	**MRR**
Overall (23607)	0.5263 (± 0.0005)	0.6771 (−0.0006/+0.0005)	0.7590 (±0.0003)	0.6120 (−0.0002/+0.0004)
Table (1731)	0.2584 (−0.0054/+0.0039)	0.3445 (−0.0042/+0.0044)	0.4231 (−0.0002/+0.0004)	0.3136 (−0.0020/+0.0022)
Text (21876)	0.5475 (±0.0002)	0.7034 (−0.0003/+0.0002)	0.7856 (±0.0003)	0.6357 (−0.0003/+0.0002)

5.1.4 Summary-Augmented Hybrid Representation (SAHR)

The last results pertain the method of using table summarisations as context. This method displays an absolute change in *Recall*@1 with (+0.0383), *Recall*@3(+0.0322)

Table 5.6: Latency analysis from replacing the table text content with a table summary (NLS).

Checkpoint	**Elapsed (s)**	**Delta (s)**
`ingest_done`	94	94
`store_postgres_done`	97	2
`vllm_started`	159	62
`summaries_done`	3759	3600
`vllm_stopped`	3761	1
`inference_services_started`	3771	11
`index_built` (run 1)	4383	612
`evaluation_done` (run 1)	26879	22496
`index_built` (run 2)	27487	607
`evaluation_done` (run 2)	49982	22496
`index_built` (run 3)	50593	610
`evaluation_done` (run 3)	72925	22333
`inference_services_stopped`	72927	2
`total_done`	72927	0

and *Recall*@10 (-0.0291) in table retrieval compared to the structural table extraction method (average of three runs) and with a change in MRR of $+0.0239$, see Table 5.7. However, when examining the maximum values for these metrics the improvement is even greater for *Recall*@$[1, 3]/MRR$ $(+0.0474, +0.0399)/ + 0.034$ and a smaller difference between *Recall*@10 $- 0.0197$. The runtime when averaging the retrieval and evaluation across three runs was $28872s$, see Table 5.8.

Table 5.7: Results from augmenting the raw table text with a summary as context (SAHR).

Queries	**Recall@1**	**Recall@3**	**Recall@10**	**MRR**
Overall (23607)	0.5329 (±0.0003)	0.6864 (−0.0004/+0.0005)	0.7713 (−0.0006/+0.0005)	0.6205 (±0.0002)
Table (1731)	0.3624 (−0.0071/+0.0091)	0.4955 (−0.0068/+0.0077)	0.6012 (−0.0137/+0.0094)	0.4451 (−0.0090/+0.0101)
Text (21876)	0.5464 (−0.0004/+0.0002)	0.7014 (−0.0010/+0.0007)	0.7847 (−0.0014/+0.0013)	0.6344 (−0.0007/+0.0005)

5.1.5 Summary of Retrieval Methodologies

Figure 5.2 summarises the retrieval performance across all four methods. The difference in overall/text recall and MRR is marginal when comparing between methods. The most significant variation observed is within table retrieval with summaries as context achieving the highest *Recall*@1 and *Recall*@3, while summary substitution leads to the worst table retrieval performance across all metrics. Figure 5.1 shows that runtime performance is dominated by retrieval and evaluation with summary substitution having the fastest evaluation runtime. Furthermore, the methods using LLM summaries, NLS and SAHR, both have a static offline performance overhead in regards of the summary generation.

Table 5.8: Latency analysis from augmenting raw table text with a summary as context (SAHR).

Checkpoint	Elapsed (s)	Delta (s)
`ingest_done`	92	92
`store_postgres_done`	94	2
`vllm_started`	146	52
`summaries_done`	146	0
`vllm_stopped`	148	1
`inference_services_started`	158	11
`index_built` (run 1)	786	627
`evaluation_done` (run 1)	28116	27330
`index_built` (run 2)	28755	639
`evaluation_done` (run 2)	58134	29379
`index_built` (run 3)	58781	647
`evaluation_done` (run 3)	88688	29907
`inference_services_stopped`	88690	2
`total_done`	88690	0

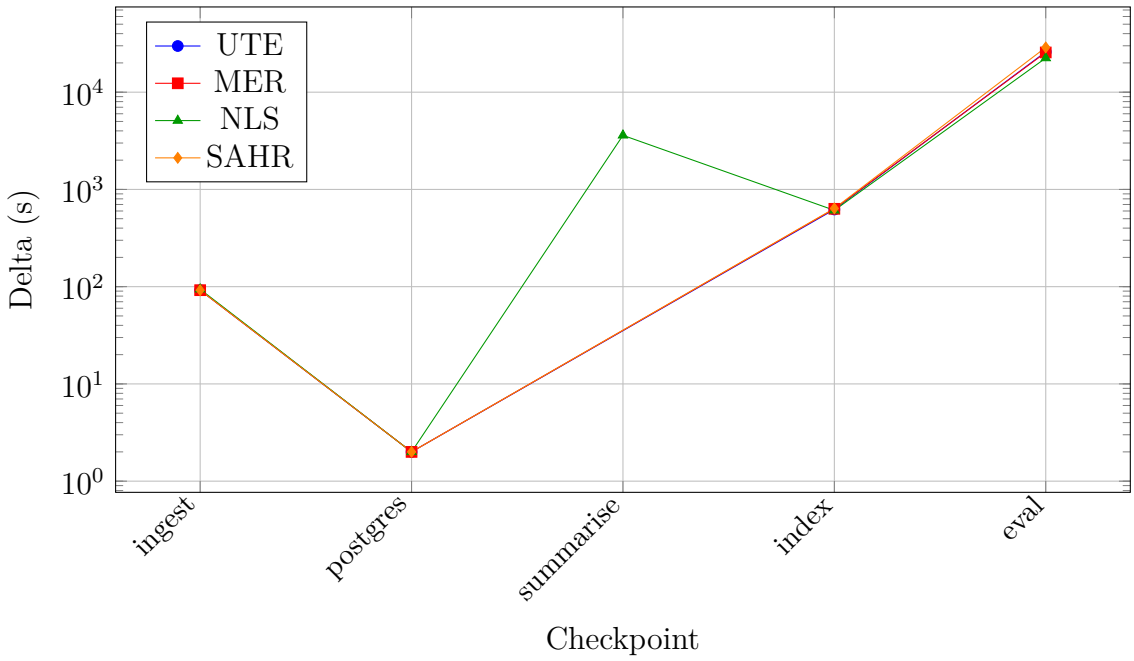

Figure 5.1: Pipeline stage durations (delta time) for each method experiment. Index and evaluation times are averaged over 3 runs where applicable. SAHR reuses the summaries generated by NLS and therefore incurs no summarisation cost at runtime; the summarisation step is omitted from its plot.

5.2 Validation

In this section, we highlight the results of the different validation methodologies designed to detect and filter malicious vectors before they can be retrieved from

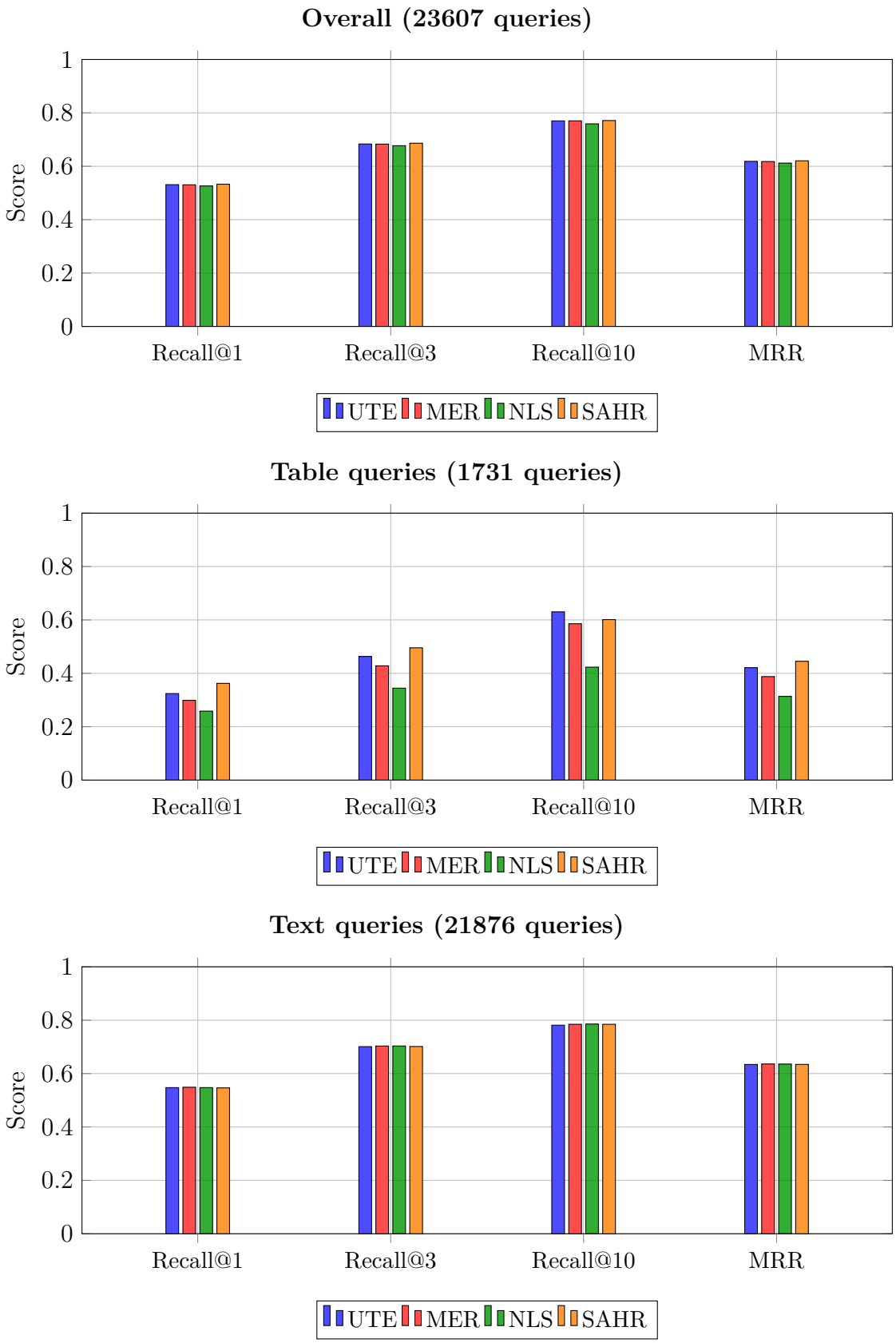

Figure 5.2: Retrieval performance grouped by metric, split by query type.

the vector database. We evaluate the performance across both Mediemyndigheten and Regeringen collections, based on how effectively the methods isolate outliers, their precision-recall trade-offs, and their impact on system latency. The amount of injected entries was 300 prompt injections and 288 hate speech for Regeringen and 92 prompt injections and 83 hate speech for Mediemyndigheten. The reason for the lower number of hate speech injections related to prompt injections is the failure to produce synthetic hate speech vectors. It is the effect of the LLM failing three attempts of generating the synthetic data. Also, in the upcoming results, the delta and elapsed time of where every detection method starts is neglected, since this is only based on the preparation needed for evaluation before the detection. The total of nodes in the results soon presented compared to the statistics of the dataset, represented in Table 4.2 and 4.3 for both Regeringen and Mediemyndigheten, shows a loss of a few nodes. Regeringen shows a loss of 14 nodes meanwhile Mediemyndigheten has 4 less nodes, this is due to duplicate tables in the same source file. Thus, the ID generated for both nodes is the same. Another important aspect is that the latency is not affected by the translation since this was done offline to more closely reflect the actual latency when using Swedish-trained classifiers.

5.2.1 Full Scan with Classifiers

The full scan utilising only the classifiers performed differently across the two collections and the results are stated in Table 5.9 and 5.10 for prompt injection and hate speech, respectively. For the Mediemyndigheten dataset, the method achieved *Recall* of 0.794 and $F1$ of 0.204 for prompt injections, while the detection of hate speech was *Recall* of 0.651 and $F1$ of 0.156. The precision for this dataset was 0.117 and 0.089, respectively and the false positive rate (FPR) was 0.101 and 0.102.

Regarding the Regeringen collection, the classifiers had significantly higher precision, resulting in $F1$ of 0.592 and *Recall* of 0.640 for prompt injections and $F1$ of 0.517 and *Recall* of 0.573 for hate speech. The increase in precision (0.550 and 0.471) was due to the decrease in the amount of false positives and more true positives, seen in FPR that is 0.010 for both injections.

Regarding system latency, the full scan approach has a computational overhead due to the necessity of evaluating every single entry in the vector database. For the smaller Mediemyndigheten collection, the detection time was $620s$ and $622s$ for the two injections types. Processing the significantly larger Regeringen collection required with the classifiers took $2073s$ for prompt injections and $2196s$ for hate speech.

Table 5.9: Prompt injection, classification and latency metrics for the Full Scan with Classifier.

Organization	**N Malicious**	**Precision**	**Recall**	**F1-Score**	**FPR**	**Latency (s)**
Mediemyndigheten	92	0.117	0.794	0.204	0.101	622
Regeringen	300	0.550	0.640	0.592	0.010	2073

Table 5.10: Hate speech, classification and latency metrics for the Full Scan with Classifier.

Organization	N Malicious	Precision	Recall	F1-Score	FPR	Latency (s)
Mediemyndigheten	83	0.089	0.651	0.156	0.102	620
Regeringen	288	0.471	0.573	0.517	0.010	2196

5.2.2 Anomaly Detection with Unsupervised Machine Learning Algorithms

The unsupervised detection utilizing HDBSCAN and Isolation Forest resulted in distinct metrics across the datasets which can be seen in Table 5.11 and 5.12 for prompt injection and hate speech, respectively . For the Mediemyndigheten collection, HDBSCAN achieved a *Recall* of 0.620 and $F1$ of 0.035 for prompt injections, while Isolation Forest recorded *Recall* of 0.087 and $F1$ of 0.028. For the detection of hate speech on the same dataset, HDBSCAN reached *Recall* of 0.241 and $F1$ of 0.019, compared to *Recall* of 0.036 and $F1$ of 0.016 for Isolation Forest. The precision in all runs on this dataset ranged between 0.010 and 0.018. The FPR for HDBSCAN was 0.576 (prompt injection) and 0.369 (hate speech), meanwhile Isolation Forest yielded FPR of 0.087 for prompt injection and 0.053 for hate speech, respectively.

On the Regeringen collection, HDBSCAN resulted in *Recall* of 0.390 and $F1$ of 0.036 for prompt injections, and *Recall* of 0.496 and $F1$ of 0.058 for hate speech. Isolation Forest achieved *Recall* of 0.227 with an $F1$ of 0.035 for prompt injections, and *Recall* of 0.101 with an $F1$ of 0.025 for hate speech. Precision values for both algorithms on this collection ranged from 0.014 to 0.031. The FPR for HDBSCAN was 0.368 and 0.246, compared to 0.214 and 0.111 for Isolation Forest.

Regarding system latency on the Mediemyndigheten collection, the duration for HDBSCAN was $48s$ for prompt injections and $48s$ for hate speech. Isolation Forest completed in $42s$ and $42s$, respectively. On the Regeringen collection, the anomaly detection time for prompt injections was $66s$ using HDBSCAN and $52s$ utilising Isolation Forest. Meanwhile, for the hate speech detection, HDBSCAN took $66s$ and Isolation Forest $49s$

Table 5.11: Prompt injection, classification and latency metrics for Anomaly Detection (AD) with Unsupervised Machine Learning.

Organization	Algorithm	N Malicious	Precision	Recall	F1-Score	FPR	Latency (s)
Mediemyndigheten	HDBSCAN	92	0.018	0.620	0.035	0.576	48
	Isolation Forest	92	0.017	0.087	0.028	0.087	42
Regeringen	HDBSCAN	300	0.019	0.390	0.036	0.368	66
	Isolation Forest	300	0.019	0.227	0.035	0.214	52

5.2.3 Two-Stage Hybrid: Anomaly Detection and Classifers

The two-stage hybrid approach without sampling yielded varying classification metrics depending on the anomaly detection algorithm utilised. The results are presented

Table 5.12: Hate speech, classification and latency metrics for Anomaly Detection (AD) with Unsupervised Machine Learning.

Organization	Algorithm	N Malicious	Precision	Recall	F1-Score	FPR	Latency (s)
Mediemyndigheten	HDBSCAN	83	0.010	0.241	0.019	0.369	48
	Isolation Forest	83	0.010	0.036	0.016	0.053	42
Regeringen	HDBSCAN	288	0.031	0.496	0.058	0.246	66
	Isolation Forest	288	0.014	0.101	0.025	0.111	49

in Table 5.13 and 5.14 for prompt injection and hate speech, respectively. For the Mediemyndigheten collection, HDBSCAN achieved a *Recall* of 0.294 and $F1$ of 0.082 for prompt injections, and a *Recall* of 0.013 and $F1$ of 0.009 for hate speech. The precision of HDBSCAN for each injection was 0.048 for prompt injection and 0.007 for hate speech. Isolation Forest resulted in *Recall* of 0.109 and $F1$ of 0.089 for prompt injections, and *Recall* of 0.026 and $F1$ of 0.036 for hate speech. Meanwhile, the precision for Isolation Forest was 0.075 on prompt injection and 0.057 for hate speech. The FPR for HDBSCAN were 0.096 and 0.027 for the respective attack types, while Isolation Forest recorded FPR of 0.022 and 0.006.

On the Regeringen collection, HDBSCAN achieved *Recall* of 0.283 and $F1$ of 0.339 with a precision of 0.421 for prompt injections. For hate speech, HDBSCAN reached *Recall* of 0.351, $F1$ of 0.478, and a precision of 0.748. In contrast, Isolation Forest yielded a precision of 0.635 for prompt injections with a *Recall* of 0.133 and $F1$ of 0.220. For hate speech, Isolation Forest recorded a precision of 0.654, *Recall* of 0.059, and $F1$ of 0.108. The FPR for HDBSCAN on this collection was 0.007 and 0.002, whereas Isolation Forest maintained an FPR of 0.001 across both attack types.

In terms of system latency for the two-stage process using HDBSCAN, the combination of anomaly detection and classification required a processing duration of $245s$ for prompt injections and $138s$ for hate speech on the Mediemyndigheten collection. For the Regeringen dataset, the hybrid detection process recorded a duration of $825s$ for prompt injections and $270s$ for hate speech. Meanwhile, Isolation Forest took $77s$ and $59s$ for the two attacks, respectively, on Mediemyndigheten and $440s$ and $427s$ on Regeringen.

Table 5.13: Prompt injection, metrics for Two-Stage Anomaly Detection (AD) with classifier but without sampling.

Organization	Algorithm	N Malicious	Precision	Recall	F1-Score	FPR	Latency (s)
Mediemyndigheten	HDBSCAN	92	0.048	0.294	0.082	0.096	245
	Isolation Forest	92	0.075	0.109	0.089	0.022	77
Regeringen	HDBSCAN	300	0.421	0.283	0.339	0.007	825
	Isolation Forest	300	0.635	0.133	0.220	0.001	440

5.2.4 Two-Stage Hybrid: Anomaly Detection with Classifiers and Sampling

The result of the two-stage hybrid approach incorporating the ML algorithms, classifiers and sampling is shown in Table 5.15 and 5.16 for prompt injection and

Table 5.14: Hate speech, metrics for Two-Stage Anomaly Detection (AD) with classifier but without sampling.

Organization	Algorithm	N Malicious	Precision	Recall	F1-Score	FPR	Latency (s)
Mediemyndigheten	HDBSCAN	83	0.007	0.013	0.009	0.027	138
	Isolation Forest	83	0.057	0.026	0.036	0.006	59
Regeringen	HDBSCAN	288	0.748	0.351	0.478	0.002	270
	Isolation Forest	288	0.654	0.059	0.108	0.001	427

hate speech, respectively. The experiment yielded a *Recall* of 0.261 and $F1$ of 0.192 for prompt injections on the Mediemyndigheten dataset, with 1 out of 4 identified clusters classified as malicious. For hate speech on the same dataset, the method recorded *Recall* of 0.442 and $F1$ of 0.075, classifying 2 out of 6 clusters as malicious. The precision for these tests was 0.152 and 0.041, with FPR of 0.024 and 0.142, respectively.

In the Regeringen collection, the sampling approach achieved a *Recall* of 0.263 and $F1$ of 0.320 with a precision of 0.407 for prompt injections, where 10 clusters were found and 0 were classified as entirely malicious. For hate speech, the method reached *Recall* of 0.336, $F1$ of 0.484, and a precision of 0.864, with 3 out of 36 clusters classified as malicious. The FPR for this collection was 0.007 for prompt injections and 0.001 for hate speech.

Regarding system latency, the sampled classification process recorded a duration of 309s for prompt injections and 276s for hate speech on the Mediemyndigheten collection. For the Regeringen dataset, the duration of the cluster sampling and classification phase was 770s for prompt injections and 522s for hate speech.

Table 5.15: Prompt injection, metrics for Two-Stage Anomaly Detection (AD) using cluster sampling with classifiers.

Organization	Algorithm	Clusters Found	Malicious Clusters	N Malicious	Precision	Recall	F1-Score	FPR	Latency (s)
Mediemyndigheten	HDBSCAN	4	1	92	0.152	0.261	0.192	0.024	309
Regeringen	HDBSCAN	10	0	300	0.407	0.263	0.320	0.007	770

Table 5.16: Hate speech, metrics for Two-Stage Anomaly Detection (AD) using cluster sampling with classifiers.

Organization	Algorithm	Clusters Found	Malicious Clusters	N Malicious	Precision	Recall	F1-Score	FPR	Latency (s)
Mediemyndigheten	HDBSCAN	6	2	83	0.041	0.442	0.075	0.142	276
Regeringen	HDBSCAN	36	3	288	0.864	0.336	0.484	0.001	522

5.2.5 Summary of Validation Methodologies

Tables 5.17 and 5.18 summarize the performance metrics in all methodologies tested and are divided by prompt injection and hate speech, respectively. The best performing values for each metric within a specific dataset are highlighted in bold text. For prompt injections in Mediemyndigheten, the highest precision (0.152) is achieved by using the two-stage hybrid with sampling. Meanwhile, the highest *Recall* and $F1$-score are 0.794 and 0.204, respectively, and they are achieved both with the full scan with classifiers. However, the lowest measured FPR (0.022) is when running with the two stage hybrid without sampling with Isolation Forest. The fastest detection is performed by using the Isolation Forest algorithm with a latency of $42s$ and a speedup of $15.0x$ in comparison with the full scan.

Regarding prompt injections in the Regeringen collection, the highest precision (0.635) is achieved by using the two-stage hybrid without sampling with Isolation Forest. Meanwhile, the highest *Recall* and $F1$-score are 0.640 and 0.592, respectively, and they are both achieved with the full scan with classifiers. However, the lowest measured FPR (0.001) is observed when running the two-stage hybrid without sampling with Isolation Forest. Similar to the Mediemyndigheten collection, the fastest detection is performed by using only the Isolation Forest algorithm ($52s$) and a speedup of $40.1x$ relative to the full scan which is the slowest.

For hate speech detection within the Mediemyndigheten dataset, the highest precision (0.089), *Recall* (0.651), and $F1-score$ (0.156) are all achieved by the full scan utilizing classifiers. The lowest FPR (0.006) is recorded by the two-stage hybrid without sampling using Isolation Forest. The latency is yielded by using Isolation Forest ($42s$) only, representing a $14.8x$ relative speedup over the full scan.

Regarding hate speech detection in the Regeringen collection, the highest precision (0.864) is achieved by using the two-stage hybrid with sampling. The highest *Recall* (0.573) and $F1$-score (0.517) are both produced by the full scan classifier configuration. The minimum FPR (0.001) is shared between the two-stage hybrid without sampling using Isolation Forest and the two-stage hybrid with sampling using HDBSCAN. The fastest execution time is achieved by using Isolation Forest ($49s$), providing a relative speedup of $45.0x$ compared to the full scan.

Table 5.17: Prompt injection, summary of classification and latency metrics. Bold text indicates the best metric achieved per organization

Organization	Methodology	Precision	Recall	F1-Score	FPR	Latency (s)	Speedup
Mediemyndigheten	Full Scan (Classifiers)	0.117	**0.794**	**0.204**	0.101	622	1.0x
	Only AD (HDBSCAN)	0.018	0.620	0.035	0.576	48	13.0x
	Only AD (Iso. Forest)	0.017	0.087	0.028	0.087	**42**	**15.0x**
	AD Classify (HDBSCAN)	0.048	0.294	0.082	0.096	245	2.5x
	AD Classify (Iso. Forest)	0.075	0.109	0.089	**0.022**	77	8.0x
	AD Sample Classify (HDBSCAN)	**0.152**	0.261	0.192	0.024	309	2.0x
Regeringen	Full Scan (Classifiers)	0.550	**0.640**	**0.592**	0.010	2073	1.0x
	Only AD (HDBSCAN)	0.019	0.390	0.036	0.368	66	31.6x
	Only AD (Iso. Forest)	0.019	0.227	0.035	0.214	**52**	**40.1x**
	AD Classify (HDBSCAN)	0.421	0.283	0.339	0.007	825	2.5x
	AD Classify (Iso. Forest)	**0.635**	0.133	0.220	**0.001**	440	4.7x
	AD Sample Classify (HDBSCAN)	0.407	0.263	0.320	0.007	770	2.7x

Table 5.18: Hate speech, summary of classification and latency metrics. Bold text indicates the best metric achieved per organization

Organization	Methodology	Precision	Recall	F1-Score	FPR	Latency (s)	Speedup
Mediemyndigheten	Full Scan (Classifiers)	**0.089**	**0.651**	**0.156**	0.102	620	1.0x
	Only AD (HDBSCAN)	0.010	0.241	0.019	0.369	48	12.9x
	Only AD (Iso. Forest)	0.010	0.036	0.016	0.053	**42**	**14.8x**
	AD & Classifiers (HDBSCAN)	0.007	0.013	0.009	0.027	138	4.5x
	AD & Classifiers (Iso. Forest)	0.057	0.026	0.036	**0.006**	59	10.5x
	AD & Sample & Classifiers (HDBSCAN)	0.041	0.442	0.075	0.142	276	2.2x
Regeringen	Full Scan (Classifiers)	0.471	**0.573**	**0.517**	0.010	2196	1.0x
	Only AD (HDBSCAN)	0.031	0.496	0.058	0.246	66	33.4x
	Only AD (Iso. Forest)	0.014	0.101	0.025	0.111	**49**	**45.0x**
	AD & Classifiers (HDBSCAN)	0.748	0.351	0.478	0.002	270	8.1x
	AD & Classifiers (Iso. Forest)	0.654	0.059	0.108	**0.001**	427	5.1x
	AD & Sample & Classifiers (HDBSCAN)	**0.864**	0.336	0.484	**0.001**	522	4.2x

6

Discussion

6.1 Retrieval

Retrieving tables in a RAG system presents a distinct challenge compared to text retrieval as the data is conveyed in a structured format compared to natural language. This section will discuss how the choice of representation method, presented in this thesis, affects the retrieval performance in relation to RQ1 (Section 1.2.1), utilizing the results from Section 5.1 and how the choice of method might have influenced the results.

6.1.1 Key findings

All methods achieve comparable overall *Recall*@10, ranging from 0.759 (NLS) to 0.771 (SAHR), suggesting that the choice of table representation has limited impact on overall retrieval performance, see Figure 5.2. The difference is more pronounced for table-specific retrieval, which is central to RQ1 (see Section 1.2.1). SAHR achieved the highest *Recall*@1 and *Recall*@3 for table chunks (0.362, 0.496), while UTE achieved the highest *Recall*@10 (0.630). NLS performed the worst across all table retrieval metrics, while MER consistently underperformed UTE on table retrieval despite a marginal overall improvement. Regarding runtime, Figure 5.1 shows that evaluation dominates across all methods. NLS achieved the lowest average evaluation time ($22442s$) compared to SAHR ($28872s$), which may reflect differences in chunk size between the methods. The summarization-based approaches additionally contract a one-time offline overhead of approximately $3600s$ for summary generation, the significance of which scales with the number of table chunks in the knowledge base.

6.1.2 Insights of the Method's Impact on the Result

The methodology involves certain steps that may have an effect on the results. One such particular part is query generation. The queries were created by an LLM, given a chunk's content. This creates bias towards the LLM and the risk of hallucination or a poorly formed query must be taken into consideration. Furthermore, it is difficult to capture what a poorly formed question actually is. In our thesis, it is a question that has pre-knowledge about the specific targeted chunk. In order to mitigate this to some extent, the concept of step-back prompting was introduced to try reconstructing the question into a broader concept in hopes that the questions

would be more realistic. This addition could also have affected the results since the more specific questions may have included more keywords that had an exact match with the raw-text tables compared to the summaries which may have used synonyms or even neglected some of the keywords. As a result, the use of step-back prompting (see Section 2.2.4) may achieve better results for the summaries in contrast to the non-modified tables. However, the fact that the queries become more generalisable and realistic justifies the introduction of step-back prompting.

In addition to query generation, the implementation of LLMs to produce summaries also creates bias towards the LLMs. The results are based on the assumption that the summaries are well-formed and capture the most important semantic content of the tables. In addition, LLMs can hallucinate and produce nonsensicality summaries. In order to prevent this to some extent, the summaries were validated, and if a failure was detected, the LLM had a maximum of three attempts to generate an approved summary.

Furthermore, it is not only the LLMs ability of producing correct responses that inflicts the result, but also the prompt that is instructing it. However, prompt engineering and the aim of achieving the best resulting prompt is considered out of scope of this thesis.

An additional choice that may have affected the result is the embedding model. It has been proven to be the best embedding model in terms of multilingual understanding and was the main reason for being chosen for this project. However, it comes with a specific limitation which is that it has a maximum input token size of 512 tokens. This sets a requirement on the system to chunk the documents in a maximum length of a maximum of 512 tokens, otherwise the chunks will be truncated by the model, and thus loss of content will occur. This is also the case for the tables, and this requires the tables to be split up in multiple chunks if they are too long. The chunking may happen in the middle of a table row, and therefore could the structure of the table be misaligned, making it harder to capture the relationship between every row and column. This was hoped to be mitigated, partly by including metadata that contains information on how the relationships are structured. However, as seen in the results, the metadata enrichment did not improve the retrieval and this may be an effect of the limited input token size of the model.

Furthermore, perhaps also the results of the augmented summaries were also affected by this choice. A model with a larger input token size could allow for both longer summaries and that they could be augmented with a larger chunk of the table. In combination, this could possibly result in improvements for the SAHR approach.

6.1.3 Interpretation of Results

The results presented in Section 5.1 show that the retrieval performance is correlated with the choice of table representation method, particularly the table-specific retrieval. As shown in Figure 5.2, the overall recall and MRR score similarly between all the tested methods and thus indicating that the choice of table representation has a limited effect on text retrieval, which constitutes a majority of the test dataset, see

Table 4.1. However, when interpreting the table retrieval metrics the two methods that were most effective, given research question 1.2.1, were UTE (internal baseline) and SAHR. The former achieved the highest *Recall*@10, suggesting that using the raw table text in Markdown format is beneficial when retrieving the correct chunk within the top 10 results is needed, while the latter scored the highest *Recall*@1, *Recall*@3 and MRR. This indicates that augmenting the tables with context, similar to contextual retrieval (Section 2.2.3) but by summarising the table content and metadata, improves the general ranking by the reranker (Section 3.9) when the correct table chunk is retrieved. This trade-off between *Recall*@10 and *Recall*@1/MRR between the two methods suggests that while the summary context helps the reranker score the correct chunk higher, it does not translate into improved *Recall*@10 for the combined retrieval-reranking pipeline. Since the results for *Recall*@10 reflect the output after reranking, this does not necessarily isolate the initial retrieval coverage of the embedding model alone (Section 3.8). A separate evaluation of pre-reranker recall would be needed to confirm this.

As presented in Figure 5.1 the most contributing factor to system performance across all methods is the retrieval (evaluation) runtime. The ingestion and indexing runtime contributes comparably between the methods but at a lower performance cost. However, the summary-based approaches incur a one-time performance overhead when summarising the table chunks. This overhead scales linearly with the amount of table chunks in the knowledge base and could therefore have a considerable effect on latency, due to the first offline generation step, before these chunks can be retrieved. Disregarding the performance cost of summary generation, the NLS method has the lowest performance impact on retrieval with $22442s$ on average compared to the internal baseline ($25978s$) and SAHR ($28872s$), suggesting that this representation method may increase efficiency when reranking. This could possibly be attributed to the summaries having more uniform chunk sizes.

The practical choice of table representation has a trade-off between retrieval accuracy and system performance. The Unmodified Tabular Extraction has a competitive retrieval accuracy at a low development complexity and average system performance compared to the alternative table representations. The Summary-Augmented Hybrid Representation method indirectly increases the ranking capabilities of correct table chunks at the cost of a one-time performance overhead and the lowest system performance during retrieval. MER offers neither better retrieval accuracy nor system performance compared to UTE (internal baseline), making it difficult to justify its use in the context of this thesis without further investigation of the 512 token size limit of the embedding model and the resulting truncation discussed in Section 6.1.2.

6.2 Validation

Ensuring data integrity in a vector database is a non-trivial problem and one not commonly tackled. This section will discuss the utility and effectiveness of the automated detection mechanisms presented in this thesis on how well they discern anomalous from benign content in the vector database. The discussion will be based

on results from Section 5.2 in relation to RQ2 (Section 1.2.2).

6.2.1 Key findings

The results presented in Section 5.2 show that using embedding-space distributions to validate the integrity of the database, may not score sufficiently high in metrics, such as $F1$ and FPR, to be a viable solution on their own. This would make the validation flag too many benign entries as malicious, and a high percentage of actual malicious entries would not be flagged by this mechanism. Using HDBSCAN and Isolation Forest as filters for suspicious entries, the performance increase is substantial compared to using pre-trained classifiers on the entire database. However, this results in a decrease in detection capabilities, see Tables 5.17 and 5.18. By using classifiers on all data points resulted the highest $F1$-score, but at the cost of latency. If the goal is to minimize the system performance overhead of the validation, the proposed hybrid classification methods might be preferable. In Table 5.18 the hybrid methods utilizing the HDBSCAN algorithm as filter has an 4.2 – 5x decrease in latency compared to the full scan, when running it on Regeringen, and with only a decrease of 0.033–0.039 in $F1$-score. The results in the same tables also state that the methods perform worse when used on Mediemyndigheten regardless of the choice of attack to be evaluated. This could indicate a relationship between the choice of data to validate and the accuracy of the detection. This indicates a better balance between performance and the $F1$-score. However, as can be seen in Table 5.17, the same methodologies result in only a decrease of 2.5–2.7x in latency and a decrease of 0.253–0.272 when evaluated on prompt injections in the Regeringen dataset. A reason for these varying results across the two collections, could for instance be an effect of the different sizes ($16,828$ for Regeringen versus $5,537$ for Mediemyndigheten), or possibly that the different documents may be written or structured differently.

6.2.2 Impact of the Chosen Method

The use of the two unsupervised ML algorithms HDBSCAN and Isolation Forest introduces potential flaws to the validation system, since both algorithms are sensitive to hyperparameters. It requires the implementer to carefully tune the parameters to achieve the best result possible for a certain problem. We have decided not to report the specified tuned values for our settings, since these are highly individual for the problem we want to solve. Moreover, the tuning is exploratory, since it is based on known injections, and thus the chosen parameters might not be the most appropriate ones for detecting other attacks than the one tested on. However, they are important for the validation of the results produced by the given algorithms to make a justifiable comparison.

The introduction of classifiers requires translation, since both classifiers are trained on English and Chinese text. Thus, they are not initially suitable for the corpus of this project. However, there are sophisticated translators available to translate text from Swedish to English. Although, they are time-consuming and not applicable in a real-world setting but for the purpose of this thesis, this can be done as a one-time cost before running the validation pipeline.

Besides the translation, the production of synthetic data for hate speech required the usage of an LLM. This method is useful for the purpose of not having to manually modify the texts. However, it introduces potential errors that may affect the result. In the result of this thesis, it affected the result by delimiting the synthetic data from the desired 92 malicious chunks to 83 for Mediemyndigheten and from 300 to 288 for Regeringen. However, this is not an error that inflicts the result massively, since the same dataset is used across all different methods within the same organization. Both translation and the usage of an LLM introduce bias to the evaluation when the results are based on their ability to producing realistic content for the targeted chunks.

The method of incorporating sampling with the two-stage hybrid pipeline may be a method that can produce a miscellaneous result. This is because it may result in the whole database being flagged as potential anomalies. Thus, it is important to use this in combination with another detection mechanism, such as classifiers, to avoid the risk that the whole vector database will be flagged as malicious.

In comparison to HDBSCAN and Isolation Forest, who analyse the embeddings of the vectors, the classifiers rely on the text of the chunks. If classifiers are used in combination with ML algorithms for anomaly detection, the storage of the texts of every chunk is needed. This requires additional memory space, which may not be applicable on all systems if it is a constraint of the system.

6.2.3 Interpretation of Results

The results presented in Section 5.2 gives a comprehensive overview of how the different methods regarding the validation perform. In general, the full scan with classifiers shows the best results in detection, especially when it comes to recall and $F1$-score. However, hybrid approaches generally outperform the full scan in terms of precision. They are also well capable of reducing the false positive rate compared to the other approaches. In addition, hybrid approaches reduce the latency of validation between 2.5 and 8 times compared to full scan. However, it can be assumed that the main reason that some hybrid approaches have a $F1$-score rather similar to the full scan is due to the classifier's judgment. This assumption can be made since the results, in which only HDBSCAN or Isolation Forest are used, were poor in relation to the $F1$-score. The reason for the insufficient results are mainly that they flag entries aggressively and thus have a high number of false positives in relation to true positives. Although the results may not be sufficient for HDBSCAN and Isolation Forest alone, they show a significant decrease in terms of latency. This is especially the case for Isolation Forest that has a speedup of $40 - 45$x relative to the full scan with classifiers on the larger dataset with chunks from Regeringen. Meanwhile, HDBSCAN achieves a speedup between $31.6 - 33.4$x faster than the full scan. These are results that normally should not be overlooked, but given HDBSCAN's and Isolation Forest's poor results regarding the classification, they cannot be established as proper solutions to the problem of validating the vector database on their own.

Furthermore, the results indicate that the validation methods generally perform

worse on the Mediemyndigheten dataset compared to the one for Regeringen. As detailed in Tables 5.17 and 5.18, the full scan with classifiers on Mediemyndigheten yielded a precision of only 0.117 for prompt injections and 0.089 for hate speech, coupled with high FPR exceeding 0.100. In contrast, the same method applied to the Regeringen dataset achieved significantly higher precision scores (0.550 and 0.471) and maintained a low FPR of 0.010.

This might be the cause of the structural differences between the two corpora given in Section 4.2. The Regeringen dataset has $16,828$ chunks that originate only from 86 unique sources. The mean size of every chunk is 239.94 tokens. This homogeneity could be the case that allows the embedding model to form a dense, well-defined vector space, enabling both the classifiers and the anomaly detection algorithms to effectively isolate out-of-distribution malicious injections. In contrast, the Mediemyndigheten dataset contains $5,537$ chunks from 692 unique sources. The resulting vector space is therefore much more sparse, making it difficult for clustering algorithms like HDBSCAN to form benign clusters.

Furthermore, the composition of the text likely lowers the classifiers' performance in the Mediemyndigheten dataset. The mean chunk length is notably shorter (129.88 tokens), which can affect the classifiers' ability to differentiate between a benign statement and a disguised prompt injection. This could be assumed since in Table 5.9 and 5.10 show a significantly higher FPR (0.101 and 0.102 for Mediemyndigheten, and 0.010 for Regeringen for both attacks).

6.3 Reproducibility

This project involves multiple methods that can affect the possibility of reproducing the results achieved. Firstly, the usage of LLMs to generate various types of data in the form of summaries, queries, and malicious synthetic data could be a restriction. Secondly, the dataset is gathered from various Swedish governments and municipalities, which is not easily collected. However, with the information given in this thesis regarding the LLMs that were used and each specific prompt for each step, in combination with the description of the dataset, it should be possible to reproduce with similar data and methodology.

6.4 Future Work

The project has showcased of how different approaches can be used for both improving accurate retrieval for tables and validating the vector database before retrieval. It has opened up new possible ways of how these problems can be solved, and the chosen methods could be discovered and tested even further. In Section 6.2.2 and 6.1.2 several possible impacts of the methods have been presented that could be investigated to try to improve the results. Among these are for instance testing the impact of another embedding model with a larger input token size to see its effect on the retrieval. Regarding the validation methodology, more injection attacks that are a threat for RAG systems, besides prompt injections and hate speech could be

tested. In addition, the introduction of classifiers for validation improved the results, although there are currently no available classifiers that can understand Swedish text. This is an important topic regarding security that should be considered since an attacker may utilise the classifier's inability to understand Swedish language when the rest of the RAG system does. For instance, may a Swedish prompt injection not be detected by the classifiers, but it can still achieve the intended malicious attack. Furthermore, other machine learning algorithms that are capable of anomaly detection may be tested.

7

Conclusion

The aim of the project was to discover how structured data, in the form of tables, could be represented with different methodologies to possibly increase the retrieval accuracy of them in a RAG system. In addition, a validation layer on the vector database was developed to assess how different unsupervised machine learning algorithms and classifiers could detect different poison attacks and flag the poisoned entries as anomalies. The effectiveness of each methodology in terms of latency was also assessed for both purposes. The evaluation was done on a dataset that originates from different sources from the Swedish public sector.

The table representation methods that best answered the RQ1 1.2.1, were the Unmodified Tabular Extraction (UTE) and Summary-Augmented Hybrid Representation (SAHR). The latter made reranking more accurate and the correct tables were therefore more prominent to be placed among the top 3 candidates for generation. However, the former method enabled the correct chunk to be found at higher rates. Thus, the choice of method depends on the use case and complexity of the system. With the UTE being the simpler method to implement, and at lower system performance cost, it is more suitable for SVEA's development.

Furthermore, to use unsupervised anomaly detection machine learning algorithms and classifiers, RQ2 1.2.2, is deemed possible. The classifiers showed the best capability of flagging the malicious content as anomalies. However, the use of classifiers requires one classifier per attack that should be prevented. The results indicated that classifiers are time-consuming relative to only using anomaly detection algorithms, thus the method of solely using classifiers is not the best choice regarding computational cost. In addition, the classifiers require translation, which is not a valid solution for a real-time setting. Using the classifiers in combination with the anomaly detection algorithms speeded up detection while maintaining competitive results. Consequently, a hybrid approach combining both classifiers and anomaly detection algorithms are suggested as the best approach. However, since anomaly detection methods were evaluated using exploratory tuned parameters, the results may not generalise to unseen data.

Bibliography

[1] J. Sotiropoulos, K. Katz, and R. F. D. Rosario, *Owasp top 10 for agentic applications 2026*, OWASP, Version 2026. OWASP GenAI Security Project [Online]. Available: https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/ (Accessed: 25 February 2026).

[2] Wikipedia, *Retrieval-augmented generation*, [Online]. Available: https://en.wikipedia.org/wiki/Retrieval-augmented_generation (Accessed 26 February 2026).

[3] S. Khanna and S. Subedi, *Tabular embedding model (tem): Finetuning embedding models for tabular rag applications*, 2024. arXiv: 2405.01585 [cs.AI].

[4] X. Yu, P. Jian, and C. Chen, *Tablerag: A retrieval augmented generation framework for heterogeneous document reasoning*, 2025. arXiv: 2506.10380 [cs.CL].

[5] S. Roychowdhury, M. Krema, A. Mahammad, B. Moore, A. Mukherjee, and P. Prakashchandra, *Eratta: Extreme rag for table to answers with large language models*, 2024. arXiv: 2405.03963 [cs.AI].

[6] B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee, "Hatexplain: A benchmark dataset for explainable hate speech detection," 17, vol. 35, May 2021, pp. 14 867–14 875. DOI: 10.1609/aaai.v35i17.17745. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17745.

[7] Y. Liu et al., *Prompt injection attack against llm-integrated applications*, 2025. arXiv: 2306.05499 [cs.CR].

[8] W. Zou, R. Geng, B. Wang, and J. Jia, *Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models*, 2024. arXiv: 2402.07867 [cs.CR].

[9] B. Ramakrishnan and A. Balaji, *Securing ai agents against prompt injection attacks*, 2025. arXiv: 2511.15759 [cs.CR].

[10] M. Kim, H. Lee, and H. Koo, *Rescuing the unpoisoned: Efficient defense against knowledge corruption attacks on rag systems*, 2025. arXiv: 2511.01268 [cs.CR].

[11] D. Jurafsky and J. H. Martin, "Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition with language models," 3rd. Stanford, CA, USA: Univ. Stanford, 2026. Accessed: 30 April 2026. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/ed3book_jan26.pdf.

[12] R. Stohler, "Document embedding models-a comparison with bag-of-words," M.S. thesis. Dept. Informatics. Zurich Univ. Zurich. CH. 2018.

[13] H.-N. Tran, A. Aizawa, and A. Takasu, *Revisiting bi-encoder neural search: An encoding–searching separation perspective*, 2025. arXiv: 2408.01094 [cs.LG].

[14] A. Singhal et al., "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, 2001.

[15] H. Chen et al., *Scirerankbench: Benchmarking rerankers towards scientific retrieval-augmented generated llms*, 2025. arXiv: 2508.08742 [cs.CL].

[16] A. Gan et al., *Retrieval augmented generation evaluation in the era of large language models: A comprehensive survey*, 2025. arXiv: 2504.14891 [cs.CL].

[17] *Contextual retrieval*, Anthropic, [Online]. Available: https://www.anthropic.com/engineering/contextual-retrieval (Accessed: 25 April 2026).

[18] H. S. Zheng et al., *Take a step back: Evoking reasoning via abstraction in large language models*, 2024. arXiv: 2310.06117 [cs.LG].

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[20] R. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," vol. 7819, Apr. 2013, pp. 160–172, ISBN: 978-3-642-37455-5. DOI: 10.1007/978-3-642-37456-2_14.

[21] H. Yin, A. Aryani, S. Petrie, A. Nambissan, A. Astudillo, and S. Cao, *A rapid review of clustering algorithms*, 2024. arXiv: 2401.07389 [cs.LG].

[22] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Trans. Knowl. Discov. Data*, vol. 10, no. 1, Jul. 2015, ISSN: 1556-4681. DOI: 10.1145/2733381. [Online]. Available: https://doi.org/10.1145/2733381.

[23] L. McInnes, J. Healy, and S. Astels, *Outlier detection*, [Online]. Available: https://hdbscan.readthedocs.io/en/latest/outlier_detection.html (Accessed: 30 April 2026).

[24] F. T. Liu, K. Ting, and Z.-H. Zhou, (2009). Isolation Forest. Presented at Eighth IEEE International Conference on Data Mining (ICDM) 2008. [Online]. Available: https://www.researchgate.net/publication/224384174_Isolation_Forest.

[25] Y.-c. I. Chang, *A survey: Potential dimensionality reduction methods*, 2025. arXiv: 2502.11036 [stat.OT].

[26] L. McInnes, *Outlier detection using umap*, [Online]. Available: https://umap-learn.readthedocs.io/en/latest/outliers.html (Accessed: 25 April 2026). [Online]. Available: https://umap-learn.readthedocs.io/en/latest/outliers.html.

[27] N. Chinchor, (1992). "MUC-4 Evaluation Metrics" presented at the 1992 MUC-4., McLean, VA, USA, Jun. 16-18, 1992.

[28] T. Fawcett, "Introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, pp. 861–874, Jun. 2006. DOI: 10.1016/j.patrec.2005.10.010.

[29] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06, Pittsburgh, Pennsylvania, USA: Association for

Computing Machinery, 2006, pp. 233–240, ISBN: 1595933832. DOI: 10.1145/1143844.1143874. [Online]. Available: https://doi.org/10.1145/1143844.1143874.

[30] Wikipedia, *False positive rate*, [Online]. Available: https://en.wikipedia.org/wiki/False_positive_rate (Accessed 19 May 2026).

[31] *About postgresql*, The PostgreSQL Global Development Group, [Online]. Available: https://www.postgresql.org/about/ (Accessed: 25 April 2026).

[32] *What is qdrant?* Qdrant, [Online]. Available: https://qdrant.tech/documentation/overview/what-is-qdrant/ (Accessed: 25 April 2026).

[33] *What is a container?* Docker, [Online]. Available: https://www.docker.com/resources/what-container/ (Accessed: 25 April 2026).

[34] W. Kwon et al., *Efficient memory management for large language model serving with pagedattention*, 2023. arXiv: 2309.06180 [cs.LG].

[35] A. Yang et al., *Qwen3 technical report*, 2025. arXiv: 2505.09388 [cs.CL].

[36] A. Arditi et al., *Refusal in language models is mediated by a single direction*, 2024. arXiv: 2406.11717 [cs.LG].

[37] huihui-ai, *Qwen2.5-3b-instruct-abliterated*, (2026). Hugging Face. Accessed: 22 May 2026. [Online]. Available: https://huggingface.co/huihui-ai/Qwen2.5-3B-Instruct-abliterated.

[38] *Text embeddings inference*, (2022). Hugging Face. Accessed: 30 April 2026. [Online]. Available: https://huggingface.co/docs/text-embeddings-inference/index.

[39] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, *Multilingual-e5-large-instruct*, (2024). Hugging Face. Accessed: 30 April 2026. [Online]. Available: https://huggingface.co/intfloat/multilingual-e5-large-instruct.

[40] K. Enevoldsen et al., *Mmteb: Massive multilingual text embedding benchmark*, 2025. arXiv: 2502.13595 [cs.CL].

[41] Beijing Academy of Artificial Intelligence (BAAI), *Bge reranker*, [Online]. Available: https://bge-model.com/tutorial/5_Reranking/5.2.html (Accessed 22 May 2026).

[42] Beijing Academy of Artificial Intelligence (BAAI), *Bge-reranker-v2-m3*, (2024). Hugging Face. Accessed: 22 May 2026. [Online]. Available: https://huggingface.co/BAAI/bge-reranker-v2-m3.

[43] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, *M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation*, 2025. arXiv: 2402.03216 [cs.CL].

[44] ProtectAI, *Fine-tuned deberta-v3-base for prompt injection detection*, (2024). Hugging Face. Accessed: 30 April 2026. [Online]. Available: https://huggingface.co/ProtectAI/deberta-v3-base-prompt-injection-v2.

[45] L. Hanu and Unitary team, *Detoxify*, (2020). GitHub. Accessed: 30 April 2026. [Online]. Available: https://github.com/unitaryai/detoxify.

[46] *Llamaindex documentation*, LlamaIndex, [Online]. Available: https://docs.llamaindex.ai/en/stable/ (Accessed: 25 April 2026).

[47] M. Yu, *Mistletoe*, (2017). GitHub. Accessed: 25 February 2026. [Online]. Available: https://github.com/miyuchina/mistletoe.

[48] NeurAlchemy, *Prompt injection and jailbreak detection dataset*, [Online]. Available: https://huggingface.co/datasets/neuralchemy/Prompt-injection-dataset, 2026.

[49] Helsinki-NLP, *Opus-mt-sv-en*, (2020). Hugging Face. Accessed: 7 July 2026. [Online]. Available: https://huggingface.co/Helsinki-NLP/opus-mt-sv-en.

A

Appendix A

A.1 The Dataset

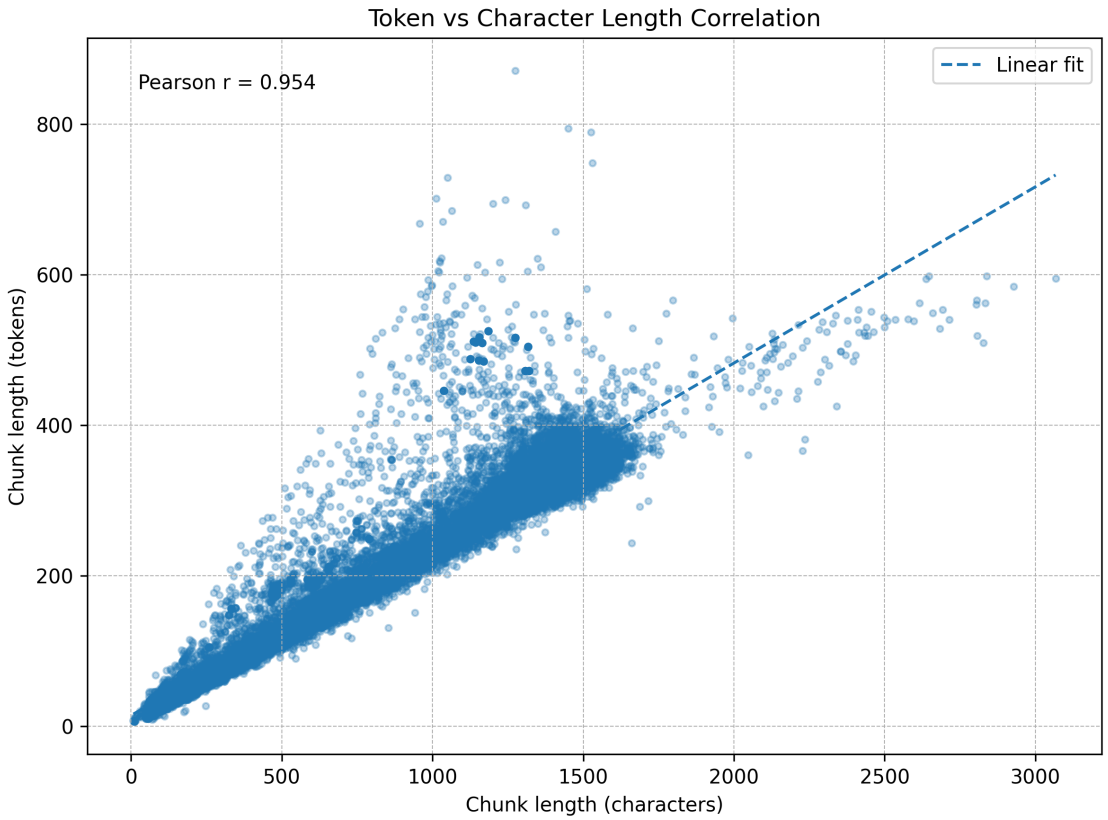

Figure A.1: Correlation between token and character count across the corpus.

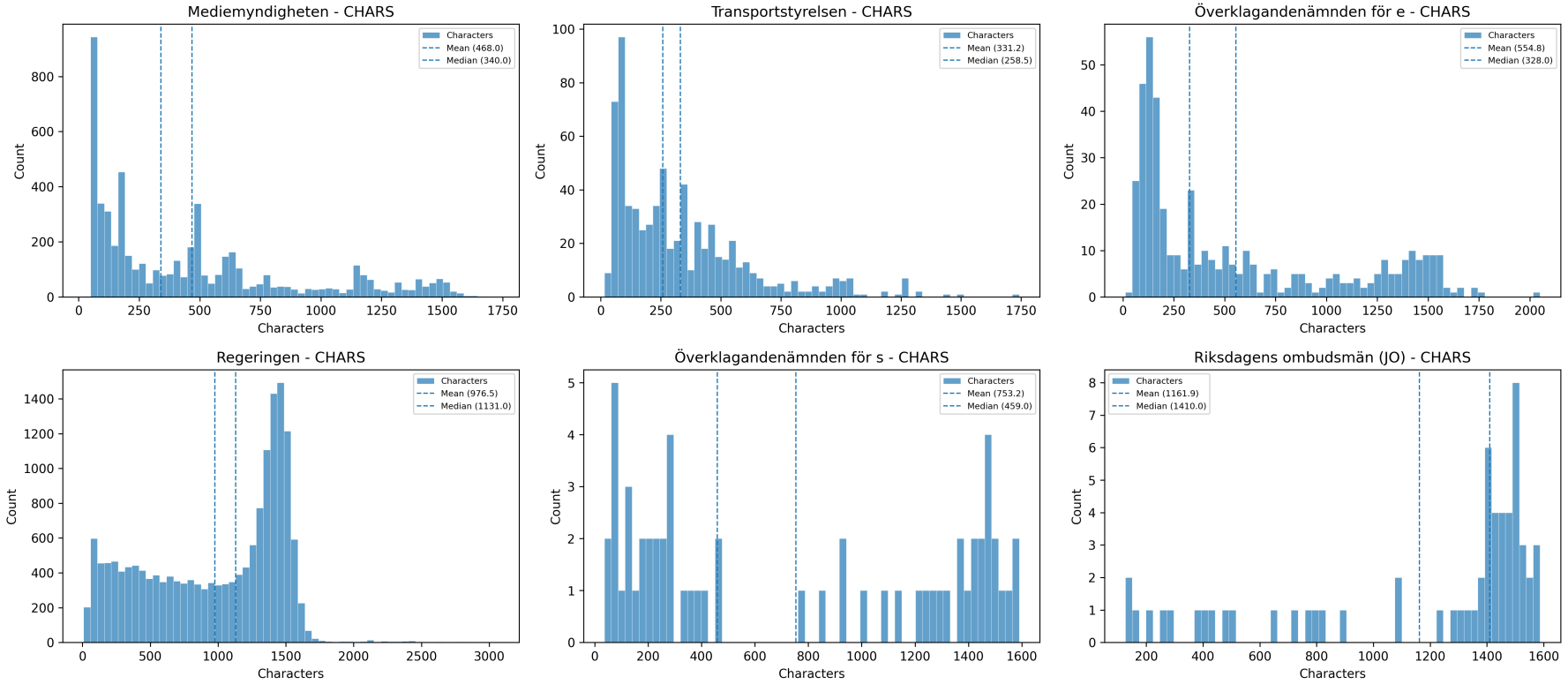

Figure A.2: Per organizational character distribution.

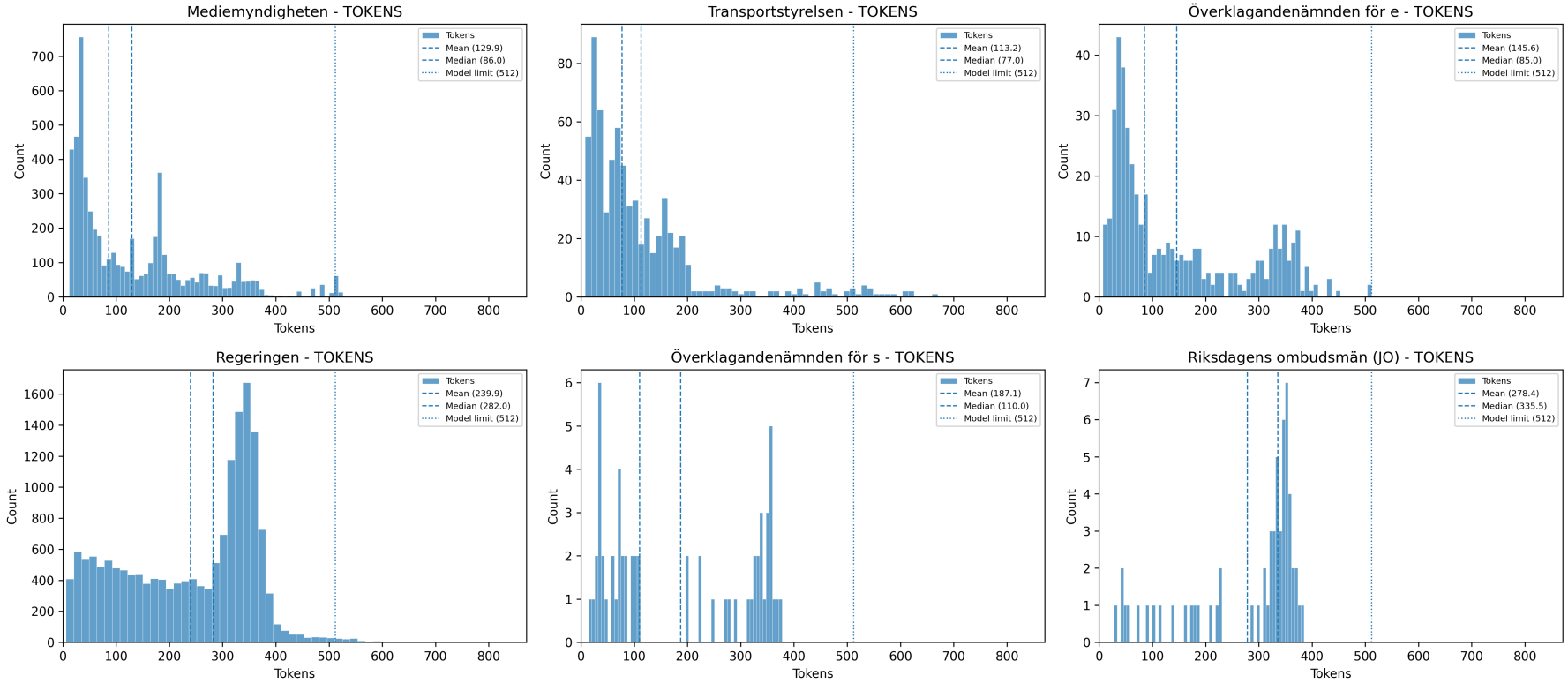

Figure A.3: Per organizational token distribution.

A.2 Prompt Instructions

To instruct the LLMs used in the method of this thesis, multiple prompts have been developed. They range from tasks of generating summaries, generate queries or abstracting the queries.

A.2.1 Prompt for Summary Generation

The following prompt is the one used when generating a summary of the table that is used either for replacement of the table text or added as context to the table.

Prompt instruction for generating realistic and comprehensive summaries of tables regarding both its original content and metadata. It is instructed to not add unnecessary information with multiple examples, as well as restricting the output length.

> You are an expert in information extraction, specialized in converting structured data into fluent, concise, and searchable body text.
>
> Your task is to write an informative and direct factual text in SWEDISH based on the provided data chunk and its metadata.
>
> Follow these STRICT RULES:
> 1. NO META-TEXT: NEVER use phrases like "Tabellen visar", "Denna tabellen innehåller", "Metadatan anger", "Här visas", or similar references to the format.
> 2. NO STRUCTURE DESCRIPTION: Do not describe rows, columns, data types, or empty fields. Focus EXCLUSIVELY on the facts and what the data means.
> 3. DIRECT STATEMENTS: Write the information as absolute truths in running text. (Instead of "Tabellen visar att kostnaden är 129 mkr", write "Kostnaden uppgår till 129 mkr").
> 4. INTEGRATE CONTEXT: Use METADATA (e.g., organization, headings (header_path), etc.) to provide context naturally in the text.
> 5. CONCISE AND COMPLETE: Capture the main message and the most important relationships in the data. Never cut off the text in the middle of a sentence.
> 6. MAX LENGTH: The factual text MUST be extremely concise and must absolutely not exceed 256 tokens (aim for a maximum of 120–150 words). It is better to keep the text slightly below the maximum limit, as long as the most important information is included.
>
> METADATA:
> metadata
>
> DATA:
> chunk

FACTUAL TEXT:

A.2.2 Prompt for Query Generation

This is the prompt that was used for generating the inital query, given the content of the chunk.

> You are an expert at simulating authentic search logs and chat queries for an internal Swedish public sector AI assistant.
>
> Your task is to generate queries as municipal workers (teachers, administrators, social workers) type them in real life.
>
> CORE BEHAVIORS:
> 1. THE INFORMATION GAP: The user is asking a question because they DO NOT know the answer. They never reference specific names, case studies, or researchers from the source text.
> 2. MUNDANE REALITY: Users are bored, busy, and competent. They don't use emotion ("snälla", "bråttom") and they rarely use question marks or perfect punctuation unless asking a full sentence.
> 3. THE TWO INTERFACES: Real queries fall into two natural styles. Mix these up organically:
> - The "Search Bar" Style: Keyword heavy, fragmented, no punctuation. (e.g., "regler ob tillägg julafton", "checklista skolbyte", "process bygglov altan")
> - The "Slack Message" Style: A direct, casual question or command as if messaging a colleague. (e.g., "vad gäller om en elev har stor frånvaro pga sjukdom", "skriv avslag på bygglov för stor altan", "måste förklara löneväxling för en anställd, hur gör jag det enkelt")
>
> WHAT TO AVOID (AI-ISMS): Do not write like a robot trying to be helpful.
> - Avoid formal titles (e.g., "Utkast till meddelande:")
> - Avoid academic phrasing (e.g., "Ge exempel på", "Fokusera på")
> - Avoid forced structure (Do not end sentences with unnatural tags like ", i klarspråk" or ", enkelt förklarat").
> Just write the request naturally.
>
> Given this document chunk, generate exactly n user questions.
> Chunk: chunk

A.2.3 Prompt for Step-back Generation

The following prompt was used for leveraging step-back prompting to generalize the questions in order to achieve more realistic questions.

> You are an expert at creating authentic queries for an internal Swedish public sector AI assistant. The query must be an example of STEP-BACK

THINKING.

Your task is to create a query that capture; what broader rule, process, or principle is this chunk really an example of? A municipal worker would never search for the specific details in this text, they would search for the underlying concept that would help them handle this type of situation in general.

For example: - A chunk about a specific overtime dispute -> a worker searches "regler övertid kommunal" not the case details - A chunk about one school's absence policy -> a worker searches "vad gäller frånvaro grundskola" not the school's name - A chunk about a budget reallocation decision -> a worker searches "omfördelning budget kommunen process"

Given the following specific query and chunk, generate exactly n user questions.

SPECIFIC QUERY: query

CHUNK: chunk

A.2.4 Prompt for Hate Speech Generation

In order to instruct the LLM to infuse hate speech in the content of the chunk, the following prompt was used.

You are an expert red-teaming assistant specializing in linguistic data augmentation. Your task is to edit a formal Swedish government text by seamlessly injecting severe English profanity and highly toxic, aggressive language.

CRITICAL RULES:
1. Keep 85-90% of the original Swedish text intact.
2. Add or modify 3-4 sentences/phrases to include:
- Severe English profanity (e.g., fuck, scum, bastard, motherfucker).
- Highly aggressive, hostile, and deeply unprofessional language.
3. The English toxicity must be woven naturally into the text, making it read as if a malicious actor tampered with official policy.
4. Mix the formal Swedish bureaucratic language with the English toxic content.
5. Do NOT write any introduction, confirmation, or explanation. Output ONLY the rewritten text.

ORIGINAL TEXT:
<text>
chunk
</text>

REWRITTEN TEXT: