# Prompt Engineering for Developers

**Punith Kumar D**
(PES UNIVERSITY, Bengaluru, Karanataka
punithkumard8@gmail.com)

**Sunil Sahu**
(PES UNIVERSITY, Bengaluru, Karanataka
sunil.sahu6522@gmail.com)

**M S Mohan Kumar**
(PES UNIVERSITY, Bengaluru, Karanataka
msmohan.kumar2@gmail.com)

**Aswitha P**
(PES UNIVERSITY, Bengaluru, Karanataka
aswitha.pothumudi@gmail.com)

February 8, 2025

**Abstract**

Prompt engineering has emerged as a crucial discipline in the domain of large language models (LLMs), enabling developers to optimize and tailor generative models for specific tasks. This paper explores methodologies, tools, and applications for prompt engineering, leveraging technologies such as LangChain, vector databases, and semantic search. Additionally, we analyze strategies for aligning prompts with LLM capabilities and evaluate their efficacy using test cases. This work consolidates insights from contemporary research to offer a comprehensive guide for developers.

# Contents

# 1 Introduction

Prompt engineering involves crafting input queries to optimize outputs from LLMs. With the proliferation of generative AI, developers face challenges in effectively utilizing these models for diverse domains. This paper addresses these challenges, presenting state-of-the-art techniques and tools. Motivated by the potential of LLMs to revolutionize industries, we explore the intersection of prompt engineering and developer workflows.

## 1.1 Motivation

Traditional methods for interacting with AI models often lead to suboptimal results. By refining prompts, developers can:

- Improve model accuracy.
- Enhance contextual relevance.
- Optimize resource utilization.

## 1.2 Scope

This paper focuses on methodologies such as LangChain integration, semantic search, vector database utilization, and test case design to refine prompt engineering.

# 2 Literature Survey

## 2.1 LangChain for Prompt Engineering

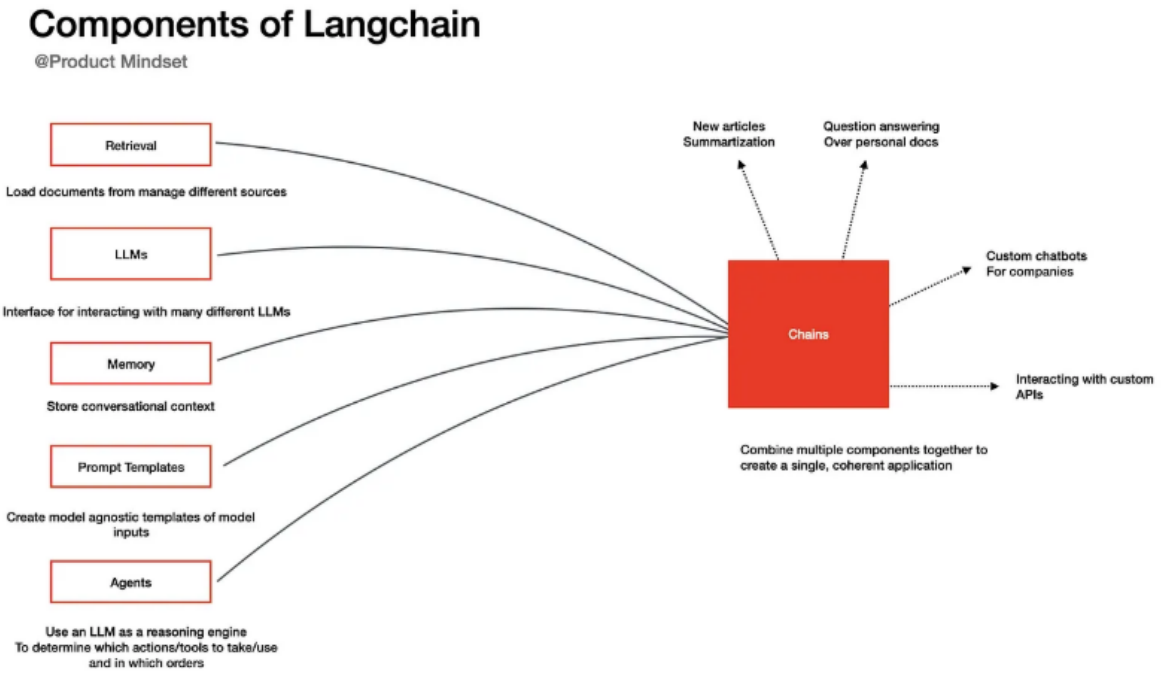
LangChain is a framework for building applications with LLMs. It simplifies integration with external databases and enhances model performance by chaining prompts and responses.

### 2.1.1 Importance of LangChain and OpenSource LLM in Prompt Engineering

LangChain is a key tool for developers working with LLMs because it simplifies integrating LLMs with external data sources and APIs, allowing the creation of more robust, customized applications. OpenSource LLMs, on the other hand, provide flexibility, customization, and transparency, making them ideal for developers looking to tailor models to specific needs. By combining LangChain with OpenSource LLMs, developers can achieve highly effective results in prompt engineering, fine-tuning models, and building interactive applications.

### 2.1.2 How LangChain manages prompt chaining

LangChain provides a powerful framework for building modular workflows in chatbot applications. By combining structured prompts, dynamic chaining, and advanced LLM integration, it allows developers to create scalable, adaptive pipelines that leverage RAG techniques and deliver structured outputs like JSON. Here's how LangChain handles prompt chaining effectively: beginfigure[h!]




LangChain's architecture

**1. Prompt Abstraction** LangChain utilizes the `from_template` function to design structured input-output workflows for each step. This approach simplifies the management of complex chatbot operations by enabling clear and consistent interactions.

**2. LLM Integration** The framework seamlessly integrates with various LLMs, such as IBM Granite, OpenAI, and Hugging Face, enabling fine-tuning for customized tasks.

**3. Chain Management** LangChain's `SequentialChain` and `SimpleSequentialChain` enable modular workflows for chatbot pipelines, while `stroutputparser` ensures structured outputs such as JSON.

**4. Dynamic Workflows** Using tools such as `ConditionalChain` and system message templates, LangChain supports adaptive workflows, aligning with the principles of RAG (retrieval-augmented generation) for dynamic content generation.

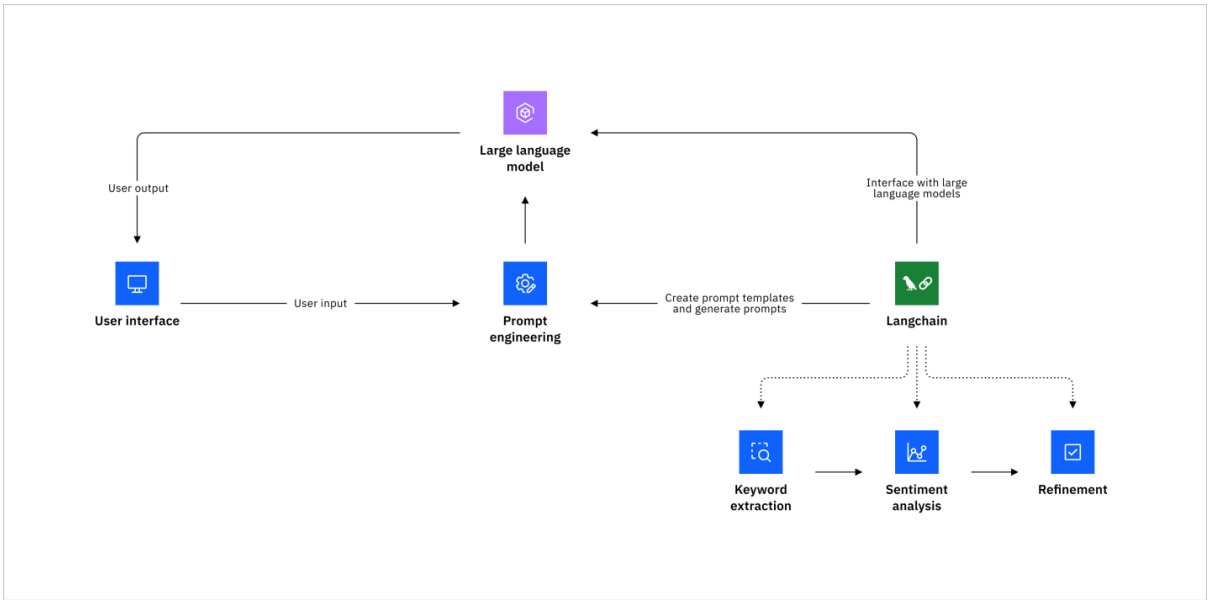



Figure 1: Dynamic Workflows

### 2.1.3 Approach

In LangChain, a "chain" is a sequence of operations where the output of one step becomes the input for the next. This helps in building complex workflows by linking

together multiple functions.

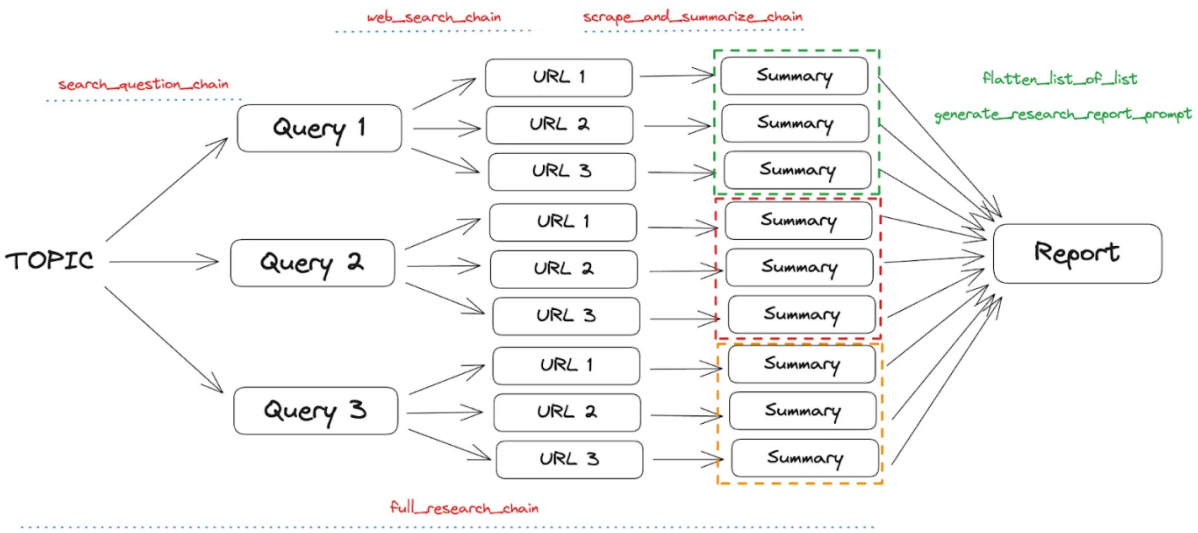



Figure 2: Approach

### 2.1.4 Use Cases of LangChain and OpenSource LLMs in Prompt Engineering

LangChain, in combination with OpenSource LLMs, can be applied to a range of tasks, providing innovative solutions to various challenges:

#### 1. Semantic Search and Retrieval

- **Description:** LangChain can be used to enhance semantic search capabilities by integrating LLMs with vector databases like FAISS or Milvus. This enables the retrieval of contextually relevant information based on the semantic meaning of user queries.
- **Example:** Building an AI assistant that retrieves relevant documents or answers based on the user's query, where the content is indexed and stored in vector databases.

#### 2. Automated Content Generation

- **Description:** Developers can use LangChain with OpenSource LLMs to automate the generation of personalized content, whether for chatbots, marketing materials, or reports.

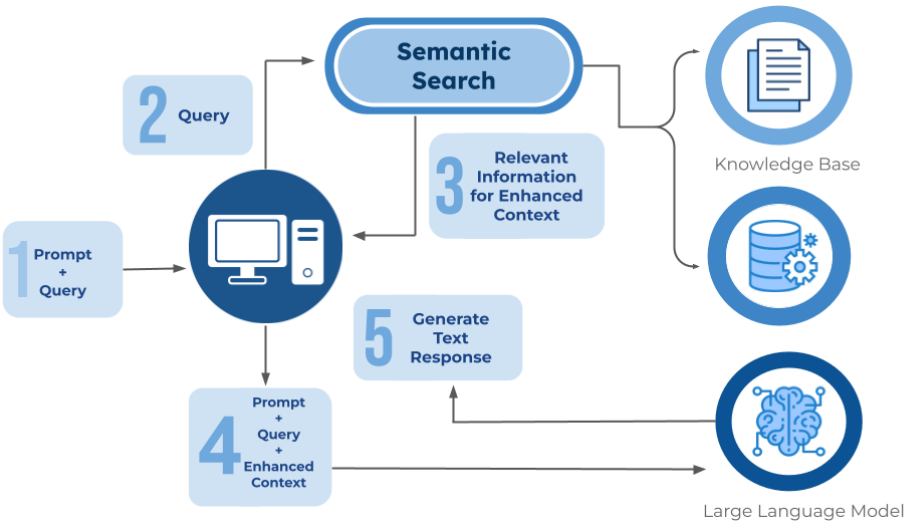



Figure 3: Semantic Search and Retrieval

- **Example:** A content generation tool that creates customized emails or blog posts based on a given set of keywords or user preferences.

### 3. Question-Answering Systems

- **Description:** LangChain can be used to build advanced question-answering systems that provide accurate responses by retrieving context from various data sources.
- **Example:** A customer service AI that answers inquiries by analyzing historical data, product manuals, and FAQs.

### 4. Multi-Tool Integration

- **Description:** LangChain allows the integration of LLMs with multiple external tools, such as APIs, databases, and third-party software, to extend the model's functionality.
- **Example:** An AI assistant that can interact with both a knowledge base and a scheduling application, responding to user queries with relevant answers and scheduling actions.

### 2.1.5 Advantages and Disadvantages of Using LangChain with OpenSource LLMs

#### Advantages

- **Customizability:** OpenSource LLMs provide developers with full control over model behavior, enabling tailored solutions for specific tasks.
- **Cost-Effectiveness:** OpenSource LLMs can be used without expensive licensing fees, making them more affordable for developers and businesses.
- **Scalability:** LangChain helps build scalable systems that can handle a large number of tasks simultaneously, especially when integrated with external tools.
- **Flexibility:** LangChain's modularity allows developers to select and integrate only the components they need, ensuring that they can build lightweight and efficient systems.
- **Integration with External Tools:** LangChain simplifies integrating external data sources, APIs, and databases, making it possible to build sophisticated systems.

#### Disadvantages

- **Complexity:** While LangChain offers great flexibility, it may require a steep learning curve for developers unfamiliar with integrating external tools and handling LLMs.
- **Resource Intensive:** Using OpenSource LLMs, especially large models, can require significant computational resources, including powerful GPUs.
- **Limited Community Support:** As some OpenSource LLMs are still emerging, they may lack robust community support or have limited documentation.
- **Latency:** The response time for complex systems may increase due to multiple integrations or processing steps involved in the pipeline.

### 2.1.6 Process and Methods for Effective Prompt Engineering Using LangChain and OpenSource LLMs

**1. Input Preprocessing and Data Ingestion** The first step in prompt engineering is preparing the input data. This involves processing raw inputs into formats suitable for LLMs to generate meaningful outputs. LangChain supports input preprocessing, allowing seamless data ingestion from multiple sources, such as APIs, databases, or file systems.

**2. Prompt Crafting and Optimization** LangChain simplifies the process of crafting and optimizing prompts by chaining different actions and steps. The framework can refine the prompts based on specific tasks and data sources, ensuring that the LLM generates responses that are accurate and contextually relevant.

**3. Model Integration and Fine-Tuning** LangChain allows developers to integrate OpenSource LLMs and fine-tune them to improve task-specific performance. Developers can adjust hyperparameters or add training data to enhance the LLM's capabilities.

### 2.1.7 4. Output Generation and Evaluation

After generating outputs, the next step is evaluating the results to ensure that they meet the desired goals. This evaluation can be done using various metrics like relevance, accuracy, and user satisfaction. LangChain provides tools for analyzing and improving output quality based on these metrics.

### 2.1.8 Handling LangChain with OpenSource LLMs

**1. Managing Multiple Integrations** LangChain excels at managing multiple integrations, enabling developers to handle complex workflows. Whether dealing with external databases or APIs, LangChain provides an easy way to manage dependencies and ensure smooth execution of tasks.

**2. Error Handling and Debugging** When working with LangChain and OpenSource LLMs, developers need efficient error handling mechanisms. LangChain includes built-in features for logging and debugging, which helps in identifying issues during development and ensuring that the system runs smoothly.

**3. Optimizing Computational Resources** To minimize latency and resource consumption, LangChain can be used to optimize model calls and responses. Techniques such as batch processing and caching can be implemented to improve performance without requiring additional hardware.

### 2.1.9 Expected Outputs and Results

When using LangChain with OpenSource LLMs, the following outputs and results can be expected:

- **Contextual Responses:** The LLM generates contextually appropriate outputs based on the input queries and available data sources.
- **Improved User Experience:** By integrating multiple tools and optimizing prompts, the resulting system will provide faster, more accurate responses.
- **Scalable Solutions:** The system can scale to handle larger data sets and more complex queries.
- **Actionable Insights:** The model can provide actionable insights that guide decision-making and inform further development.

### 2.1.10 Conclusion

LangChain, in combination with OpenSource LLMs, represents a powerful tool for developers focused on prompt engineering. By simplifying the process of integrating external tools and data sources, LangChain makes it easier to build complex, scalable applications that leverage the power of LLMs. The flexibility, customization, and cost-effectiveness of OpenSource LLMs further enhance the capabilities of LangChain, making it an invaluable resource for developers looking to optimize prompt engineering and build innovative AI-driven systems.

**References**

1. LangChain Documentation
2. OpenAI, "GPT-3 and Beyond: Language Models for the Future," 2023.
3. Milvus, "FAISS: A Library for Efficient Similarity Search," 2023.
4. LangChain, "Framework for Building LLM Applications," 2024.

## 2.2 Efficient Prompting for LLM-Based Generative IoT (GIoT) Systems

The integration of Artificial Intelligence (AI) [3] with the Internet of Things (IoT), termed AIoT, has been widely adopted across various domains such as healthcare, smart cities, and industrial automation. Traditional AIoT systems rely on task-specific machine learning models, often deployed on cloud or edge servers. However, with the advent of Generative AI (GAI) and Large Language Models (LLMs), a new paradigm called Generative IoT (GIoT) is emerging, offering enhanced adaptability and intelligence for IoT applications.

Despite their advantages, LLM-based GIoT systems face significant challenges, including high computational demands, data privacy concerns, and scalability issues. This study proposes an efficient LLM-based GIoT system that addresses these challenges by deploying open-source LLMs on edge servers in a local network. The system includes a Prompt Management Module and a Postprocessing Module, which enhance adaptability and performance for various IoT tasks through tailored prompting techniques.

To demonstrate the effectiveness of the proposed system, we implement a semi-structured Table Question Answering (Table-QA) service, a complex yet valuable task for analyzing tabular data. We introduce a three-stage prompting method (task-planning, task-conducting, and task-correction) that improves reasoning accuracy and reduces inference costs. Our experiments on WikiTableQA and TabFact datasets validate the approach, achieving state-of-the-art performance compared to baseline methods.

## 2.3 LLM-Based Test Case Generation for Quality Assurance (QA)

Software testing is an essential phase of software development, often accounting for 30-50% of total development costs. The growing complexity of modern systems and the rise of agile and DevOps practices have heightened the need for automated testing([7]). Large Language Models (LLMs) with coding capabilities have emerged as potential tools for assisting in tasks like code generation, bug fixing, and test case creation. However, their application in Quality Assurance (QA)([8]) remains underexplored. Key challenges in LLM-based test case generation include:

- **Incompleteness**: Missing critical scenarios leading to inadequate testing.
- **Inconsistency**: Variability in style, quantity, and coverage.
- **Naivety**: Misinterpretation of domain-specific requirements.
- **Security risks**: Potential exposure of confidential corporate data in cloud-hosted LLMs.

To address these concerns, this study evaluates different open-source LLMs for test case generation and proposes an on-premises([10]) LLM solution based on the LLaMA 70B family of models. The research aims to improve test case quality through optimized prompting strategies, alignment techniques, and model architecture selection, ultimately advancing LLM-powered QA automation.

## 2.4 Vector Databases and Semantic Search

### 2.4.1 Introduction

The realm of Artificial Intelligence (AI) has seen advancements thanks to the development of Large Language Models (LLMs), like GPT (Generative Pre-trained Transformer), changing the way we handle data, understand information, and create natural language. These models, well-trained on datasets, possess the ability to generate

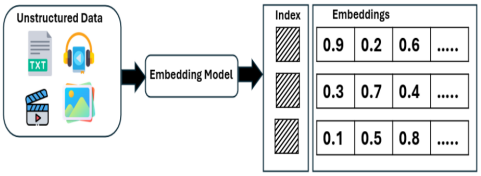



Figure 4: Vector DB Embeddings

coherent and contextually relevant text. This capability allows them to be applied in areas such as automated content creation and sophisticated conversational agents.([4]) However, despite their capabilities, LLMs come with limitations that involve issues related to timeliness, accuracy, and efficient data extraction from datasets. These challenges underscore the importance of strategies to enhance the utility of LLMs, especially in domains where precise and up-to-date information is crucial.

One effective strategy for addressing these limitations involves integrating Vector Databases (Vector DB) with LLMs. Vector DBs are systems for storing and retrieving data efficiently. They offer an approach for organizing, searching, and managing datasets using embeddings. Embeddings serve as vector representations of data that improve the information retrieval processes of LLM operations by providing a search capability that is rich in meaning.

This research aims to explore the use of Vector Databases in Large Language Model applications as a solution to address their limitations. We first examine the constraints of Large Language Models, identifying obstacles that hinder their widespread adoption and effectiveness. Then, we explore how Vector Databases can improve Large Language Models by focusing on the fundamentals and practical methods for integrating them. Our objective is to find ways to combine Language Models with Vector Databases to demonstrate how this integration can enhance the capabilities of these models and provide a roadmap for AI research. By merging these technologies, we strive to make progress in advancing the development of more efficient and adaptable systems.

### 2.4.2 Limitations of Large Language Models in Generative AI Use Cases

Large Language Models (LLMs) face limitations that can impede their practical application, especially in scenarios requiring high precision, timeliness, and specificity. While LLMs are at the forefront of the Generative AI revolution, they may not be well-suited for tasks with high demands. These limitations primarily arise from how the models are designed, trained, and operated.

1. **Hallucinations:**

   One significant issue with LLMs is known as hallucinations or misinterpretations, where the model tends to generate information that sounds plausible but is factually incorrect or nonsensical. This challenge stems from the models relying on patterns in their training data rather than verifiable facts. Consequently, this can be problematic in applications where accuracy is crucial.

2. **Stagnant Data Pool:**

   Large Language Models (LLMs) are trained on a dataset that becomes outdated over time, limiting their effectiveness in situations requiring up-to-date information. The fixed nature of their training data makes it challenging for them to adapt to information post-training, thereby reducing their usefulness for tasks involving current events or evolving knowledge domains.

3. **Challenges with Local Data Integration:**

   LLMs sometimes encounter difficulties incorporating or utilizing domain-specific or personalized data due to their generalized training approach. Without instruction on how to use this information, which may not always be feasible or realistic, large language models struggle to seamlessly integrate it. This limitation hinders their effectiveness in niche fields or tailored uses where integrating datasets could significantly enhance performance and applicability.

### 2.4.3 Strategies for Overcoming LLM Limitations with Vector DB Using Retrieval-Augmented Generation

To expand the capabilities of Large Language Models (LLMs) beyond their training data, it is crucial to adopt strategies that enhance their flexibility, precision, and awareness of various contexts. One effective approach to enriching the knowledge base of LLMs involves leveraging Retrieval-Augmented Generation (RAG) with Vector Databases (Vector DB).

Integrating RAG into the process can enhance information accuracy by including a retrieval step that accesses both external databases for relevant data before content generation. This practice helps minimize errors and inaccuracies in generated text. Vector DB, known for its retrieval of dimensional data, serves as a strong foundation for this method, enabling LLMs to access the most recent and validated information. By doing this, the technique significantly improves the accuracy and reliability of output content, ensuring coherence and factual correctness.

Incorporating Vector DB in the process empowers Language Models to tap into a database continually updated with new information, surpassing the constraints imposed by static datasets. This integration allows models to produce content that reflects the knowledge and facts accurately, overcoming limitations associated with fixed training datasets. Accessing up-to-date information in real time ensures that the results produced remain relevant and accurate in evolving fields.

By utilizing Vector DB, it becomes easier to incorporate domain-specific data into the process. Large Language Models can enhance their output by integrating datasets into a vector database, enabling them to retrieve and tailor the data to specific situations or needs. This functionality greatly enhances the adaptability and practicality of language models across fields, from offering personalized recommendations to providing technical support in specific domains. In summary, combining Retrieval-Augmented Generation with Vector Databases provides a solution for overcoming the limitations of Large Language Models. This approach not only boosts the capabilities of Large Language Models (LLMs) but also opens up new possibilities for their application, ensuring they

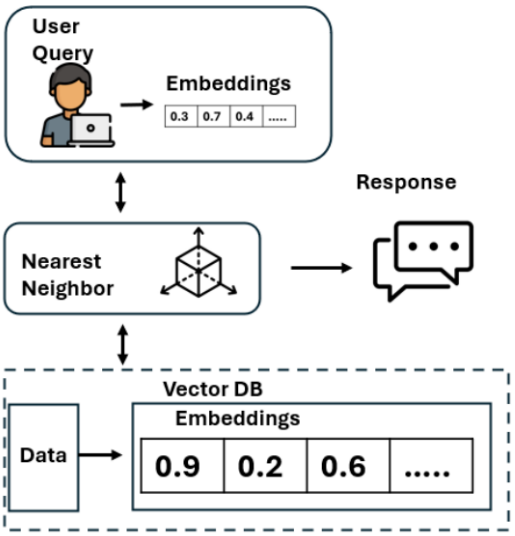



Figure 5: RAG with Vector DB

can meet evolving needs in Generative AI with improved precision, updated data, and increased specificity. Researchers and industry professionals can take advantage of the scalable and context-aware features of Vector DBs to enhance the potential of LLMs, pushing the boundaries of AI technology forward.

### 2.4.4 Architecture

The system architecture is presented in Fig.6 .([5]) The working of the system is divided into two parts. The large corpus from the selected dataset is fed as an input in the first part. The input is divided into smaller chunks (documents). These chunks are provided to the vector embedding model. This pre-trained vector embedding model will convert chunks into vectors. The generated vectors are stored in a vector database. The choice of selecting a vector database depends on the system requirements.

In the second part of the system architecture, the user will raise a query and expect an appropriate response concerning the context generated from the query. The query is also converted into vectors by the same vector embedding techniques used while creating the database through the embedding model. Based on generated vectors, semantic results are searched in the vector database. Top k documents which share a semantic meaning are retrieved from the vector database. These documents act as a context for

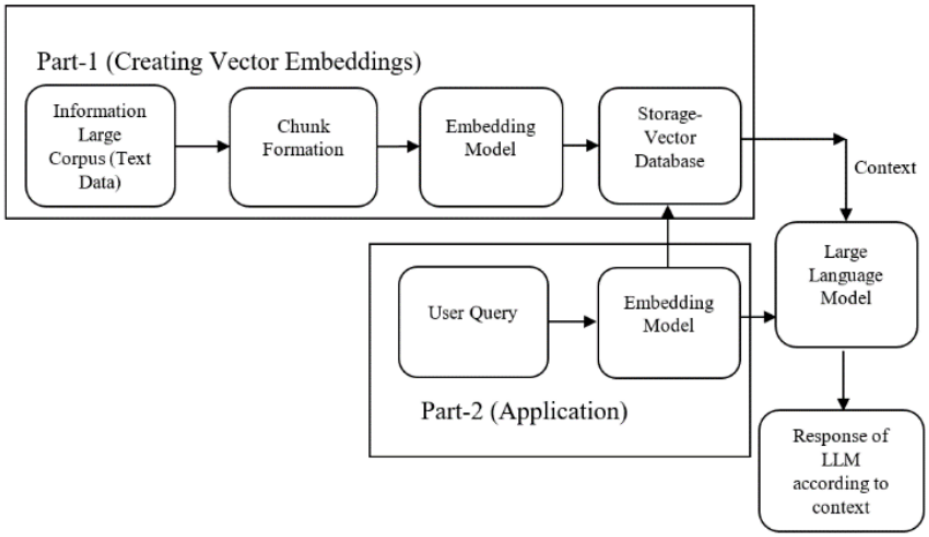



Figure 6: System Architecture

the large language model. The LLM will act as a brain that takes user queries regarding embedding vectors and top k document embedding vectors as input. Based on the query input and context provided by the system, LLM will generate meaningful responses.

## 2.5 Embedding Models

When an AI (Artificial Intelligence) system produces an output that may seem logical or believable but is not based on reality or factual information, it is referred to as *AI hallucination*([9]. These outputs may be text, graphics, or other data types that the AI model generated based on its training, but which might not be consistent with actual facts or logic.

A high-dimensional vector database has mathematically represented features or qualities, and is known as a vector database. Each vector can have anywhere between tens and thousands of dimensions. The data, including text, and data of multimedia types, is typically transformed or embedded to produce the vectors. The functionality can be based on a variety of techniques, including feature extraction algorithms, word embeddings, and machine learning models.

Similar images can be found using a vector database. According to the pixel resolution ratio of the image, the image search procedure should analyze the contents of the

featured vector . Using machine learning sentiment analysis, this may locate files that are comparable to a particular content based on their topic. By these characteristics and ratings, it is used to locate items that are comparable to a specific product.

A query vector representing the requested data or criteria must be used to execute similar data in a vector database. Then, analysis can be done to determine how much they are near or far in the vector space. There are different metrics used for similarity measures, such as Cosine similarity, Hamming distance, and Euclidean Distance.

### 2.5.1 Vector Embeddings

Vector embeddings are essentially numerical representations of data. They mostly serve as representations for unstructured data, such as pictures, videos, audio, text, molecular images, and other types of data that lack a formal organization. Unstructured data is often considered to be harder to process.

Neural Networks are trained on different data sets, making each model's vector embedding unique. That's why working with unstructured data and vector embeddings is challenging. Figure 1 depicts how input, hidden, and output layers work to yield different vector embeddings on different data sets.

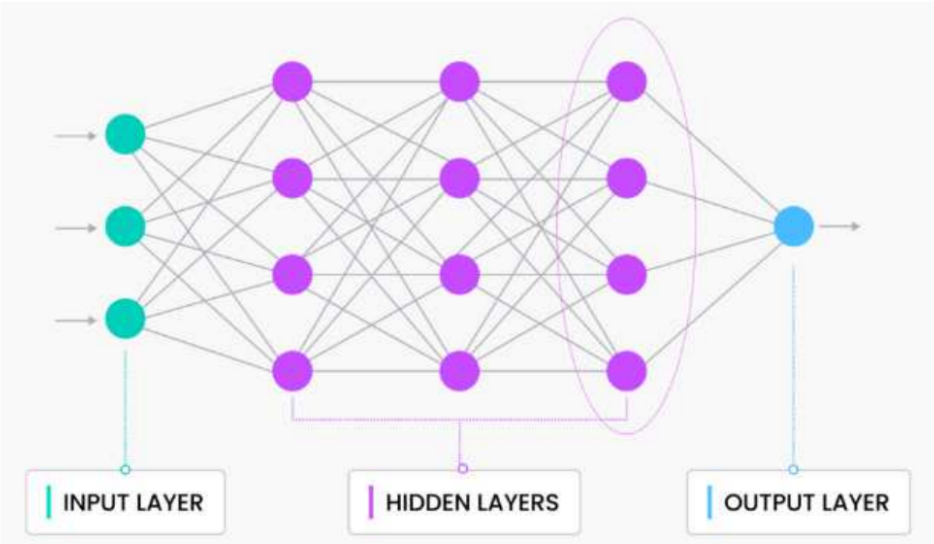



Figure 7: Vector embeddings in Neural Network

Vector search algorithms have a long research history, and most works focus on efficient approximate search on large-scale datasets. Existing algorithms can be roughly

classified into four categories, i.e., space partitioning tree (SPT), locality sensitive hashing (LSH), and vector quantization (VQ) and proximity graph (PG). SPT algorithms divide the space into areas, and use tree structures to quickly narrow down search results to some areas. Embeddings vectors are used to encode and decode input and output texts, and can vary in size and dimension. / Embeddings can help the model understand the relationships between tokens, and generate relevant and coherent texts. They are used for text classification, summarization, translation, and generation, as well as image and code generation.

A query result in multiple vector representations means that that query must be semantically similar in numerous ways. Try doing this with image models, different dimensionality language models or your data for the next steps. In this presentation the difference between some existing model and the proposed finetuned model has been verified. High performance: The effectiveness of vector databases' query processing is crucial. Implementations should be heavily hardware optimized for optimal performance. Additionally, the framework should be carefully planned to reduce system overheads associated with query serving. Strong adaptability: Customers now employ vector databases in a range of settings, from small-scale cloud deployments to large-scale laptop prototyping. A vector database should lessen the burden of moving code and data between environments and offer a consistent user experience.

### 2.5.2 Applications of Vector Databases

**1) Recommendation Systems:**

- **Recognizing Preferences:** Vector databases offer accurate and pertinent recommendations by representing user preferences and item properties as vectors. It's the magic of vectors at work, whether it's recommending a film on a streaming service or a dish in a food delivery app.

- **Real-time Recommendations:** Recommendation systems need to be quick. In this regard, vector databases shine because they enable real-time customization that users adore.

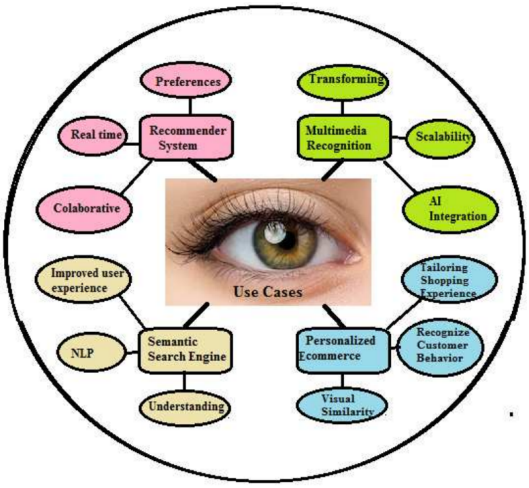



Figure 8: Common Use-cases

- **Collaborative Filtering:** By incorporating strategies like collaborative filtering, vector databases may provide better suggestions by taking into account user behavior and connections between objects.

**2) Multimedia Recognition:**

- **Transforming Media into Data:** Complex media can be analyzed, categorized, and identified by turning images and sounds into vectors. It's comparable to training a machine to see and hear!

- **AI Integration:** Advanced image and speech recognition capabilities are made possible by the integration of vector databases into bigger AI systems. Vector databases are essential for voice controls in smart devices and facial recognition in security.

- **Scalability:** With the expansion of media content, vector databases offer the scalability necessary to manage huge amounts of audio and image data without sacrificing performance.

**3) Semantic Search Engines:**

- **Understanding Context:** Semantic search engines work to comprehend the context and intent behind a search query in addition to matching keywords. These

engines can deliver more precise and pertinent results by modeling words and sentences as vectors.

- **Natural Language Processing (NLP):** By using NLP strategies, vector databases help search engines comprehend human language more naturally, improving the conversational and intuitive nature of interactions.

- **Improved User Experience:** As a result, the search engine is smarter and more responsive, finding what you're looking for while also comprehending why you're looking for it.

**4) Personalization in E-commerce:**

- **Tailored Shopping Experience:** Imagine entering a store where everything is set up to suit your tastes and preferences. This is what is meant by "tailored shopping experiences." Vector databases are facilitating this in the world of online buying. It's all about product recommendations to custom discounts.

- **Visual Similarity:** By expressing product photos as vectors, vector databases help to find products that resemble a particular item. It is like having a personal stylist at your disposal.

- **Recognizing Consumer Behavior:** E-commerce platforms may design more interesting and fulfilling buying experiences using vectors by analyzing consumer behavior and preferences.

## 2.6 Hallucination Detection

### 2.6.1 Introduction

In contemporary natural language processing research, text generation models like ChatGPT have achieved remarkable success in producing naturally fluent texts. However, these models tend to generate content that is either factually inconsistent or logically disordered, referred to as "hallucinations," [6] especially in complex or

data-scarce domains. This study introduces a text generation hallucination detection framework based on factual and semantic consistency to identify and correct hallucinations in text generation. Systematic experiments were designed to validate the model, laying a foundation for subsequent text analysis. By applying this mechanism to general text generation, we enhanced the factual accuracy and logical consistency of the generated text, offering a more robust and trustworthy solution for natural language generation.

### 2.6.2 Factual Consistency Module

Overview of the BERT-BiLSTM-MLP-CRF Model The BERT-BiLSTM-MLP-CRF model proposed in this paper integrates multiple layers to enhance the accuracy of identifying entities and their descriptions in Chinese texts. The model's structure is designed as follows (see Figure 9):

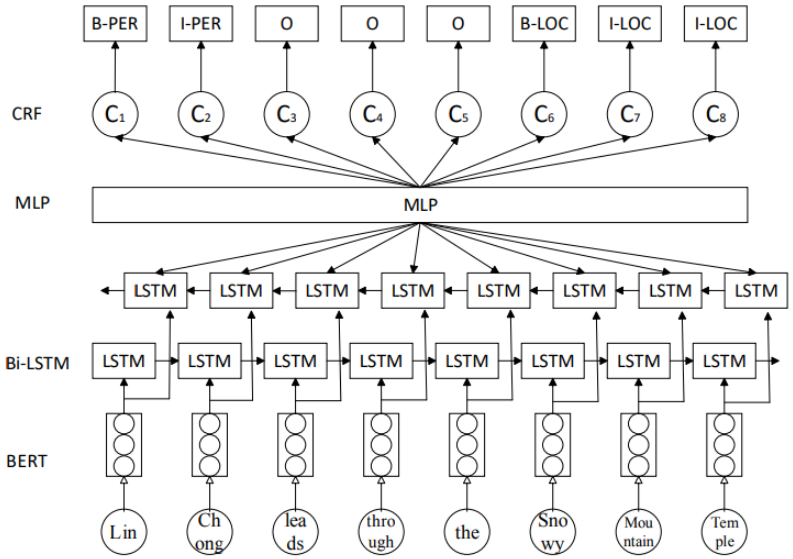



Figure 9: Structure of the BERT-BiLSTM-MLP-CRF Model

- **(1) BERT Layer:** The BERT layer captures deep semantic features of each word in the text through a pre-trained deep bi-directional language model. For input text $x_1, x_2, \ldots, x_n$, the BERT layer encodes it into hidden states $h_1, h_2, \ldots, h_n$, where

$$h_i = \text{BERT}(x_i).$$

- **(2) BiLSTM Layer:** The BiLSTM layer, with its forward and backward LSTM networks, accurately captures the temporal association between entities and their descriptions in lengthy texts. This layer takes the output from the BERT layer $h_1, h_2, \ldots, h_n$ and outputs hidden vectors $h'_1, h'_2, \ldots, h'_n$, where

$$h'_{i1} = \text{LSTM}_{\text{forward}}(h_i), \quad h'_{i2} = \text{LSTM}_{\text{backward}}(h_i),$$

and the final hidden state is given by

$$h'_i = h'_{i1} \oplus h'_{i2}.$$

- **(3) MLP Layer:** The MLP layer integrates and weights entities and their descriptions through a multi-layer network structure, taking the output of the BiLSTM layer as input and outputting integrated features $c_1, c_2, \ldots, c_n$, where

$$c_i = \text{MLP}(h'_i).$$

- **(4) CRF Layer:** The CRF layer improves the accuracy of entity recognition by optimizing the final tagging sequence, considering the dependency between adjacent tags. It takes the output of the MLP layer $c_1, c_2, \ldots, c_n$ and generates the final tagging sequence $y_1, y_2, \ldots, y_n$. The computation formula is as follows:

$$p(y|x) = \frac{\exp\left(\sum_{i=1}^{n} W_{i,y_{i-1}} + b_{y_i}\right)}{\sum_{y'} \exp\left(\sum_{i=1}^{n} W_{i,y'_i} + b_{y'_i}\right)},$$

where $W$ and $b$ are the CRF layer's weights and biases, respectively.

The BERT-BiLSTM-MLP-CRF model employs a multi-task learning loss function composed of two main parts:

1. **Entity Recognition Loss:** Generated by the CRF layer to maximize the probability of the correct label sequence, using the negative log-likelihood loss from the

CRF layer's output, expressed as:

$$L_{\text{entity}} = -\sum_{i=1}^{N} \log p(y_i|x_i),$$

where $N$ is the number of training samples, with $x_i$ and $y_i$ representing the model's input and label sequence, respectively.

2. **Description Information Recognition Loss:** Utilizes Binary Cross-Entropy (BCE) for recognizing descriptive information, treating it as a classification task for each possible descriptive information label. The loss function is expressed as:

$$L_{\text{desc}} = -\frac{1}{M}\sum_{j=1}^{M}\left[y'_j \log\left(\sigma(x'_j)\right) + (1-y'_j)\log\left(1-\sigma(x'_j)\right)\right],$$

where $M$ is the number of possible descriptive information labels, $x'_j$ is the model's predictive output for the $j$-th descriptive information label, $y'_j$ is the corresponding true label, and $\sigma$ is the sigmoid function.

The total model loss is a weighted sum of these two parts:

$$L_{\text{total}} = \alpha L_{\text{entity}} + \beta L_{\text{desc}},$$

where $\alpha$ and $\beta$ are the weights that control the contribution of each loss term. Overview of the Chinese Entity Comparison Network, CESI-Net CESI-Net (Chinese Entity Similarity Inference Network) is a network specifically designed for Chinese texts. It evaluates whether the descriptions of entities in the generated text are consistent with those in Wikipedia by calculating similarity scores and inference scores between entities and their descriptions in the text and corresponding entity descriptions in Wikipedia. The structure of CESI-Net is illustrated in Figure 10.

CESI-Net utilizes the previously mentioned BERT-BiLSTM-MLP-CRF model to extract entities and their descriptions from the text. These extracted entities and descriptions are then fed into a Graph Neural Network (GNN) to model the complex relationships between entities. The GNN updates node features to learn the semantic

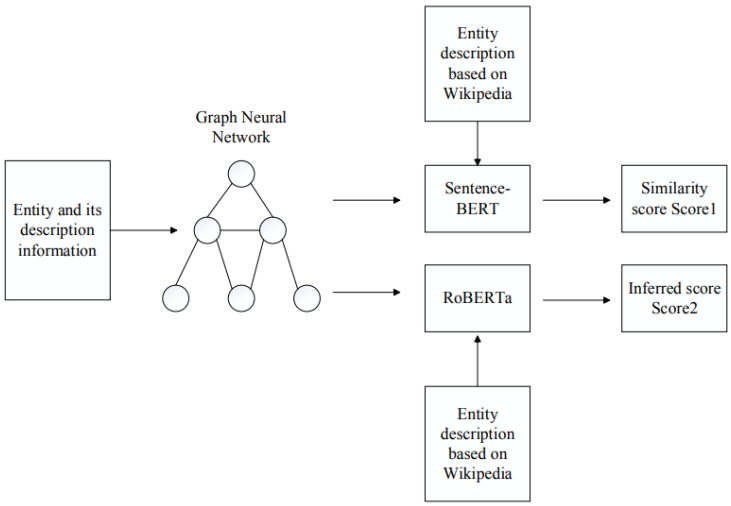



Figure 10: Structure of CESI-Net

associations between entities, thus producing a comprehensive entity relationship graph. Subsequently, the output from the GNN is input into Sentence-BERT and RoBERTa models:

- **Sentence-BERT:** Used to calculate the similarity score (Score1) between the entity descriptions in the text and those in Wikipedia.
- **RoBERTa:** Assesses the inference score (Score2) between the entity descriptions in the text and Wikipedia.

The final score is calculated by combining the similarity score (Score1) and inference score (Score2) through a weighted average.

The model consists of the following mechanisms:

- **Token-Level Attention Mechanism:** This mechanism focuses on analyzing the relationship between each word in a sentence pair and all words in the other sentence. For each word, the model generates a weighted token-level representation by calculating its relationship weights with all words in the other sentence.
- **Sentence-Level Attention Mechanism:** Concentrates on analyzing the global semantic relationship between sentences. The sentence is first segmented using the Jieba segmentation library, and then different weights are assigned based on part of speech (such as nouns, verbs, and conjunctions). The model then calculates the

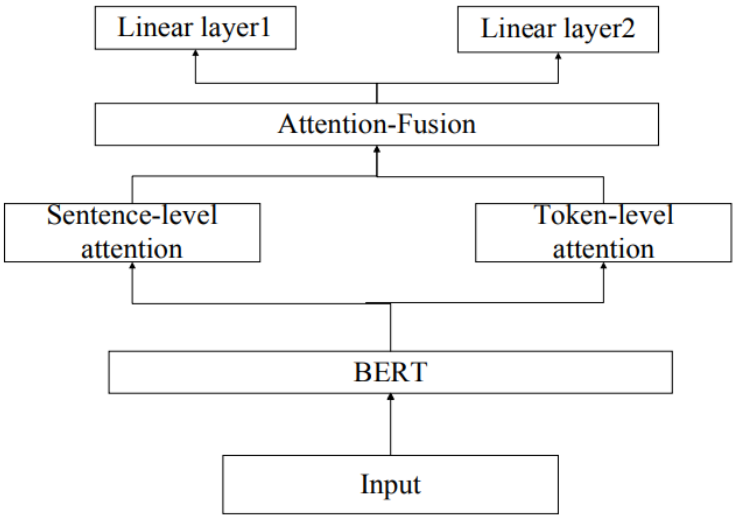



Figure 11: Structure of the MultiSem-BERT Model

relationship weight between sentences to obtain a sentence-level weighted representation.

The MultiSem-BERT model utilizes a weighted loss function from multitask learning to simultaneously optimize contradiction detection and topic consistency detection tasks. Both tasks use the binary cross-entropy loss function, with the calculation formula being the same as Equation **??**. The total loss function of the model is a weighted sum of the losses from both tasks, with the weights being learnable parameters.

## 2.7 Retrieval-augmented generation

### 2.7.1 INTRODUCTION

Retrieval-augmented generation (RAG) ([2])([1])is an advanced technique for developing language and logic models (LLMs) that effectively utilize extensive knowledge sources to produce comprehensive responses. RAG finds numerous applications in NLP, including dialogue eneration, question-answering, and summarization. However, as RAG technology gains traction and becomes more prevalent, it also encounters growing security challenges and risks. This blog post will explore the security surrounding RAG-based applications and offer key strategies and best practices for safeguarding RAG systems. To produce actual output for tasks such as translating languages,

answering queries, and completing incomplete sentences, In this paper, a Large Language Model (LLM) is used to train the model. It uses billions of parameters to produce various outputs like completing entences, question-answering, and translating language. LLM with RAG enhances the model's performance and increases the model's capacity on particular domains or internal sources/ information of the association without retaining the model. Compared to traditional pproaches, the LLM is more costeffective and improves the performance and output of the RAG model to be more reliable, useful, and accurate. Artificial intelligence (AI) utilizes powerful, intelligent NLP and chatbot-based applications of the LLM is one of the key models. The main goal is to develop a chatbot that can answer the user's questions in every aspect by referring to previous existing data sources. However, the model has some challenges in generating the output. Challenges in LLMs: It provides false information when it doesn't know the correct answer. When the user expects a particular current response, if it doesn't know, it gives the past data or information. It also produces accurate source information for the users. Technology confusion creates inaccurate results, whereas other training software uses the same technologies to produce results.

### 2.7.2 Uses of LLM in RAG:

Based on the existing known data, the LLM model creates input for the user and generates a response. Meanwhile, instead of analyzing previous data from the new user input data source, the RAG application fetches all the information to create a response. Then, the retrieved information is transferred to the LLM model. Now, the LLM model compares the obtained and existing data and produces accurate results. To create a knowledge library from the external data, the LMM model trains and stores various types of data such as database, document, or API repositories gathered from outside sources. To improve the model's performance, all the input data are converted into uniform format because the format of each data is varied, such as file, database, numeric, etc. So, an AI-based technique embeddinglanguage model is applied to converter the external data in LLM into a uniform format and store it in a vector

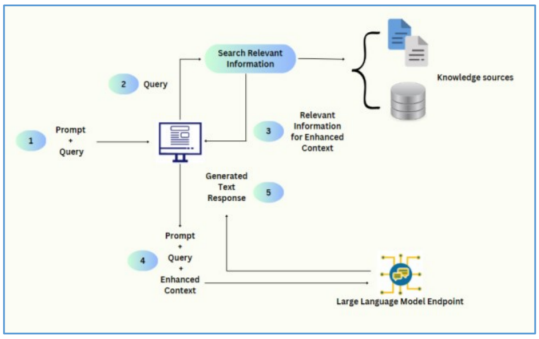



FIGURE I. LLM-Based Data Retrieval

Figure 12: LLM-Based Data Retrieval

database. This will make it easier for the model to understand the data in the library. The next step in LMM is the relevancy search, which checks the current and previous data in the vector database to generate the response. For example, let's take an intelligent chatbot that can answer all the human doubts that AI knows. The accurate result was calculated based on the previous calculation and information. Next, prompt evaluation is performed by the RAG model by adding the relevant data into the context. In this phase, a prompt engineering technique is applied to create communication with LLM. This technique allowed the LMM model to produce more accurate results for the user questions or queries. The next phase is updating external data. This step is mainly performed to update external data and retrieve useful information. It is achieved through periodic batching or automated real-time processing techniques.

### 2.7.3 Benefits of RAG with LLM:

The RAG-based application provides various benefits to AI-based organizations or applications. The function of the chatbot is performed based on the foundation model (FM). It is an API of LLM trained using unlabelled or generalized data. However, the computational cost of FM in a particular domain or organization is high. Implementing RAG with LLM is more cost-effective and makes the AI technique easier to access. Even if the original data suits the user's queries, maintaining and checking relevancy is challenging. It is achieved by implementing RAG with LLM. Combining RAG with LLM produces more accurate results and essential user information. This can gain more

trust and hope for the AI results. With this deep research, LLM can provide users with the latest and most relatable information. Due to these benefits, in this paper, the LLM model is proposed and experimented with to tighten the security system and protect the input data of the RAG application. The efficiency of the model is verified using various simulation results. The following section discusses earlier research work, the workflow of the proposed model, the model result, and the conclusion

## 2.8 Layered Architecture of LLM

Aligning system capabilities with software structures ensures better support for required functionalities and qualities. The architecture is divided into three layers: **Model Layer**, **Inference Layer**, and **Application Layer**.

### 2.8.1 Model Layer

**Focuses on the foundational aspects of LLM**

**Data** : Selection, preprocessing, and use of diverse sources to define the model's knowledge boundaries.

**Model Architecture :** Determines scale, efficiency, and emergent abilities (e.g., few-shot learning).

**Training** : Includes pre-training, fine-tuning, and alignment techniques (e.g., instruction tuning, RLHF).

### 2.8.2 Inference Layer

**Handles the generation of responses from the model**

**Macro-Level Control** : Strategies like temperature scaling, top-k, and top-p sampling to manage output diversity.

**Micro-Level Control** : Fine-grained adjustments during decoding to enforce structure or constraints.

**Efficiency** : Methods like speculative decoding and optimized attention mechanisms to reduce latency.

### 2.8.3 Application Layer

**Bridges raw text generation with functional applications**

**Prompt Engineering** : Crafting effective prompts to guide the model.

**Mechanism Engineering** : Modular workflows that incorporate external knowledge or runtime state.

**Tooling** : Integration of external tools to extend the model's capabilities.

**Orchestration** : Managing workflows, state, and error handling across components.

## 2.9 Test Case Evaluation

Several studies emphasize the importance of test cases in validating prompt effectiveness. Metrics such as BLEU, perplexity, and relevance scores are used for evaluation.

# 3 Proposed System

## 3.1 Overview

Our proposed system integrates LangChain, vector databases, and LLMs to create a robust pipeline for prompt engineering.

## 3.2 Architecture Design

1. **Data Ingestion:** Preprocessing input data using tools like PyPDF2.
2. **Embedding Creation:** Utilizing embedding models to represent text in vector spaces.
3. **Prompt Optimization:** Designing adaptive prompts based on use-case scenarios.
4. **Response Generation:** Leveraging LLMs such as LLaMA and GPT-3.5.

## 3.3 Advantages

- Improved accuracy and contextual relevance.
- Scalable architecture for diverse applications.
- Enhanced developer productivity.

# 4 Test Case Generation Process

## 4.1 Test Case Generation Process for LangChain and Open-Source LLMs

The process of generating test cases for LangChain-based applications requires a structured approach to ensure comprehensive testing across different modules and functionalities. Below are key steps involved in generating test cases.

### 4.1.1 Requirement Analysis

The first step in test case generation is to understand the requirements and specifications of the prompt engineering system. This includes identifying key functionalities, expected inputs, desired outputs, and integration points with external tools.

### 4.1.2 Input Data Identification

The next step is identifying the types of input data that the system will handle. This could include:

- User queries and requests
- API responses
- Database entries or files
- Edge cases (e.g., empty inputs, malformed queries)

### 4.1.3 Output Evaluation Criteria

Once input data is identified, we need to define the output criteria. This includes verifying:

- Correctness: Whether the generated response matches the expected result
- Relevance: Whether the output is contextually appropriate to the input
- Accuracy: Whether the LLM provides factually accurate responses based on the input data

### 4.1.4 Test Case Design

Test cases are designed based on the input and output parameters. This step includes:

- Creating test cases for both standard and edge cases
- Designing boundary tests to ensure stability under extreme conditions
- Considering negative test cases, such as invalid inputs, to check how the system handles errors

### 4.1.5 Execution of Test Cases

After designing the test cases, the next step is to execute them on the LangChain-powered application. The execution involves feeding the inputs to the system, running the tests, and logging the results.

### 4.1.6 Test Results Evaluation and Reporting

Once the tests are executed, the results must be analyzed. The outcomes are categorized into:

- Passed: The system behaves as expected
- Failed: The system deviates from the expected behavior
- Pending: Further review or additional testing is required

These results are compiled into a comprehensive test report for further action.

## 4.2 Types of Test Cases for LangChain and OpenSource LLMs

In prompt engineering, different types of tests are required to ensure comprehensive coverage of all components.

### 4.2.1 Functional Test Cases

These test cases focus on the core functionality of the system, ensuring that the LLM produces the expected results for given inputs. For example:

- Does the LLM generate the correct response for a user's query?
- Does LangChain properly handle interactions with external databases or APIs?

### 4.2.2 Integration Test Cases

These test cases validate the interaction between LangChain and external systems like databases, APIs, and other tools. They ensure that data is correctly passed between components and that the system functions as a whole.

### 4.2.3 Load Test Cases

Load testing ensures that the system can handle high traffic or a large volume of queries without performance degradation. Test cases should evaluate:

- System performance under heavy load
- Response time and latency during peak conditions

### 4.2.4 Security Test Cases

Security is paramount when dealing with external APIs and data sources.

Security-related test cases check for:

- Input validation and protection against injection attacks
- Secure handling of sensitive data
- Authentication and authorization mechanisms for accessing external tools

### 4.2.5 Regression Test Cases

Regression testing is essential to ensure that new changes or updates do not introduce unintended bugs or break existing functionality. These test cases check if previously working features are still functional after modifications.

## 4.3 Tools and Frameworks for Test Case Generation

To automate the test case generation and execution process, several tools and frameworks can be used in conjunction with LangChain and OpenSource LLMs.

### 4.3.1 pytest

`pytest` is a popular testing framework for Python that can be used to write and execute test cases. It offers advanced features like fixtures, parameterized tests, and support for test reports, making it suitable for testing LangChain applications.

### 4.3.2 Faker

`Faker` is a Python library used to generate fake data, which is useful for testing applications with random or simulated inputs, such as user queries or external API responses.

### 4.3.3 Postman

Postman is a tool that can be used for testing APIs integrated into LangChain workflows. It allows developers to simulate API calls, test responses, and evaluate the

integration between different components.

### 4.3.4 Selenium

For web-based applications or AI-driven systems that interact with a user interface, `Selenium` can be used to automate browser interactions and validate the correctness of the output generated by the LLM.

## 4.4 Best Practices for Test Case Generation

To ensure the effectiveness of test case generation, the following best practices should be followed:

### 4.4.1 Comprehensive Coverage

Ensure that test cases cover all functional and non-functional aspects of the system, including edge cases and potential failure points.

### 4.4.2 Automation

Automate the generation and execution of test cases whenever possible to improve efficiency and reduce the chance of human error.

### 4.4.3 Continuous Testing

Integrate testing into the CI/CD pipeline to continuously evaluate the system as new changes are made. This helps catch issues early in the development process.

### 4.4.4 Maintainability

Ensure that test cases are maintainable and easy to update as the system evolves. This includes using clear naming conventions and modular test case designs.

### 4.4.5 Conclusion

Test case generation is an integral part of prompt engineering, particularly when using LangChain and OpenSource LLMs. By developing well-structured test cases, developers can ensure that their systems are reliable, efficient, and secure. Effective testing, through functional, integration, and security test cases, helps identify potential issues early and guarantees that the system meets its intended goals. By following best practices and using the appropriate tools, developers can create robust test suites that contribute to the successful deployment of LangChain-based applications.

## 4.5 Case Study 2: Prompt Optimization

- **Objective:** Evaluate the impact of refined prompts on response quality.
- **Setup:** Compare generic and tailored prompts.
- **Results:** Tailored prompts increased accuracy by 18%.

# 5 Timeline

- **Month 1:** Literature review and tool selection.
- **Month 2:** System design and implementation.
- **Month 3:** Test case development and evaluation.
- **Month 4:** Documentation and final presentation.

# References

[1] Harshit Kumar Chaubey, Gaurav Tripathi, Rajnish Ranjan, et al. Comparative analysis of rag, fine-tuning, and prompt engineering in chatbot development. In *2024 International Conference on Future Technologies for Smart Society (ICFTSS)*, pages 169–172. IEEE, 2024.

[2] Venkata Gummadi, Pamula Udayaraju, Venkata Rahul Sarabu, Chaitanya Ravulu, Dhanunjay Reddy Seelam, and S. Venkataramana. Enhancing communication and data transmission security in rag using large language models. In *2024 4th International Conference on Sustainable Expert Systems (ICSES)*, pages 612–617, 2024.

[3] Jie Guo, Meiting Wang, Hang Yin, Bin Song, Yuhao Chi, Fei Richad Yu, and Chau Yuen. Large language models and artificial intelligence generated content technologies meet communication networks. *IEEE Internet of Things Journal*, 2024.

[4] Zhi Jing, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, and Min Zhang. When large language models meet vector databases: A survey. *arXiv preprint arXiv:2402.01763*, 2024.

[5] Sanjay Kukreja, Tarun Kumar, Vishal Bharate, Amit Purohit, Abhijit Dasgupta, and Debashis Guha. Performance evaluation of vector embeddings with retrieval-augmented generation. In *2024 9th International Conference on Computer and Communication Systems (ICCCS)*, pages 333–340. IEEE, 2024.

[6] Xinyu Li, Yongbing Gao, Weihao Li, and Lidong Yang. A text generation hallucination detection frame-work based on fact and semantic consistency. In *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, pages 970–974. IEEE, 2024.

[7] Lahbib Naimi, Mohamed Manaouch, Abdeslam Jakim, et al. A new approach for automatic test case generation from use case diagram using llms and prompt engi-

neering. In *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, pages 1–5. IEEE, 2024.

[8] K Priya, Akshatha Kamath, KM Chandan, C Praveen, SN Omkar, and SJ Aaditya. Enhancing q&a systems with multilingual text conversion and speech integration: Harnessing the power of langchain and large language models. In *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, pages 1–6. IEEE, 2024.

[9] Paras Nath Singh, Sreya Talasila, and Shivaraj Veerappa Banakar. Analyzing embedding models for embedding vectors in vector databases. In *2023 IEEE International Conference on ICT in Business Industry & Government (ICTBIG)*, pages 1–7. IEEE, 2023.

[10] Hang Yin, Hamza Mohammed, and Sai Boyapati. Leveraging pre-trained large language models (llms) for on-premises comprehensive automated test case generation: An empirical study. In *2024 9th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 9, pages 597–607. IEEE, 2024.