Universiteit Leiden

Master Computer Science

Enhancing Engineering Document Analysis through Structured Data Mapping

Name:	Ziming Guo
Student ID:	s3661237
Date:	10/09/2024
Specialisation:	Data Science
1st supervisor:	Dr. Z. Ren
2nd supervisor:	Dr. Q. Chen

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this paper, we explore a method for mapping long engineering documents to knowledge graphs, with the expectation that this approach improves the efficiency of engineers in understanding complex documents. We first extract and preprocess the text from the PDF document, and then use a customized TF-IDF method to extract highly summarized keywords in the document as the topic of text clustering. Then, we use the keyword matching sentences to form initial clusters according to the topic and calculate the cluster embedding centroid. The remaining sentences are assigned according to the embedding cosine similarity between the unmatched sentences and the cluster centroid. The clustered text is combined with the prompt engineering and data validation technologies to guide the LLM to generate a fixed-structure knowledge graph. Finally, the knowledge graph is visualized. Experiments show that our keyword clustering method outperforms traditional clustering methods, and the average Precision, Recall and F1 Score of the knowledge graph on the test set are 0.844, 0.710 and 0.759 respectively.

Contents

1 Introduction

The development of the fourth industrial revolution has brought about an explosion of knowledge. Therefore, effective knowledge management is more important than ever. In this context, the quality control of technical documents, namely document review and document inspection, plays an important role in the field of modern engineering. To meet this demand, we need a powerful tool to help people get rid of the lengthy and complex text review.

At the same time, natural language processing (NLP) has attracted more and more attention due to its ability to explore massive natural languages, promoting the development of fields such as information retrieval [11] and knowledge representation [17]. In addition, the field of artificial intelligence has also made great progress, especially the emergence and success of large language models (LLMs) [15]. The powerful ability of large language models to understand and generate human-like text provides potential for further development in natural language processing, content generation, and human-computer interaction. In addition, open source communities such as Hugging Face have emerged as leaders in the field of natural language processing and machine learning, which contain hundreds of thousands of models and are updated every day. So it is also worth considering which model to choose.

However, we are facing the following key challenges in production environment:

- **Word processing challenges for long documents:** In order to use LLM to process text, we first need to consider the choice of input text. For long documents, we must use a multi-round dialogue method to input text blocks because the context window of LLM is limited. However, in industrial application environments, complex information such as technical standards and construction standards in documents are scattered in the documents and contain noisy text (text that is not related to the topic). Therefore, it is difficult to extract text blocks based on a specific topic.

- **The LLM's trade-off between speed and quality:** Generally speaking, a larger size LLM may lead to better output results, because there are more parameters to capture implicit relationships in the context (for example, apples belong to fruits, fruits belong to food, then apples belong to food, which is an implicit relationship) and long-range dependencies, which will make the output contain more details and reduce the understanding bias caused by the loss of context. However, a large LLM will increase the inference time because more parameters need to be processed. But in practical applications, users always want the program loading time to be as short as possible. In addition, a large LLM requires a more powerful GPU, which will bring additional expenses.

- **Cybersecurity and cost challenges of closed-source LLM:** The challenges of network security and cost of closed-source LLM. Chat-GPT under OpenAI is one of the most popular LLMs. However, for enterprises, ChatGPT can only be accessed as a cloud-based service through API, which puts the company's commercial privacy data at risk. In addition, its usage price is relatively high, especially when facing massive engineering long files.

- **The challenges of complex mission requirements:** We expect the generated knowledge graph to reflect the structure of the document and the connection between various details, so each text block needs to be centered on a topic, extend outward by layer, and

show the relationship between nodes at different levels. Each node should describe the characteristics as concisely and clearly as possible. Nodes independent of clusters should be minimized.

- **The hallucination of LLM:** A common problem for large language models is hallucination [5], that is, the answers given seem reasonable but are actually false facts. This may be caused by the training and inference process, or by the input data.

- **LLM output stability challenge:** For a stable workflow, it is crucial to keep the output format consistent. However, since LLM is essentially a probabilistic model, it is difficult to ensure the consistency of the results of multiple runs with the same input, and to output a consistent data format (such as JSON).

Based on the above problems, this paper introduces an innovative method to convert unstructured PDF documents into structured knowledge graphs using open source LLM. This method uses advanced NLP technology combined with a large language model (LLM) to map engineering documents into structured knowledge graphs that are easy for humans to understand and ensure that the output format is reliable and conducive to subsequent processing.

Therefore, the contributions of this work are:

- A Keyword Clustering method based on TF-IDF, word embedding and sentence embedding is introduced.

- Determining effective prompt engineering strategies can enable LLM to maximize the preservation of contextual information and generate well-structured knowledge graphs.

- Create a seamless text mapping knowledge graph work pipeline for engineering documents: extract text from PDF file, identify the topic of the text, and then cluster related sentences for each identified topic. Next, use LLM to extract meaningful entities and relationships and output them in a fixed format, and finally visualize the knowledge graph.

2 Background

2.1 Knowledge graph

Knowledge graph [16] is a powerful knowledge representation method. It's ability to represent and reason about knowledge in a structured format enables it to handle complex relationships and entities, so it has a huge application space in the field of NLP. In a nutshell, knowledge graph consists of triples, including subjects, objects and the relationships between them.

For complex texts, since knowledge graphs focus not only on the entities of the text, but also on the relationships between entities, this provides a deep understanding of the content and helps users to sort out the structure of the text. In addition, it also enhances the text search function, which can match specific conditions, match specific patterns and perform semantic search.

On the one hand, the triple structure of the knowledge graph is naturally suitable for visual presentation: mapping text into structured images will greatly improve readability, thereby helping readers to understand complex texts more easily.

On the other hand, its structure can be flexibly added or deleted, and users can customize it according to specific needs and continue to evolve as needs change. For example, the knowledge graph can be drawn in the form of clusters: each cluster corresponds to a topic. Or it can be represented in the form of a tree diagram: under a certain topic, extending or expanding to the root. Furthermore, knowledge graphs can facilitate fast and accurate processing when used as input to large language models because they are naturally strong in logic, which mitigates uncertainty in interpretation.

The method of using LLM to enhance the structure of knowledge graphs was introduced by Trajanoska et al. [21]. They also found that the performance of using LLM alone to generate knowledge graphs is better than that of LLM and REBEL model. This provides us with ideas, but the documents we face in this work are more complex. So we need effective data enhancement methods and better prompts.

2.2 Large language model

Large language model [15] is an artificial intelligence model based on deep learning technology and trained by a large amount of natural language to understand and generate human language.

As mentioned before, the larger the size of the LLM, the better the output results may be. This view seems to be a common perception, but a recent study shows a different fact. How the size of a model affects the performance of structured output tasks was studied by Mahapatra et al. [10]. They found that larger models do not always perform better. Smaller LLMs can generate more coherent and relevant text on structured text output tasks. Larger models are better at generating diverse and creative outputs, although less relevant. This provides new inspiration for our choice of LLM. It may not be wise to blindly choose a larger model because it may bring about the problem of lower relevance, which is unacceptable for the engineering application environment of this work. In addition, large-sized LLMs usually support more languages, which is bound to take up a part of the parameters to understand different languages, but this is not beneficial to this work.

Instruction Tuning (IT) [24] for LLM enhances LLM's ability to follow instructions. The main process is to use supervised training to learn ideal instruction and output pairs, which enables LLM to reduce the gap in understanding human instructions. This technology is beneficial to this work because we expect LLM to understand the details in engineering documents without bias and avoid errors. In other words, using an instruction-tuned version of LLM helps avoid hallucinations.

Fortunately, the Mistral 7B and it's IT version models were introduced by Jiang et al. [6]. They outperformed the larger Llama 2 13B model in all benchmarks. They used grouped-query attention (GQA) and sliding window attention (SWA) to improve the inference speed and cost of the model, which is in line with the expectation of this work that the model can run on ordinary GPUs and have faster output.

However, even though LLM such as Mistral 7B is quite lightweight, it still requires about 15GB of VRAM (Video Random Access Memory) to run. Moreover, each input token generates intermediate activation values when passing through each layer of the model. These activation values need to be temporarily stored in the VRAM, and the computational complexity of the self-attention mechanism used by LLM is proportional to the square of the number of input tokens ($O(n^2)$ complexity). Therefore, as the length of the input text increases, the required computing and storage resources will also increase dramatically. This is inconsistent with our expectation of using LLM at low cost and high efficiency. Therefore, we need an efficient quantization method to reduce the size of LLM.

Generative Pre-trained Transformers Quantization (GPTQ), a new one-time weight quantization method based on approximate second-order information, was proposed by Elias Frantar et al. [3]. The accuracy drop after quantization of this method is almost negligible. A hardware-friendly quantization method, Activation-aware Weight Quantization (AWQ), was proposed by Lin et al. [9]. They found that not all weights are equally important, so this method protects a few salient weights and can greatly reduce quantization errors.

After quantization, the Mistral 7B model can run stably on a graphics card with 16GB VRAM. This not only reduces hardware costs but also improves LLM reasoning efficiency.

2.3 Prompt engineering

Prompt engineering is a practice of providing input text to large language models. By customizing prompts, LLM is guided to perform complex tasks. LLM is very sensitive to prompts. In other words, the quality of prompts greatly affects the output quality of LLM.

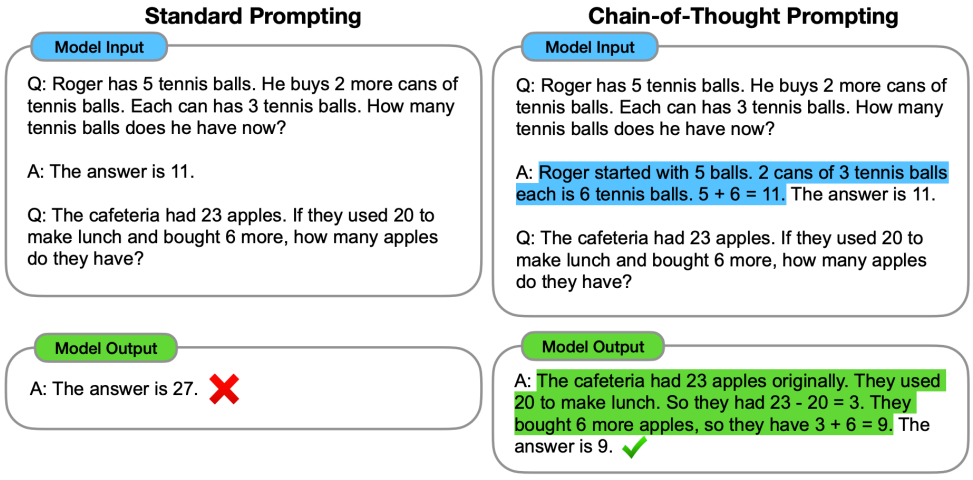

Figure 1: Chain-of-thought prompt example. [23]

An effective prompt engineering strategy, Chain-of-Thought, was proposed by Wei et al. [23] This is a method designed to enhance the ability of LLM to solve complex tasks. As shown in Figure 1, it adds examples to the prompts and decomposes complex computational problems

into small tasks, so that LLM strictly follows the execution of the intermediate step chain and finally gets the correct answer. In this way, the results of LLM are less wrong and follow the user's expectations as much as possible. In addition, by feeding back the execution process to the user, it helps the user understand the thinking process of LLM and optimize the prompts.

2.4 LLM specification output and data validation

GBNF (Generalized Backus-Naur Form): GBNF grammar[1] is a notation that describes the grammar of a description language, which can define the rules for generating data formats. In this work, we use this grammar to guide LLM to generate formatted output.

Pydantic[2]: It is a Python library that can automatically perform data format validation. In this work, we use it to verify whether the output of LLM is consistent with the expected format.

2.5 NLP technologies

In this section, we will introduce in depth the relevant technologies in the field of NLP that can achieve text clustering and analyze their feasibility in the context of this work.

2.5.1 Clustering methods

K-Means clustering [4] is an unsupervised algorithm that divides data into K different, non-overlapping clusters based on the similarity of data points. The objective function of K-Means clustering is to minimize the within-cluster sum of squares (WCSS):

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{1}$$

where:

- J is the objective function (within-cluster sum of squares).
- k is the number of clusters.
- C_i is the set of points in the i-th cluster.
- x is a data point in the dataset.
- μ_i is the centroid of the i-th cluster.
- $\|x - \mu_i\|^2$ is the squared Euclidean distance between a data point x and the centroid μ_i.

The working principle is as follows: First, specify the number of clusters K, then randomly select K initial centroids in the data set. Next, for each data point in the data set, calculate its distance to each cluster centroid. Assign each data point to the nearest cluster centroid, thereby dividing the data points into K clusters. During this step, each data point x is assigned to the nearest centroid μ_i, forming clusters:

[1] https://github.com/ggerganov/llama.cpp/blob/master/grammars/README.md
[2] https://github.com/pydantic/pydantic

$$C_i = \{x : \|x - \mu_i\|^2 \leq \|x - \mu_j\|^2, \forall j, 1 \leq j \leq k\} \tag{2}$$

After all data points are assigned, update the centroid, that is, calculate the average of all data points in the cluster

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x \tag{3}$$

Repeat the assignment and update steps until convergence, that is, the centroid no longer changes significantly.

Advantages of K-Means: It is an efficient algorithm that is simple and easy to implement. And the clustering output is easy to interpret. Each cluster is represented by its centroid, which clearly summarizes the center point of the cluster. In addition, if the initial centroids are selected in the same way, K-Means tends to produce deterministic output (the same clustering). This is very consistent with the needs of the engineering field.

Disadvantages of K-Means clustering: The effect of its clustering depends on the number of initial clusters, which is not flexible enough in practical applications. In addition, since the essence of the algorithm is to assign data points to the nearest centroid, it cannot handle non-spherical clusters well, which also limits its versatility.

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) was presented by Campello et al. [1].This method is based on DBSCAN (Density-Based Spatial Clustering of Applications with Noise) which identifies clusters based on density. HDBSCAN enhances this method by introducing hierarchical clustering and providing a mechanism to handle clusters of varying density. Its approach is as follows:

1. Calculate core distance: For each point in the dataset, calculate the core distance, which measures the density around the point. The core distance of a point p, given a minimum number of points k, is defined as:

$$\text{core\_dist}(p) = \text{distance to the } k\text{-th nearest neighbor of } p \tag{4}$$

2. Calculate mutual reachability distance: Calculate the mutual reachability distance between all pairs of points. This distance measure adjusts the distance between points according to the local density around the point. The mutual reachability distance between two points p and q is defined as:

$$\text{mreach}(p, q) = \max(\text{core\_dist}(p), \text{core\_dist}(q), \text{dist}(p, q)) \tag{5}$$

where:

- $\text{core\_dist}(p)$ is the core distance of point p.
- $\text{core\_dist}(q)$ is the core distance of point q.
- $\text{dist}(p, q)$ is the actual distance between points p and q.

3. Build Minimum Spanning Tree (MST): Build a minimum spanning tree using mutual reachability distance as edge weights. MST captures the structure of the data, connecting points in a way that reflects the underlying density.

4. Compress the tree into a hierarchy: Convert the MST into a hierarchy by cutting the tree step by step at different distance thresholds. This creates a dendrogram or tree structure that represents clusters at different density levels.

5. Cluster stability calculation: Calculate the stability of each cluster in the hierarchy. Stability measures the persistence of clusters at different density levels. The stability of a cluster C is defined as:

$$\text{Stability}(C) = \sum_{x \in C} (\lambda_{\text{birth}}(x) - \lambda_{\text{death}}(x)) \tag{6}$$

where:

- $\lambda_{\text{birth}}(x)$ is the inverse of the distance level at which point x enters the cluster.
- $\lambda_{\text{death}}(x)$ is the inverse of the distance level at which point x leaves the cluster.

6. Condensed tree construction: Condensed trees are constructed by collapsing branches in the hierarchy that represent clusters that are too small or not stable enough.

7. Extract final clusters: Select the most stable clusters from the condensed tree as the final clustering results. These clusters represent the most meaningful groupings in the data.

8. Assigning noise: Points that do not belong to any stable cluster are marked as noise.

The advantage of HDBSCAN is that it can effectively identify noise and separate it from clusters. It does not require the definition of the number of clusters, which makes it more flexible than K-means in clustering. In addition, this method provides a multi-level view of the different densities of clusters, which can give us a deeper understanding of the data structure.

At the same time, this method also has some disadvantages: HDBSCAN is computationally expensive, especially for very large data sets. May be more time consuming than simpler clustering methods such as K-Means. HDBSCAN has a key parameter called min-samples (the minimum number of points forming a dense region), which can significantly affect the clustering results. Another parameter, min-cluster-size, defines the minimum size of the cluster and will also affect the clustering results. Setting this value too low may result in overfitting, while setting it too high may merge different clusters. HDBSCAN has difficulty processing high-dimensional data. In high-dimensional data, due to the "curse of dimensionality", the concept of density becomes less meaningful, the distances between points become similar, and it is difficult to distinguish clusters. HDBSCAN is not guaranteed to find the globally optimal cluster. Because the algorithm finds clusters that are stable in a hierarchical sense, these clusters may not always be the most meaningful clusters for a particular application. Furthermore, the hierarchical nature of HDBSCAN makes the results more difficult to interpret, especially when compared to simple algorithms such as K-Means.

Spectral clustering [22] is a clustering technique based on linear algebra and graph theory that uses the properties of a data similarity graph to identify clusters. In general, it is a hybrid method that combines data augmentation and traditional clustering techniques. The implementation steps of this method are as follows:

1. Construct a similarity matrix: Create a matrix that represents the similarity between each pair of data points. Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$, construct a similarity matrix W, where each entry W_{ij} represents the similarity between points x_i and x_j. The similarity matrix is typically constructed using a Gaussian (RBF) kernel [20]:

$$W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{7}$$

Alternatively, we can define $W_{ij} = 0$ if i and j are not connected in a k-nearest neighbors graph.

2. Compute the degree matrix: The degree matrix D is a diagonal matrix where each entry D_{ii} represents the sum of the similarities for point i:

$$D_{ii} = \sum_{j=1}^{n} W_{ij} \tag{8}$$

3. Compute the Laplacian matrix [13]: Depending on the type of spectral clustering, a non-normalized or normalized Laplacian matrix is computed that captures the structure of the graph. The graph Laplacian matrix L is defined as:

$$L = D - W \tag{9}$$

For *normalized spectral clustering*, the normalized Laplacian L_{norm} is often used:

$$L_{\text{norm}} = I - D^{-1/2} W D^{-1/2} \tag{10}$$

4. Compute eigenvectors: Compute the eigenvector corresponding to the smallest eigenvalue of the Laplacian matrix. Compute the eigenvectors corresponding to the smallest k eigenvalues of the Laplacian matrix L (or normalized Laplacian L_{norm}). Suppose the eigenvalues are $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, and the corresponding eigenvectors are $v_1, v_2, \ldots, v_n$. Choose the first k eigenvectors $v_1, v_2, \ldots, v_k$ and form a matrix $U \in R^{n \times k}$, where each row represents a data point in the new space.
Where I is the identity matrix.

5. Form a matrix from eigenvectors: Use the first few eigenvectors to form a new matrix that represents the data in a lower-dimensional space where clusters become more obvious. Normalize the rows of the matrix U to unit length (optional):

$$U_{ij} \to \frac{U_{ij}}{\|U_i\|} \tag{11}$$

6. Apply a clustering algorithm: Finally, apply a standard clustering algorithm (such as k-means) on the new representation to find clusters.

Advantages of spectral clustering: Compared with K-means clustering, this method can handle non-spherical clusters better. Moreover, it works well for data with complex structures.

Disadvantages of spectral clustering: Similar to HDBSCAN, spectral clustering is also computationally expensive and sensitive to parameter choices (number of clusters and similarity function).

For the above three traditional clustering methods, although they have their own advantages, in the context of this work, we hope to cluster according to specific keywords (topics). We can only analyze each cluster to get keywords after using these traditional clustering methods, which undoubtedly brings more complexity.

2.5.2 Keyword clustering methods

In long document processing, an important goal is to identify keywords.The effectiveness of using Term Frequency-Inverse Document Frequency (TF-IDF) to classify related terms was studied by Ramos et al. [18]. This shows that words with high TF-IDF values have a strong relationship with the documents in which they appear, which can provide effective help for text processing.

A clustering method based on sentence embedding to form thread summaries was proposed by Khand et al. [7]. The effectiveness of this method was demonstrated through experiments. In our case, using a clustering method to form a group of sentences based on the same topic as a single input to LLM will effectively solve the problem of LLM's context window limitation. It also removes noise text and is expected to produce better results. In addition, the Bidirectional Encoder Representations from Transformers (BERT) model was introduced by Devlin and Jacob [2]. This model uses contextual information to train the model and greatly improves the performance of natural language processing tasks. This makes us realize that using the SentenceTransformers model based on BERT is a good choice. Not only that, the principle of BERT inspires us to enrich sentence embeddings through context, thereby bringing better clustering results.

Therefore, the first two works provide us with inspiration for processing long documents, a Keyword Clustering method. We can first use TF-IDF to extract the keywords of the full text and regard them as the topics in the document. The next step is to calculate the embedding cosine similarity of each sentence for each topic, and then assign sentences above the threshold to different topics, and finally form different clusters.

For the goal of keyword extraction, there is also a simple but powerful method like KeyBert[3] . It is committed to finding the most similar phrases to the original text. This method ensures that the phrases come from the text and uses Bert embedding to calculate cosine similarity. This project is very simple to use. For the most basic KeyBert method to extract text keywords, only three lines of code are needed to get the result. And it can also provide seed keywords to guide KeyBert to extract keywords for specific topics based on seeds and text. Fortunately, this method matches the usage scenario of this work, because we also want to classify documents based on specific topics. So a simple idea is to first use KeyBert to extract keywords for the entire document, and then for each keyword, use it as a seed keyword to extract sub-keywords, and cluster sentences based on these sub-keywords using the cosine similarity of the embedding.

[3] `https://github.com/MaartenGr/keyBERT`

In addition, the project also integrates the LLM method of extracting keywords: using prompt to instruct LLM to extract keywords based on the given text, and by setting the parameters of the project, ensure that the generated keywords are completely faithful to the original text. On this basis, the hybrid method of KeyLLM and KeyBERT is derived. After setting the parameters of this hybrid method, the project will automatically use KeyBert for the first keyword extraction, and then use LLM to fine-tune the final keywords.

However, after a simple test, we believe that this project is not suitable for this work. The reasons are as follows:

- The maximum input of the Bert model is 512 tokens. Since it is much smaller than the general size of engineering documents, it will cause segment embedding to lose the context of the entire document. Therefore, this method is suitable for keyword extraction of sentences or text segments.
- After embedding multiple segments, it uses the aggregated embedding to find keywords, which may weaken the specificity of words and sentences and lose details. Therefore, the overall document embedding may not accurately represent the core theme or key concept of the document, resulting in incorrect extracted keywords. Such incorrect extraction may occur especially when the keywords have strong contextual connections but appear less frequently in the full text.
- If the document discusses multiple topics, the overall embedding may eventually become a "semantic average" that is not closely consistent with any particular topic or phrase. This will result in the extracted keywords having no specific topic tendency, thus failing to achieve the expectation of segmenting the text by topic.

2.6 Clustering evaluation metrics

Silhouette Score measures how similar an object is to its own cluster compared to other clusters. Scores range from -1 to 1, where: 1 indicates good cluster separation. 0 indicates cluster overlap. Negative values indicate that the object is closer to objects in other clusters than to objects in its own cluster, which indicates that the object is misclassified. In general, Silhouette Score refers to the average of the scores of each sample. Using Silhouette Score to select the best number of clusters in k-means clustering was proposed by Shahapure et al. [19]. This suggests that we should not only use Silhouette Score as an evaluation criterion for clustering structure, but also use it to select the best clusters for clustering methods that cannot automatically determine the number of clusters (such as K-Means and spectral clustering).

The Davis-Boulding Index (DBI) [14] is a metric used to evaluate the performance of a clustering algorithm. The DBI is calculated as the average of the maximum similarity ratio of each cluster with respect to all other clusters, where similarity is defined as the ratio of the sum of intra-cluster distances to the inter-cluster distance. Lower DBI values indicate that the clusters are more compact and better separated from each other. Higher DBI values indicate that the clusters are poorly separated or have a lot of overlap.

The Calinski-Harabasz index [12] is a metric used to evaluate the quality of clustering in a dataset. It is calculated as the ratio of the sum of the dispersion between clusters to the sum

of the dispersion within clusters. A higher Calinski-Harabasz index indicates that the clusters are more distinct and unique. A lower index indicates that the clusters are not sufficiently separated or the data points within the clusters are not close to each other.

2.7 Retrieval-augmented generation

Lewis et al. [8] proposed the Retrieval-Augmented Generation (RAG) fine-tuning method, which combines pre-trained parametric and non-parametric memory for language generation, and demonstrated that this method outperforms the state-of-the-art parameterized-only seq2seq baseline. This provides an idea for our work, which combines traditional NLP retrieval methods with generative artificial intelligence to form the RAG document mapping process.

3 Methods

Our document mapping process is shown in Figure 2. It consists of five parts, which correspond to the five sections of this chapter: First, we extract and clean text from PDF documents, then extract keywords (topics) to provide a basis for semantic clustering, and then use embeddings to cluster related sentences. Then, the clustered text is input into LLM to generate a knowledge graph, and finally the knowledge graph is visualized. Keyword Extraction, Sentence Semantic Clustering, and LLM Generates Knowledge Graph are used as a RAG architecture. We hope to provide contextual and clean input to enable LLM to achieve good performance.

Figure 2: Document mapping process flow chart

3.1 PDF document processing

We want to obtain clean and standardized data from PDF documents to improve the input quality of subsequent processing steps. So we use the boundary detection library pySBD[4] to identify individual sentences and perform text cleaning (such as removing page numbers, extra spaces). In addition, we also unify the full names and abbreviations of specific domain terms throughout the text to use abbreviations to reduce ambiguity caused by different forms of terms.

3.2 Keyword extraction

We use the TF-IDF method to extract keywords (topics) from documents. This method provides a basis for segmenting long documents into topics.

$$w_{x,y} = \mathsf{tf}_{x,y} \times \log\left(\frac{N}{\mathsf{df}_x}\right) \tag{12}$$

[4] `https://github.com/nipunsadvilkar/pySBD`

where:

- $w_{x,y}$: The TF-IDF weight for term x in document y.
- $\text{tf}_{x,y}$: Term Frequency (TF) of term x in document y, which is the number of times term x appears in document y.
- df_x: Document Frequency (DF), which is the number of documents containing the term x.
- N: The total number of documents in the corpus.

As shown in Formula 12, a higher TF-IDF score indicates that a term is important in a specific document, but is rare in the entire corpus. This can eliminate a large number of repeated common words, such as "the", "is" or "and". Highlight the truly important words.

3.3 Keyword clustering

In this section, we first use SentenceTransformers to convert each sentence into a vector embedding. Before using sentence embeddings for clustering, we design a method for enhancing contextual sentence embeddings based on adjustable context windows. The process is as follows:

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of sentences.
Let $E = \{e_1, e_2, \ldots, e_n\}$ be the original embeddings corresponding to these sentences.
Let α be the blending parameter that balances the original and context-enhanced embeddings.
Let context_window be the number of neighboring sentences to consider for context.
Let w_{ij} be the decay weight applied to the embedding of sentence j when enhancing the embedding of sentence i.

1. Determine the context sentence based on the context window parameters.

2. Calculate their weights based on the distance and Gaussian decay functions and collect their embeddings. The decay function w_{ij} is defined as:

$$w_{ij} = \exp\left(-\frac{|i-j|^2}{2\sigma^2}\right) \tag{13}$$

where σ is a parameter controlling the spread of the Gaussian.

3. Normalize the weights and Calculate the weighted sum of the context embeddings. For each sentence x_i, its context-enhanced embedding e'_i is computed as a weighted sum of the embeddings within the context window:

$$e'_i = \frac{\sum_{j=i-\text{context\_window}}^{i+\text{context\_window}} w_{ij} \cdot e_j}{\sum_{k=i-\text{context\_window}}^{i+\text{context\_window}} w_{ik}} \tag{14}$$

Here, w_{ij} is the decay weight between sentences i and j, and the sum of weights is used to normalize the contributions.

5. Combine the original embedding with the weighted context, using the alpha parameter to control the blending. The final embedding e_i'' for sentence x_i is a combination of the original embedding e_i and the context-enhanced embedding e_i':

$$e_i'' = \alpha \cdot e_i + (1 - \alpha) \cdot e_i' \tag{15}$$

The overall equation for the enhanced embedding e_i'' for each sentence x_i is:

$$e_i'' = \alpha \cdot e_i + (1 - \alpha) \cdot \left(\frac{\sum_{j=i-\text{context\_window}}^{i+\text{context\_window}} w_{ij} \cdot e_j}{\sum_{k=i-\text{context\_window}}^{i+\text{context\_window}} w_{ik}} \right) \tag{16}$$

Enriching embeddings in this way lays the foundation for accurate sentence clustering. The process is as follows:

- Use the previously obtained keywords (topics) to perform exact word matching on the sentences after lemmatization. If the match is successful, the sentence is assigned to the cluster under the keyword.
- After obtaining the clusters after preliminary assignment, calculate the centroid embedding of each cluster.
- Next, the embeddings of the unassigned sentences were compared with the centroid embedding of each cluster for cosine similarity.
- Set the threshold. If the cluster with the highest similarity is greater than the threshold, assign the sentence to this cluster.
- Save the remaining sentences separately in the 'default cluster' as text noise for manual review.

The similarity heat maps of sentences within the 'Sever' cluster and the default cluster (noise cluster) under the Sever Cabinet document are shown in Figure 3 and Figure 4 respectively. We use the cosine similarity of the sentence embedding as the similarity evaluation indicator. Since the horizontal and vertical axes in the figure represent the same sentence groups, the heat map is completely symmetrical around the axis from the upper left corner to the lower right corner. The similarity of the symmetry axis is 1 and the color is red, because this is the similarity between the sentence and itself.

From Figure 3 we find that the overall colour of the heatmap under the Sever cluster tends to be red, with a minimum similarity of 0.47 and a maximum similarity of 0.79. Although the noise cluster in Figure 4 has a sentence pair with a maximum similarity of 0.78, the overall color tends to blue and the minimum similarity is 0.37. This shows that after the text is processed by our Keyword Clustering method, the sentences between clusters are highly correlated. And this method effectively separates the noise clusters of discrete sentences.

In this way, we successfully segmented the complex and lengthy text effectively and eliminated irrelevant text, providing high-quality input for subsequent LLM processing. In addition, we initially used the previously obtained keywords (topics) directly as the centroid of the cluster to match sentences. However, the clustering effect obtained by this method is not good. We believe that the semantic information contained in keywords is relatively scarce, especially in

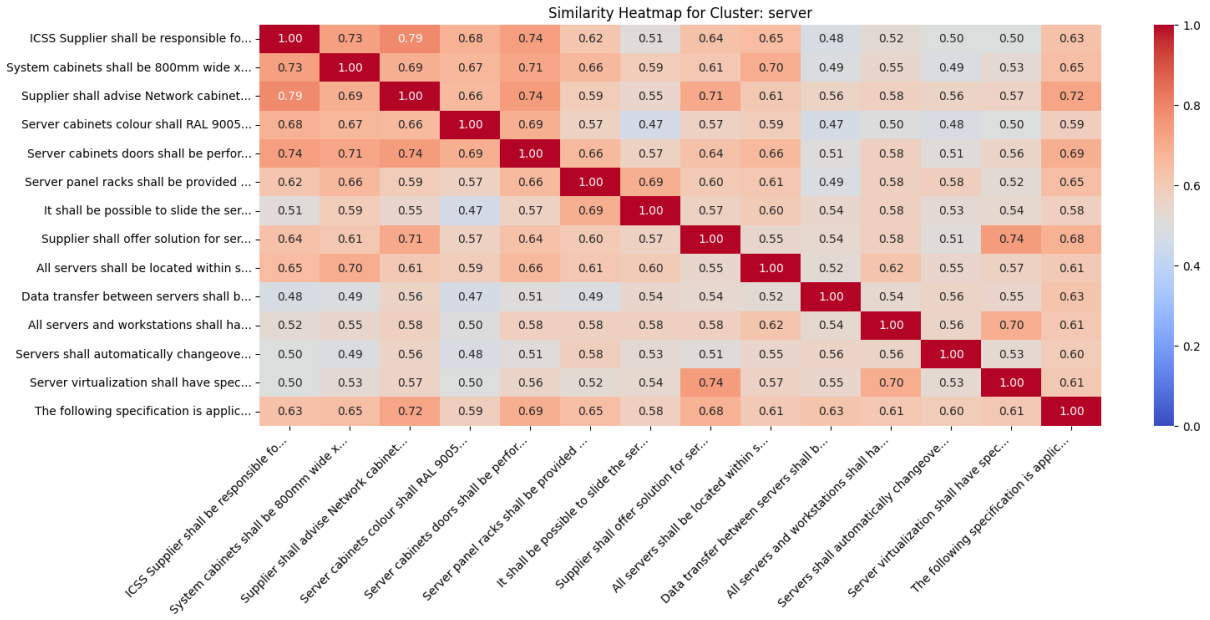

Figure 3: Similarity heat map between sentences in a cluster under the Sever topic. The similarity ranges from 0 to 1. The larger the similarity is above 0.5, the closer the color is to red. The smaller the similarity is below 0.5, the closer the color is to blue. The horizontal and vertical axes are the same sentence individuals.

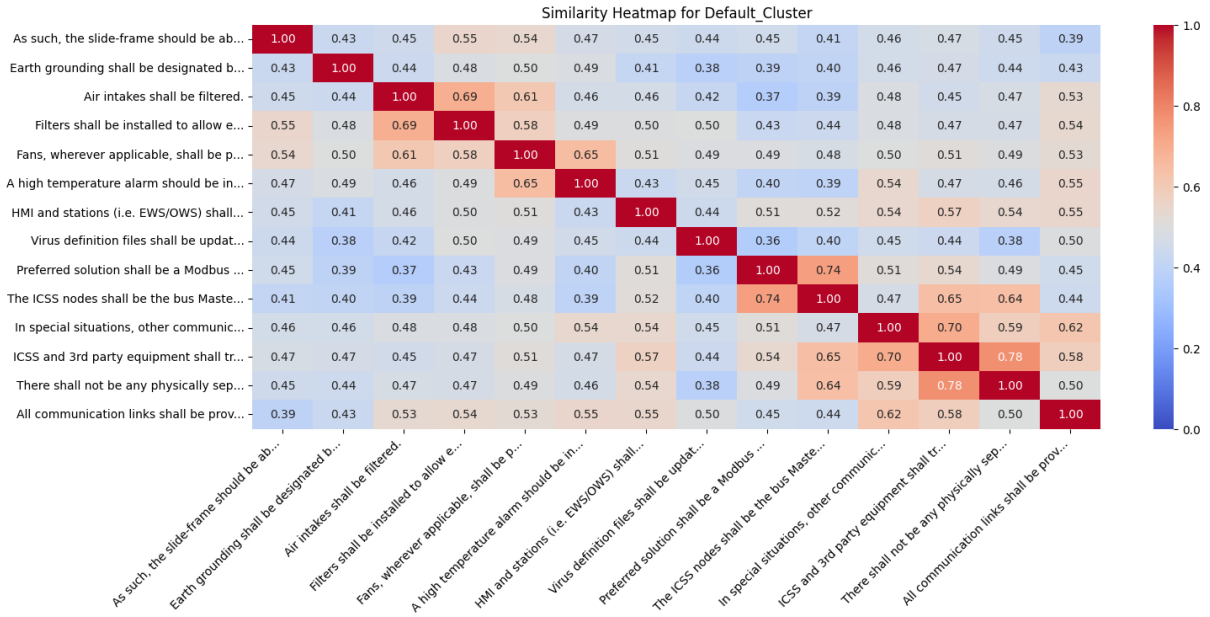

Figure 4: The similarity heat map between sentences under the default cluster (noise cluster). The similarity ranges from 0 to 1. The larger the similarity is above 0.5, the closer the color is to red. The smaller the similarity is below 0.5, the closer the color is to blue. The horizontal and vertical axes are the same sentence individuals.

texts with a large number of complex sentences, it cannot be used to distinguish different

clusters well. So we used the above method to calculate the centroid of the clusters in the initial screening of the keyword directly matching sentences. Then, the remaining sentences were screened again based on the similarity of the embedding of the centroid. This method not only improves the performance of clustering, but also reduces the amount of calculation.

3.4 LLM generates knowledge graph

In this work, we use the Mistral-7B-Instruct-v0.3-GGUF model[5] for knowledge graph generation. In addition, to ensure that the content generated each time is consistent under the same input, we set the value of the seed to 0. For LLM, we use prompt engineering to optimize the output result and control the output data format to improve the output stability of LLM.

3.4.1 Prompt engineering

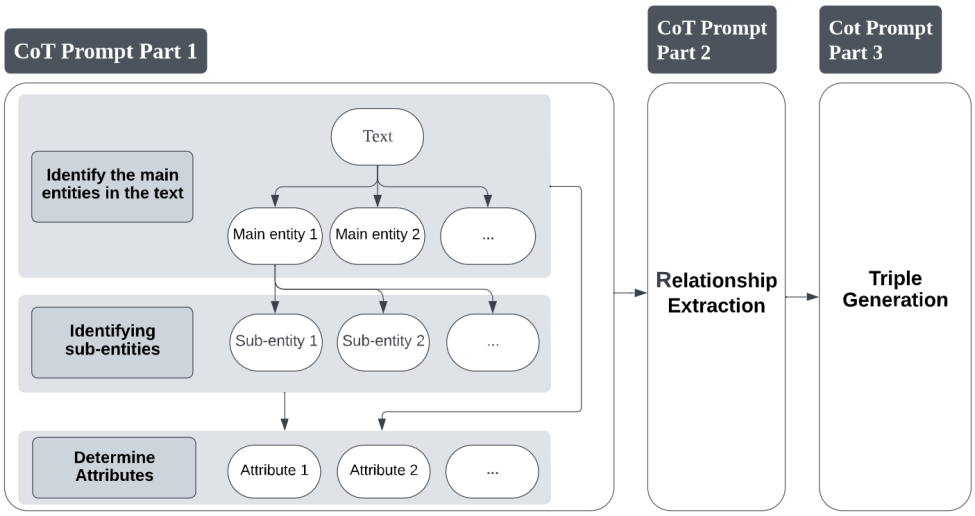

Figure 5: LLM CoT prompt engineering flow chart for entities and relationships extraction and triple generation.

In this work, we optimize the output of LLM by customizing the prompt. Therefore, the prompt we customized using the CoT strategy mainly consists of three parts: Entity Recognition, Relationship Extraction, and Triple Generation.

Entity recognition Based on the expectation that the shape of the knowledge graph is centered on the topic and extends outward layer by layer, we distinguish the entity types in the prompt into main entities, sub-entities, and attributes, and logically define these entities as subordinate relationships, in the expectation that LLM outputs a tree structure similar to part 1 of Figure 5. The following is an example of a prompt related to this part:

- Identify the main entities in the text. Main entities are the core of the 'topic', the sub-entities of the 'topic', and are usually the key items, concepts, or categories mentioned.

[5] `https://huggingface.co/MaziyarPanahi/Mistral-7B-Instruct-v0.3-GGUF`

- List all the different types of main entities, focusing on the specific characteristics that distinguish them.
- Identify Sub-Entities: Move on to identify entities that are secondarily related or subsidiary to each main entity. These could be components or related items, each characterized by unique aspects or roles tied to the 'topic'.
- Determine Attributes: For each entity identified list their attributes. Attributes are specific characteristics, features, or properties that describe an entity, such as measurements, colors, functionalities, or any descriptive detail provided in the document.

In this part of the prompt, the entity extraction is closely related to the "topic" of the cluster. This helps to reduce important computing resources and establish more meaningful relationships. Because a cluster can contain a large number of entities, these entities are not always closely related to the cluster topic, that is, there are some noisy entities. Part of the reason for the appearance of noisy entities is that when we extract topic-based clusters before, the minimum unit is the sentence, and some long sentences may contain both relevant and irrelevant content.

Relationship extraction As shown in part 2 of Figure 5, we define the relationship extraction object in prompt and provide some simple relationship examples:

- Define the relationships between entities and their attributes or between different entities.
- Use descriptive verbs to articulate these relationships, such as 'has', 'includes', 'equippedWith', 'isTypeOf', 'hascomponent' etc.
- Ensure that each main entity has relationship with the 'topic'.
- Ensure that each main entity does not share attributes, relationships, or characteristics with other main entities to maintain the clarity of its representation and specificity.

For the last two instructions, we set constraints based on the undesirable outputs in the experiment. For example, LLM may generate a main entity related to the topic in the original text, but no relationship connected to the topic is generated in the triple, resulting in a discrete knowledge graph, which is not what we expect. Another example is when multiple entities share an attribute, a list of multiple entities is generated and connected to the attribute, but we want each entity to be connected to the attribute separately because it is convenient for visualization.

Triple generation We define the specific content of generating triples in "CoT Prompt Part 3". Here we use "subject", "predicate", "object" to refer to triples instead of "entity" and "relation" because this indicates the direction of connection of each node in the triple, and in the context of this work, it is necessary to summarize both entities and attributes as nodes of triples. In addition, as in the previous section, we impose specific conditions based on the expected output of LLM.

- Generate triples using all the information obtained, following the requirement below to include only a subject (representing an entity), a predicate (representing a relationship), and an object (representing an entity or attribute).

- If there are two or more attributes that result in the same relationship, merge the same relationships and create a sub-triple that describes the unique points of the attributes.
- Make sure that all entities or attributes have a direct or indirect relationship with the 'topic'.

3.4.2 Output control

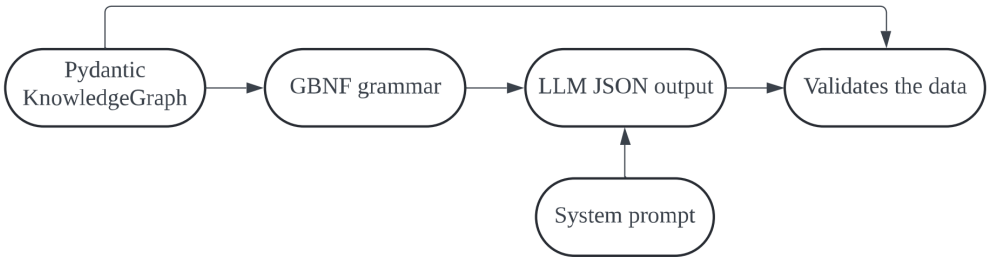

Figure 6: Structured output and verification flow chart

In this work, although the CoT strategy works well, LLM generates a thinking process in the output, which is not conducive to subsequent data processing. Therefore, we integrated the GBNF grammar, Pydantic, and system prompts in LLM to standardize the output format. The process is shown in Figure 6. We first use the BaseModel class in Pydantic to define the expected triple format, which consists of a subject, a predicate, and an object, all of which are string data types. Next, we define a KnowledgeGraph class as a collection of an indefinite number of triple lists, and set each instance of the model not to share lists.

We then generate GBNF grammar based on the 'KnowledgeGraph' class to guide LLM to generate JSON format output. In addition, we set the system prompt in the LLM response to "You are an advanced AI assistant responding in JSON format". Finally, the generated JSON format output is converted to the 'KnowledgeGraph' class format and a 'KnowledgeGraph' instance is established. At this time, Pydantic automatically validates the data according to the types and constraints specified in the BaseModel.

It is worth noting that we use GBNF grammar and system prompt to guide LLM to generate JSON format instead of letting LLM directly generate the 'KnowledgeGraph' class because the current LLM formatting output is limited.

Here is a simple partial output example as shown in Figure 7. The original text reads as follows "All 800mm wide x 800mm deep or 600mm wide x 1000mm (Server and PC cabinets) deep cabinets shall be provided with single door front and rear; cabinets shall be based on the Rittal TS8808 series. All cabinets shall be 2100mm tall, inclusive of a 100mm plinth." The output triples capture the main entities: "Cabinet", "door", "Rittal TS8808 series" and the two different sizes. The relationships between them are correctly extracted.

```
{
  "triples": [
    {
      "subject": "Cabinet",
      "predicate": "hasSize",
      "object": "800mm x 800mm x 2100mm"
    },
    {
      "subject": "Cabinet",
      "predicate": "hasSize",
      "object": "600mm x 1000mm x 2100mm"
    },
    {
      "subject": "Cabinet",
      "predicate": "hasDoor",
      "object": "Single"
    },
    {
      "subject": "Cabinet",
      "predicate": "isBasedOn",
      "object": "Rittal TS8808 series"
    },
```


Figure 7: Example of LLM output in JSON format

3.5 Knowledge graph visualization

The final knowledge graph is presented through an interactive visualization interface. Specialized libraries applied involve the usage of NetworkX[6] and PyVis[7] to create visualizations in web browsers, optimizing visual effects for knowledge graph formats: automatically scaling node size based on text volume, automatic line wrapping, assigning different colors to different topic nodes. An interactive graph layout with draggable nodes, zooming, and click highlighting is provided.

Here is a simple original document of Cabinet. I have highlighted the most relevant original words in the knowledge graph underlined:

Cabinets: All 800mm wide x 800mm deep or 600mm wide x 1000mm (Server and PC cabinets) deep cabinets shall be provided with single door front and rear; cabinets shall be based on the Rittal TS8808 series. All cabinets shall be 2100mm tall, inclusive of a 100mm plinth. The cabinets shall have a rating of IP41 as a minimum for all cabinets and be provided with fans and filters. All cabinets shall be provided with tag plates in accordance with project specifications. The design/layout of all cabinets shall follow the approved designs/layouts being implemented for the RRE and CBDC projects. All cabinets having power supplies shall have an RTD probe installed for temperature monitoring. Fan alarms shall be wired in series to a single DCS input. The VENDOR shall increase the DCS input count as required to include for all such inputs associated with cabinets with power supplies. Each cabinet shall have a fluorescent light installed and lit by the opening of the cabinet door. It shall be powered from the auxiliary power supply and not from the UPS power supply. The cabinet shall also contain a

[6] https://github.com/networkx/networkx

[7] https://github.com/WestHealth/pyvis

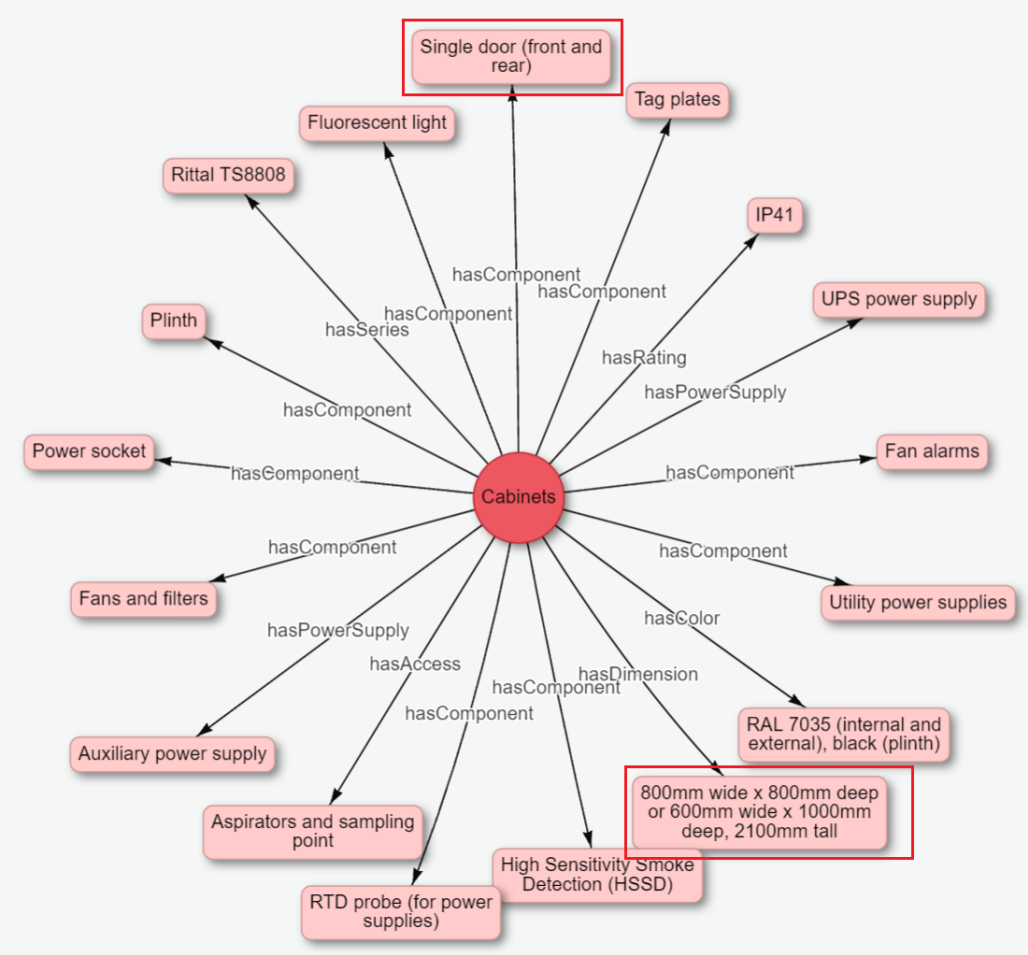

Figure 8: Example of a single cluster visualization knowledge graph with the topic 'Cabinet'.

power socket (of local type) that is not supplied via the UPS. Utility power supplies (local type socket), shall be provided at the rear right hand side of the cabinet. Internal and external colour of all cabinets shall be RAL 7035, the plinth shall be black. A High Sensitivity Smoke Detection (HSSD, supplied by Others) will be installed in all the cabinets. VENDOR shall provide access for the aspirators and sampling point.

The generated knowledge graph is shown in Figure 8. Cabinets (red node) represents the topic (keyword). The knowledge graph clearly displays the content under the current topic and indicates the subordinate relationship of knowledge through arrows. Not only that, the knowledge density of the knowledge graph is very high. In such a simple graph, a document of about 300 words can be represented. In addition, the generated content is appropriately summarized and modified. For example, the size description of the cabinet in the original text exists in these two sentences: 'All 800mm wide x 800mm deep or 600mm wide x 1000mm (Server and PC cabinets) deep cabinets shall be provided with single door front and rear; All cabinets shall be 2100mm tall'. The output merges the size data and omits the modifiers, giving a triple about size: ["Cabinets", "hasDimension", "800mm wide x 800mm deep or 600mm wide x 1000mm deep, 2100mm tall"]. LLM represents the sentence describing cabinet doors: 'cabinets shall

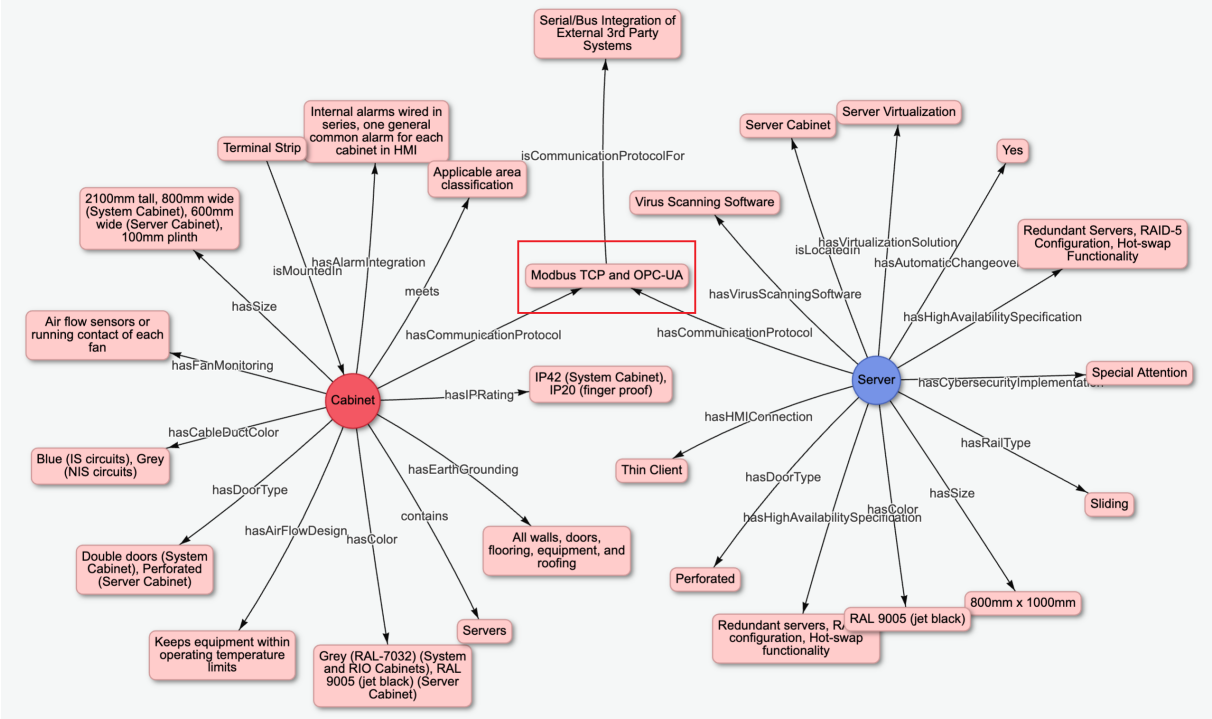

Figure 9: Example of a knowledge graph with two clusters visualized for the topics 'Cabinet' (red) and 'Sever' (blue).

be provided with single door front and rear.' as ["Cabinets", "hasComponent", "Single door (front and rear)"]. This rewrite highlights the key concepts and includes relevant details in parentheses. Such a clear structure is very conducive to the display of the knowledge graph, giving full play to the intelligence of LLM to simplify the expression, appropriately summarize the content, and do not affect the facts. The distribution of underlines in the original text is even, reflecting the breadth of the knowledge graph generated in this example.

As shown in the figure 9, this is an example of generating two clusters from a more complex document. The topics of the two clusters are Cabinet (red nodes) and Server (blue nodes). The knowledge graph well displays the content structure under each topic and clearly reveals the connection between two topics with the same communication protocol: 'Modbus TCP and OPC-UA' (red box).

4 Experiments and results

In this work, we used the L4 graphics card in the Colab platform for experiments. Since there is no comparable example of document-to-knowledge graph mapping, we divide the experiment into two parts. First, we compare the Keyword Clustering of this work with traditional clustering techniques. Then we measure the performance indicators of the generated knowledge graph. The parameters of the enhanced embedding are `context_window`$= 3$, $alpha = 0.5$. The parameter for automatically selecting the number of keywords is $cutoff = 0.85$.

4.1 Clustering experiment

For the Keyword Clustering method, we first use the TF-IDF method to extract keywords and then cluster based on keywords. We use nomic-embed-text-v1.5[8] as the SentenceTransformer model to transform sentences into vector embedding representations before clustering. The following are the specific processing steps:

- Get the word frequency and original word form, but keep the acronyms and all-capitalized words because these words are terms and do not need to be changed.
- Calculate the TF-IDF score.
- Multiply the word frequency by the TF-IDF score to generate a weighted score. Because we found that the TF-IDF score in a single document cannot well represent the ideal keyword, we added the word frequency as a weight.
- The percentile cutoff method determines the cutoff score.
- Filter and return keywords that meet the cutoff criteria.
- Finally, use the obtained keywords for clustering

In this experiment, we use K-means, Spectral, and HDBSCAN as baselines and conduct control experiments to measure the performance of Keyword Clustering.

For K-means and Spectral clustering methods, since they need to specify the number of clusters, we designed an adaptive cluster number algorithm: first determine the range of k ($1 < k \leq$ max_k) through the maximum number of clusters (`max_k`) parameter. For each K in the range of k, use the K-means model as a benchmark for 10 different initializations, calculate the silhouette score and select the number of clusters with the highest score. In the experiment, we set `max_k` and `n_init` (initialization number) of K-means and Spectral clustering methods to 10. For the HDBSCAN clustering method, there is no need to specify the number of clusters, so we set `min_cluster_size` $= 2$, `cluster_selection_epsilon` $= 0.1$.

We use four metrics, Silhouette Score, Davies-Bouldin Index, Calinski-Harabasz Index and Mean Intra-Cluster Cosine Similarity (MICCS), to evaluate different clustering methods. The evaluation process of MICCS is as follows:

- Compute the intra-cluster embedded cosine similarity matrix, which represents the similarity of each sentence to every other sentence in the cluster. And set the diagonal of the similarity matrix to zero to exclude the similarity of the sentence to itself.
- Add the intra-cluster cosine similarities and then divide by the number of valid sentence pairs to obtain the average cosine similarity of the cluster.
- Perform the above steps for all clusters (except the noise cluster in the Keyword Clustering method).
- Finally, calculate the average of the average cosine similarities of all clusters. This gives the average intra-cluster cosine similarity (MICCS).

[8] `https://huggingface.co/nomic-ai/nomic-embed-text-v1.5`

We prepared 6 real engineering documents as the test set for the experiment, namely: 'Instrument Cabinet', 'Electrical Training', 'Electrical Cabinets', 'BMS', 'MCC', and 'Testing'.

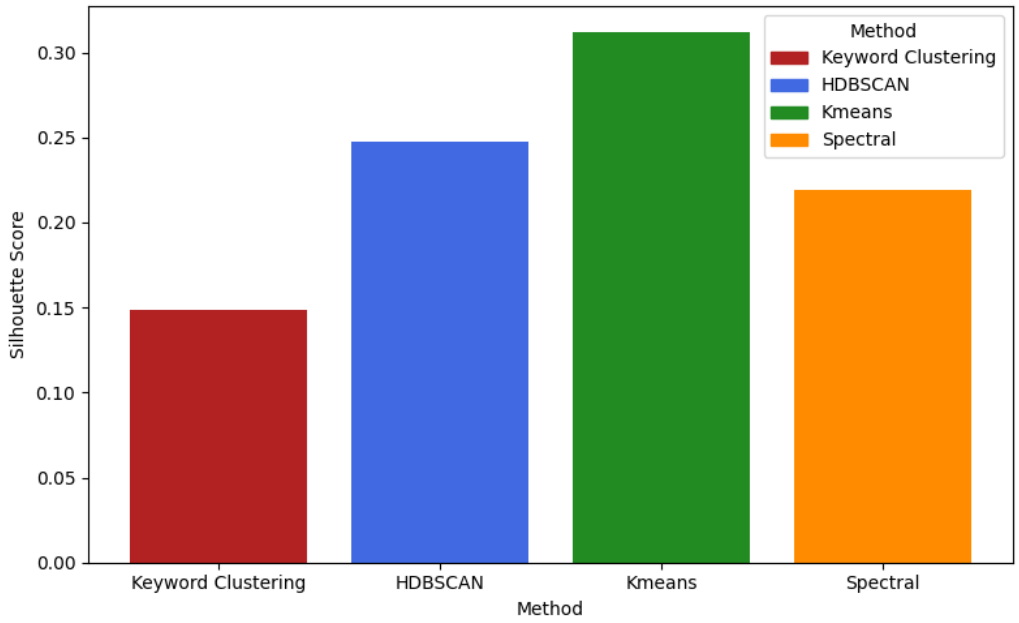

Figure 10: Histogram of the average Silhouette Score of Keyword Clustering (red), HDBSCAN (blue), K-means (green) and Spectral (orange) on the test set.

The average Silhouette Score of Keyword Clustering, HDBSCAN, K-means and Spectral on the test set is shown in Figure 10. K-means has the highest score, followed by HDBSCAN and then Spectral. The lowest score is Keyword Clustering. Since the Silhouette Score measures the degree of similarity of each point in a cluster to the points in other clusters compared to the points in its own cluster. The higher the value, the better the definition of the cluster. So Keyword Clustering has the worst performance of all methods.

The average Davies-Bouldin Index of Keyword Clustering, HDBSCAN, K-means and Spectral on the test set is shown in Figure 11. Keyword Clustering has the highest index, followed by HDBSCAN and then Spectral. K-means has the lowest index. Since the Davies-Bouldin Index measures the average similarity between each cluster and its most similar cluster, the lower the value, the better the clustering effect. Therefore, the performance ranking is opposite to the index ranking, and Keyword Clustering performs worse than the baseline methods.

The average Calinski-Harabasz Index of Keyword Clustering, HDBSCAN, K-means and Spectral on the test set is shown in Figure 12. Unlike the previous results, HDBSCAN has the lowest index, while the index of Keyword Clustering is slightly higher but lower than Spectral and K-means. Since the Calinski-Harabasz Index measures the ratio of the sum of the inter-cluster dispersion and the intra-cluster dispersion, the higher the value, the clearer the definition of the cluster. Therefore, although Keyword Clustering performs better than HDBSCAN, it is still worse than the other two baselines.
The number of sentences in each test document and the number of clusters of Keyword Clus-

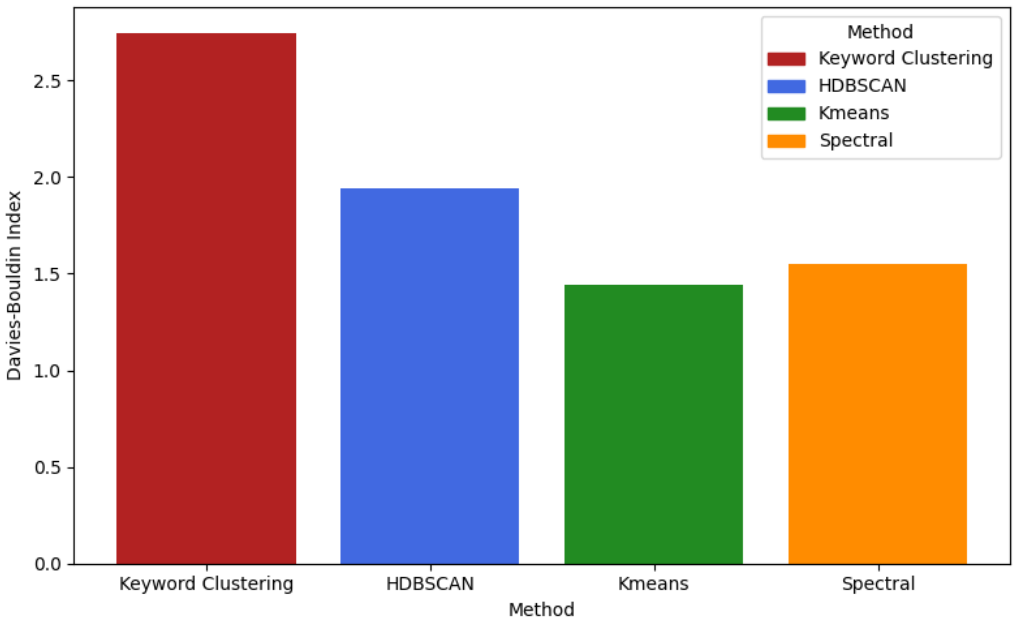

Figure 11: Histogram of the average Davies-Bouldin Index of Keyword Clustering (red), HDBSCAN (blue), K-means (green) and Spectral (orange) on the test set.

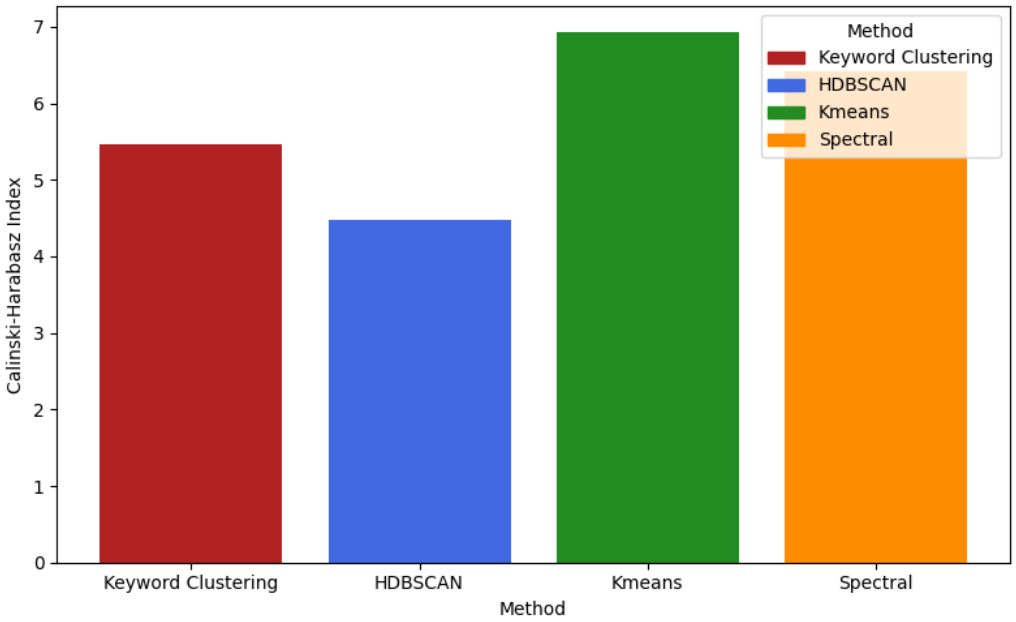

Figure 12: Histogram of the average Calinski-Harabasz Index of Keyword Clustering (red), HDBSCAN (blue), K-means (green) and Spectral (orange) on the test set.

tering, HDBSCAN, K-means and Spectral on the test set are shown in Table 1. We use the number of sentences in a document to represent the length of the document rather than the number of words, because the smallest unit of clustering is the sentence. This clearly shows the classification of each method under documents of different lengths: the number of clusters of Keyword Clustering is small in all documents, ranging from 2 to 4, while the number of

clusters of HDBSCAN, K-means and Spectral are generally large. Among them, the number of clusters of K-means and Spectral is 10 under 3 files, because the `max_k` parameter of the adaptive cluster number algorithm is 10. HDBSCAN has no maximum number of clusters, so the maximum number of clusters reaches 18. In addition, we found that the number of clusters of the baseline methods is also more easily affected by the length of the document. The number of sentences in the BMS document is 161, and the corresponding number of clusters of the baseline method is also the largest compared to other documents.

Table 1: The number of sentences in each document of the test set and the number of clusters of the four methods Keyword Clustering, HDBSCAN, K-means and Spectral on the test set.

Doc Name	**Total Sentences**	**Keyword Clustering**	**HDBSCAN**	**K-means**	**Spectral**
Instrument Cabinet	56	2	7	9	9
Electrical Training	24	2	4	10	10
Electrical Cabinets	41	2	7	10	9
BMS	161	3	18	10	8
MCC	34	3	4	9	10
Testing	52	4	8	8	10

Table 2: The number of sentences in each cluster under MCC document for four methods: Keyword Clustering, HDBSCAN, K-means, and Spectral.

Cluster Name	**Keyword Clustering**	**HDBSCAN**	**K-means**	**Spectral**
MCC	28	16	5	8
Maintenance	11	3	4	3
Wire	-	7	2	3
Test	-	-	4	3
Protection	-	-	3	-
Control	-	-	2	-
Time	-	-	-	2
Plant	-	-	-	2
Dust	-	-	-	3
MCC_1	-	8	7	4
MCC_2	-	-	4	-
Maintenance_1	-	-	3	4
Plant_1	-	-	-	2

In order to observe the results of different clustering methods in more detail, we will show all the clusters and the number of sentences in each cluster obtained by each method under a specific document. Due to the similar performance under each document and the complexity of the data, we selected the clustering details of the four methods under the document 'MCC' as shown in Table 2. Among them, Keyword Clustering has the least number of clusters, namely 'MCC' and 'maintenance'. It is worth noting that these two clusters also appear in all other clustering methods. The HDBSCAN method has slightly more clusters (4). The additional clusters are 'Wire' and 'MCC_1'. We believe that "Wire" is a technical detail in engineering documentation. Most of the time, it appears as a part of the machine. It is not general enough

to classify it as a cluster. The appearance of 'MCC_1' is because HDBSCAN recognizes a subcategory of 'MCC'. However, for this work, we expect the identification of subcategories to occur during the LLM processing period, because when all the text under a topic is used as input, it helps LLM better understand the context. For K-means and Spectral, they have more clusters, 9 and 10 respectively. The extra clusters are similar to the results of HDBSCAN, with more subcategories and details as clusters. From the number of sentences in the cluster, except for Keyword Clustering, the sentences of the other three methods are scattered. In addition, the sum of the total number of sentences of the three baseline methods is equal to 34, which is the total number of sentences in this document. However, the total number of sentences of the Keyword Clustering method is greater than 34, because some sentences are repeatedly assigned to different clusters. This is conducive to handling complex sentences containing multi-topic content.

After manual review of the remaining five documents by domain experts, the performance data of the four methods are similar to the above, that is, the Keyword Clustering method performs best. Its number of clusters is relatively stable under similar text lengths, and the cluster topics are highly summarized and truly represent the topics described in the text.

4.2 Knowledge graph experiment

The LLM parameters were set to: $temperature = 0.0$, `max_tokens` $= 40000$, $seed = 0$. The steps are as follows:

1. Convert sentences to embeddings: In this experiment, we use the pre-trained sentence conversion model: 'all-MiniLM-L6-v2[9]' to convert sentences into vector representations (embeddings). It is worth noting that we use two sets of sentences: the sentences under the original text clustering and the triples of the generated knowledge graph rewritten into normal format sentences, such as "subject": "Server", "predicate": "isLocatedIn", "object": "Server Cabinet" rewritten as "Server is located in Server Cabinet."

2. Calculate sentence embeddings: Use the model to calculate the sentence embeddings for the two sets of sentences respectively, and get the vector representation of the sentence.

3. Calculate similarity scores: The embedding similarity score of each triple sentence with all original text sentences is calculated using cosine similarity.

4. Evaluate matching: For the condition of identifying similar sentences, we did not simply set a similarity threshold to distinguish similarity from dissimilarity, because it is difficult to find a universal threshold among many similar sentences. Instead, we distinguish whether similarity is based on the difference between the average similarity of the top-n $(1 \leq n \leq 3)$ original sentences and the average similarity of the remaining original sentences, where n is an integer. Because if the generated triples are based on a certain original sentence, the similarity score between them will be much greater than the remaining original sentences. At the same time, we considered the top-n case, because the triples may be generated based on multiple sentences.

[9] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

Therefore, if the average similarity of any number of top-n original sentences of a triple sentence differs from the average similarity of the remaining original sentences by more than 0.3, they are considered to be matched successfully. If the triple sentence finds a match, it is counted as True Positive, otherwise it is counted as False Positive. Finally, count how many original sentences are not matched by any triple sentence, that is, False Negative.

Use the following formula to calculate the evaluation metrics

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{17}$$

Precision is shown in Formula 17, which calculates the proportion of true positive examples among all the results judged as positive examples.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{18}$$

Recall is shown in Formula 18, which calculates the proportion of all true positive examples that are correctly judged as positive examples.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{19}$$

F1 Score is shown in Formula 19, which is the harmonic mean of precision and recall, used to balance the relationship between the two.

Table 3: Precision, recall, and F1 score tables for a total of 380 knowledge graph triples generated under 6 different files.

File Name	**Precision**	**Recall**	**F1 Score**
Instrument Cabinet	0.935	0.795	0.859
Electrical Training	0.688	0.825	0.750
Electrical Cabinets	0.861	0.756	0.805
BMS	0.940	0.462	0.619
MCC	0.656	0.618	0.636
Testing	0.982	0.806	0.885
Average	0.844	0.710	0.759

As shown in Table 3, we evaluated the performance of generating knowledge graphs under 6 test documents (349 triples in total). The average precision of 0.844 shows that the model under the pipeline of this work is good at generating correct triplets with few false matches, which is very important in the case of high false positive costs in engineering fields. The average recall is 0.710, which means that although the precision is high, the recall is relatively low, which indicates that the model may miss some relevant information that should be matched due to complex sentence structure or ambiguous language.The average F1 score of 0.759 reflects the strong overall performance. However, it is worth noting that for the BMS document, the generated triplets have low recall (0.462) but high precision (0.940). For the Electrical Training document, the generated triplets have low precision (0.688) but high recall (0.825). For the MCC document, both precision (0.656) and recall (0.618) are low. Given that the

average values of precision, recall, and F1 score are relatively high, we believe that the model is generally effective in extracting and representing information as triplets.

5 Discussion

The results of Keyword Clustering, and the other three baseline clustering methods are basically the same in the four metrics of Silhouette Score, Davies-Bouldin Index, Calinski-Harabasz Index and Mean Intra-Cluster Cosine Similarity: Keyword Clustering is weaker than the baseline methods. However, after in-depth comparison of the number of clusters and cluster content of different methods, we found that the performance of Keyword Clustering is more in line with our work context. Because we expect to use these clusters to generate knowledge graphs with tight structure and correct grasp of the subject scope. The keyword clustering method identifies real text topics well and is not easily affected by document length to increase or decrease the number of clusters. Generally speaking, there are not many topics in a single engineering document. They are limited by the title of the document, even for documents with a length of hundreds of pages. Taking all factors into consideration, we believe that the Keyword Clustering method is better than the traditional clustering baseline methods, and is especially suitable for our project in the context of document mapping knowledge graphs.

On the other hand, we believe that fine-grained clustering methods may have a negative impact on clustering performance. From the perspective of the entire document, multiple clusters are more likely to make topics (keywords) redundant. This also explains why the baseline methods perform better under the four metrics, because more clusters contain less content, and it is easier to achieve clear boundaries and higher similarity within the cluster. Therefore, we tend to use the centroid similarity within the cluster for clustering, so that it is more universal for clustering sentences. Such a coarse-grained similarity matching method offsets the negative effects of complex sentences, complex concepts, and long documents to a certain extent.

In the evaluation experiment of generating knowledge graphs, we found a trend. The number of sentences covered by the knowledge graph in long documents is also decreasing, resulting in a smaller recall value for long documents. This explains why the knowledge graph has high precision but low recall under BMS documents. In addition to the limitation of the model's own capabilities, this trend also shows the essence of structured mapping documents, that is, summarization. However, the quality of the summary is not involved in this work, which can be an optimization direction for future work. After analysis, we believe that the abnormal scores of the Electrical Training document and the MCC document are due to the defects of our evaluation method. The two documents have opposite characteristics: the Electrical Training document contains a large number of long and complex sentences. The MCC document contains a large number of short sentences. This makes it difficult for the simple triples in the Electrical Training document that contain correct information to match the complex long sentences with similarity, resulting in low precision. However, the original document sentences are easily matched because they contain rich information, leading to high recall. The short sentences of MCC documents make it easy to reduce the similarity of the generated triples if they are rewritten, which results in most unmatched sentences because they happen to be slightly below the threshold. This explains why the results are low precision and low recall.

Limitations. In this work, the quality of the knowledge graph generated by LLM is not only affected by the prompt, but also by the quality of the input. Although our current processing flow has a text cleaning step, it still has shortcomings. For example, it cannot handle the gaps between PDF document pages well, and cannot perfectly handle sentence boundaries: when bullet points appear in the text, all points will be saved as a sentence, and the text extracted from the PDF document may appear garbled. These shortcomings sometimes have a great negative impact on the LLM output.

When processing long documents, even though the text has been segmented, the content of each cluster is still huge. Affected by the context window of LLM, the connection between knowledge becomes less, thus weakening the structure of the knowledge graph.

As described in the discussion section, the essence of document mapping is to summarize to a certain extent. This will inevitably reduce the display of details, so how to balance details and summarization becomes a problem. At the same time, there is also a lack of means to fine-tune the output details, because using prompts to guide LLM is similar to black box operations. The current work does not consider this aspect.

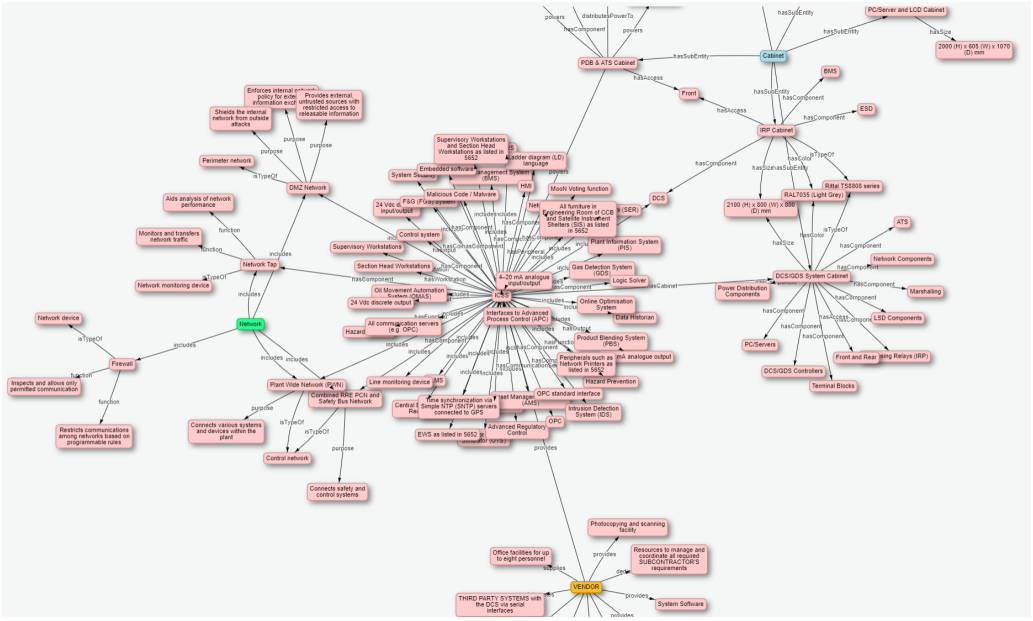

Figure 13: Example of a complex knowledge graph

As shown in Figure 13, for a visualized knowledge graph, if the content of the obtained knowledge graph is too rich, it is not conducive to visualization, and engineers can only understand the document through inefficient viewing methods such as zooming in and out, which is contrary to the vision of this work.

6 Conclusion

This work proposes the following solutions to the problems encountered when using LLM to help people review and check documents:

- In order to solve the challenge of long text processing, we use the Keyword Clustering method to segment text. That is, first use the TF-IDF method to find the keywords (topics) of the document. Then use the centroid vector of the initial cluster that exactly matches the keyword and the embedding vector of the remaining sentences to divide them into different clusters according to similarity.

- To reduce costs and ensure data security, we use a local open source model. Finally, to improve the efficiency of model inference, we finally chose the quantized version model of Mistral-7B-Instruct-v0.3.

- For complex knowledge graph task expectations, we apply the CoT prompt strategy to effectively guide LLM to generate ideal output.

- By providing the LLM with low-noise clustered inputs and using prompt under the CoT strategy to guide the LLM to generate results from the provided text, we reduce LLM hallucinations and enhance the interpretability of the outputs.

- For the problem of unstable LLM output format, we use GBNF grammar, Pydantic, and the system prompt in LLM to standardize and check the output format, avoiding the problem that the LLM output conflicts with the data format expected by subsequent processing.

The results show that our keyword clustering method can effectively transform complex engineering documents into structured visual data representations. The main advantages include: keyword identification that matches the main purpose of the document, efficient sentence clustering method, effective combination of LLM and prompting engineering to handle domain-specific content, and use of data validation technology to ensure the generation of formatted data.

Future Work Develop a stable and efficient text cleaning algorithm that can effectively filter text noise. Try to find a more powerful and efficient model to carry out text mapping knowledge graph work. Improve knowledge graph evaluation methods to reduce misjudgments. Try to find a reasonable way to balance the display of details and the summary of information. Develop a visual interaction method, such as displaying a highly summarized structure by default for dense knowledge graphs, and then expanding the detailed nodes when the user clicks. Try to further utilize the generated knowledge graph, such as document semantic matching, document difference analysis, graph analytics etc.

References

1. Campello, R. J., Moulavi, D. & Sander, J. *Density-based clustering based on hierarchical density estimates* in *Pacific-Asia conference on knowledge discovery and data mining* (2013), 160–172.
2. Devlin, J. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
3. Frantar, E., Ashkboos, S., Hoefler, T. & Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).
4. Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B. & Heming, J. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences* **622,** 178–210 (2023).
5. Ji, Z. *et al.* Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* **55.** ISSN: 0360-0300. `https://doi.org/10.1145/3571730` (Mar. 2023).
6. Jiang, A. Q. *et al.* Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).
7. Khan, A. *et al.* Sentence embedding based semantic clustering approach for discussion thread summarization. *Complexity* **2020,** 4750871 (2020).
8. Lewis, P. *et al.* Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* **33,** 9459–9474 (2020).
9. Lin, J. *et al.* AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of Machine Learning and Systems* **6,** 87–100 (2024).
10. Mahapatra, J. & Garain, U. Impact of Model Size on Fine-tuned LLM Performance in Data-to-Text Generation: A State-of-the-Art Investigation. *arXiv preprint arXiv:2407.14088* (2024).
11. Manning, C. D., Raghavan, P. & Schütze, H. *Introduction to Information Retrieval* ISBN: 0521865719 (Cambridge University Press, USA, 2008).
12. Maulik, U. & Bandyopadhyay, S. Performance evaluation of some clustering algorithms and validity indices. *IEEE Transactions on pattern analysis and machine intelligence* **24,** 1650–1654 (2002).
13. Merris, R. Laplacian matrices of graphs: a survey. *Linear algebra and its applications* **197,** 143–176 (1994).
14. Mughnyanti, M., Efendi, S. & Zarlis, M. *Analysis of determining centroid clustering x-means algorithm with davies-bouldin index evaluation* in *IOP Conference Series: Materials Science and Engineering* **725** (2020), 012128.
15. Naveed, H. *et al.* A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435* (2023).
16. Peng, C., Xia, F., Naseriparsa, M. & Osborne, F. Knowledge graphs: Opportunities and challenges. *Artificial Intelligence Review* **56,** 13071–13102 (2023).
17. Portmann, E., Kaltenrieder, P. & Pedrycz, W. Knowledge representation through graphs. *Procedia Computer Science* **62,** 245–248 (2015).

18. Ramos, J. *et al. Using tf-idf to determine word relevance in document queries* in *Proceedings of the first instructional conference on machine learning* **242** (2003), 29–48.

19. Shahapure, K. R. & Nicholas, C. *Cluster quality analysis using silhouette score* in *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)* (2020), 747–748.

20. Steinwart, I., Hush, D. & Scovel, C. An explicit description of the reproducing kernel Hilbert spaces of Gaussian RBF kernels. *IEEE Transactions on Information Theory* **52,** 4635–4643 (2006).

21. Trajanoska, M., Stojanov, R. & Trajanov, D. Enhancing knowledge graph construction using large language models. *arXiv preprint arXiv:2305.04676* (2023).

22. Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing* **17,** 395–416 (2007).

23. Wei, J. *et al.* Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* **35,** 24824–24837 (2022).

24. Zhang, S. *et al.* Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792* (2023).