Artificial Intelligence Text Processing Using Retrieval-Augmented Generation: Applications in Business and Education Fields

Bogdan – Stefan POSEDARU
Bucharest University of Economic Studies, Bucharest, Romania
bogdan.posedaru@csie.ase.ro

Florin – Valeriu PANTELIMON
Bucharest University of Economic Studies, Bucharest, Romania
florin.pantelimon@csie.ase.ro

Mihai – Nicolae DULGHERU
Bucharest University of Economic Studies, Bucharest, Romania
dulgherumihai19@stud.ase.ro

Tiberiu – Marian GEORGESCU*
Bucharest University of Economic Studies, Bucharest, Romania
**Corresponding author, tiberiugeorgescu@ase.ro*

Abstract. *The article studies the current text processing tools based on Artificial Intelligence. A literature review is done emphasizing the dynamic evolution of AI-powered text analytics, having as its central tool ChatGPT and its capabilities. The focus is centered on the techniques and methods that are using embeddings in order to improve large language models (LLMs).*

In this paper is analyzed the current situation of the literature in terms of text processing using Retrieval-Augmented Generation and is highlighted the potential of this technology to enhance the interpretability and trust in applications critical, such as those related to education or business. AI has revolutionized natural language processing (NLP), which facilitated the machines to interpret and generate text efficiently and accurately. In addition, large language models with external knowledge bases have been developed. These are used to produce more accurate and contextually relevant text responses. This approach is called Retrieval-Augmented Generation (RAG is one of the most significant recent advancements in this field.

Based on our study, two use cases are implemented to show the applicability of our study: one related to education and one related to business IT-related documents. The methodology describes the techniques used. This includes retrieval-augmented generation and embedding stored using vector databases. Our custom models are evaluated by comparing them to the general ones, without embeddings, showing superior performance.

The article highlights remarkable progress in Retrieval-Augmented Generation (RAG), which is used for AI text processing with a focus on business and education fields. Further in this paper, many of the most significant highlights are presented, which include a scalable framework for AI applications, a new integration of Retrieval-Augmented Generation and embeddings, practical application demonstrations, bridging gaps in the analysis op AI text, significant development in AI performance and optimizing educational and business processes.

Keywords: AI Text Processing, Large Language Models (LLM), Natural Language Processing (NLP), Retrieval-Augmented Generation (RAG), Embeddings, ChatGPT, LangChain

DOI: 10.2478/picbe-2024-0018

Introduction

In the field of artificial intelligence, integrating OpenAI models with vector embeddings has become a disruptive technique recently, particularly among applications that demand complex natural language generation and understanding. With the help of vector embeddings and the sophisticated capabilities of OpenAI's generative pre-trained transformers, such as ChatGPT, higher accuracy and contextual relevance in tasks ranging from information retrieval to conversational agents can be obtained. A comprehensive literature review on ChatGPT was done by (Ray, 2023).

The article presents advanced AI text processing technologies, also has a focus on embeddings and LangChain to enhance existing models. The article (Roumeliotis & Tselikas, 2023) presents a review as well as main applications in the fields of healthcare, education, finance, Machine Learning, translation, math, social, industry and also art fields. A meticulous review on the scientific literature is conducted in order to examine our research within the broader landscape of AI-powered text analytics. To conclude, both the steps make and challenge are highlighted.

A comparative analysis to support our research methodology was done. The augmented models with LangChain and embeddings are measured against the traditional models in the context of the business and education sector use cases. These two have been chosen in order to highlight the flexibility and applicability of the enhancements in the real-world.

The research questions are used to examine the custom models' potential to outperform their general equivalent. In addition, the article presents the examination of the practical application of LangChain in extending the AI practical usage as a text-processing tool. The paper's aim is to present a complete analysis of the implications of using advanced embeddings and LangChain in AI text processing technologies.

One of the latest innovations in software development is using advanced technologies to develop for complex requirements. The authors' research, completed by an application that uses an elaborate combination of the LangChain framework with the retrieval tools and intelligent agents that it provides, OpenAI's embeddings and the ChatGPT model, is not an exception. Using these technologies to maximize the application's capability is a calculated decision. This facilitates the interpretation, comprehension and reply to developer queries with a high degree of relevance and accuracy. The operation of the program depends on the components. Some of the tasks that it performs are understanding user requests and delivering and acquiring exact information. The next sections cover not only the details of these technologies and their functions, but also their contributions to making a responsive assistant for developers.

The advanced development of AI models for text interpretation happened as a consequence of integration of deep learning. Artificial neural networks have surpassed human-level performance in various subfields of AI (Harsh Trivedi, 2023). However, many of these models lack interpretability, making it challenging to trust their decisions, especially in critical scenarios (SILVA, FREITAS, & HANDSCHUH, 2019). Efforts are being made to create explainable AI, with a focus on image processing models and limited work in natural language processing (NLP) and other fields. Some studies have analyzed filter values in convolutional models. These models are trained on textual data in order to identify important words in the corpus and reducing training time (Harsh Trivedi, 2023).

The NLP models, their decisions, blind spots and biases are better explained as a result of these approaches. Before deployment, the evaluation of the performance of ML models in NLP and computer vision is crucial. Enhancing ML model interpretability can shed a light on the relevance of model performance with domain knowledge. The reason behind the users' predictions is challenging due to the complexity of AI models. Interpretability is an ever evolving domain. The rapid changes lead to still having unresolved gaps that need to provide more human-centered interpretability solutions. Now AI language models offer solutions for textual interpretation in social and humanities science. It is complementing the traditional and raising new questions for further research. The models can analyze different texts and additionally examine the coherence of its meaning.

AI-powered text analytics

AI-powered text analytics can extract valuable insights from unstructured data sources such as social media posts, surveys, live chat history and reviews through the use of Natural Language Processing (NLP). Some of the most powerful features of these technologies include Named Entity Recognition (NER). This can identify important entities like names, organizations, locations, dates and times from text data (Nuthakki, 2023). Another important element is sentiment analysis. This can identify emotions and opinions in the text, which is essential for understanding the customer and brand reputation (Madanchian, 2023). Two tools that can help identify topics and trends in text data are Heat Maps and Word Clouds. In order to provide a more comprehensive understanding of data, the integration of text analysis with computer vision or speech recognition is required.

There are several text analysis with different features and capabilities that are available. These tools can be grouped into four categories based on their functionality and applicability: natural language analysis and processing (Google Natural Language AI, Amazon Comprehend, Microsoft Azure Text Analytics), analysis of texts and feedback (Converseon, Wordnerds, SAS Visual Text Analytics, Chattermill, Thematic, Monkeylearn, WordStat, Lexalytics, Keatext), AI-based editing or conversion tools (EdrawMax AI, UPDF AI) and data analysis and Machine Learning (RapidMiner).

A powerful computational tool is represented by the numerical vectors created through vector embeddings. This is relevant when representing different types of data such as audio, text etc. The method is characterized by specialized techniques and algorithms. These are tailored to the specific data type and employing deep neutral networks.

For text data, word embeddings like Word2Vec (Johnson, 2023), GloVe (Jeffrey Pennington, 2024) and BERT (Yang, 2023) are indispensable tools that capture the semantic meaning of words and their relationships within a language. There are different natural language processing tasks for which the embeddings have proven successful. This has enabled accurate analysis of text data. These include sentiment analysis, document classification, machine translation and information retrieval.

The Word2Vec model is widely recognized and extensively utilized in natural language processing for generating high-quality word embeddings. Diverse NLP tasks make use of the Word2Vec, which has demonstrated great results. A research paper has explored a technique to scale Word2Vec on a GPU cluster, leading to significant acceleration without compromising accuracy (Jiaoyan, 2021). The potential of Word2Vec has been proven by another study. This identifies meaningful relationships in ecological music and at the same time highlighting its ability to differentiate functional chord relationships and represent key relationships (Izzidien, 2022). Furthermore, Word2Vec has been leveraged to analyze molecular cavities, where a unique approach has been proposed to topologically represent cavities through vectorization, allowing for the identification of change patterns.

GloVe stands for Global Vectors for Word Representation which is an unsupervised learning algorithm. The training process involves analyzing word-word co-occurrence statistics from a corpus, which results in fascinating linear substructures in the word vector space (Jeffrey Pennington, 2024).

BERT, short for Bidirectional Encoder Representations from Transformers, is a language representation model that can pre-train deep bidirectional representations from unlabeled text by considering both the left and right context in all layers. BERT can be adjusted to create advanced models for multiple natural language processing tasks. It has made significant strides in performance on tasks such as GLUE score, MultiNLI accuracy and SQuAD question answering Test F1 (Wei Jin, 2010). It has also been used in software maintenance to analyze behavioral differences between different versions of a program (Jacob Devlin, 2019). This has enabled developers to find regression faults. Additionally, its ability to analyze context and conduct parallel operations have helped improve not only the efficiency, but also the analysis precision in the evolution of intelligent question-and-answer systems. Furthermore, it was crucial in developing a scalable routing approach for cloud data center networks. It has also reduced traffic control and network congestion.

Large Language Models (LLMs) offer another approach that can perform the same tasks as Word2vec and achieve even better results. However, they require extensive pre-training with vast amounts of text, followed by fine-tuning with manageable domain-specific data sets.

Retrieval augmented generation

The method of Retrieval-Augmented Generation combines large language models with external knowledge resources to enhance text generation. This enhances the boundaries of language models by gaining information from external sources. A few limitation examples are factual inaccuracies and hallucinations. The generation and gathering of information are combined in recent methods to generate better outputs. This iterative approach utilizes the model's output as a context to gain more relevant knowledge. In this way a better output in the next iteration is generated. This approach achieves strong performance in question answering and fact verification, by preserving the flexibility in generation without structural boundaries (Yunfan Gao, 2023).

Large Language Models often require the incorporation of user-specific data not covered by the pre-existing training dataset. Retrieval Augmented Generation is a key technique that can help improve the accuracy and relevance of the LLM's replies by retrieving external data and including it in the generating process (Bonan Min, 2023).

Retrieval chain

The retrieval chain is a process used to find relevant information or evidence in a sequence for answering questions. While many models have focused on combining multiple modules or discretely retrieving evidence, they often overlook the importance of path information, such as sentence order, which is crucial for evidence retrievers. To address this limitation, a new approach called IRCoT has been proposed. This approach interleaves retrieval with steps in a reasoning chain and guides the retrieval process based on already derived information (Harsh Trivedi, 2023). By using this method, improvements have been demonstrated in downstream and retrieval question-answering tasks. This has stopped the model hallucination and developed a more accurate reasoning.

LangChain is known as an important support for developing applications using RAG. It is well-prepared to meet the diverse needs of RAG applications, having a comprehensive suite of tools that cater to a broad range of complexities. Despite the apparent simplicity of the retrieval process, several subtle intricacies require careful consideration. The main components of LangChain make reliable and efficient retrieval possible. In this way the LLM's capability to provide personalized and relevant information is enhanced.

Vector databases are used to store and retrieve vector embeddings. They demonstrate extraordinaty proficiency in managing high-dimensional data. Such data might include embeddings, utilizing advanced indexing methods and parallel processing capabilities. Some popular embedding models include Universal Sentence Encoder (USE) and MobileNet embeddings.

The choice of the best embedding model and vector database depends on specific requirements, use cases and preferences. Please find below a detailed analysis of some of the most popular tools that use vector embeddings for text analysis. These include ContextClue, Cohere, Kofax AI, Docugami, Glean, Onfido, Vectara, CaseText, BrainyPDF, Unriddle, Tavily AI, Writer, PopAI, Petal, AskYourPDF, Stonly and Odin (Lisowski, 2023).

Cohere is renowned for its advanced language models. These are designed for text generation, text comprehension and various NLP tasks. They provide versatile solutions relevant for many domains, such as strong text generation and comprehension capabilities. However, their usage demands substantial computational resources and training and fine-tuning the models can be complex.

Vectara is a semantic search that provides accurate and rapid search capabilities within textual documents or databases. The engine supports vector embedding in order to enhance the relevance of search results. However, it may require fine adjustments and optimizations for specific datasets.

CaseText is an NLP and AI tool designed for organizing, analyzing and searching legal documents. Even if its primary focus on the legal domain restricts its applicability to other areas, it optimizes the legal research process, minimizing search and analysis time.

Tavily AI is a tool with specific features that is not documented in great detail, which can include language comprehension and content generation models. It is designed to adapt to diverse text types and contexts. Unfortunately, the lack of detailed information makes it challenging to evaluate specific functionalities.

Writer is an AI Writing Assistant that offers style suggestions, grammar corrections and content recommendations based on contextual understanding. It increases writing quality and includes custom styles and guidelines. However, its performance may vary depending on the quality of training data and the specificity of the dataset used.

TrainMyAI is a platform designed to train AI models which offers flexibility in training models for specific tasks. This includes NLP tasks, such as the use of embedding for text analysis. However, technical knowledge is required is required for efficient model training and complexity in setting and adjusting parameters.

Each of the instrument is making a great contribution to its field. These have specific advantages that make them suitable for specific tasks or databases. Selecting the right tool depends on the specific needs of the user, the resources available and the complexity of the text-processing task.

Research Hypothesis

Models are practical tools in educational and business settings due to their adaptability and versatility. These features make them suitable for resolving complex problems, optimizing processes and facilitating learning. In education, models serve as educational support. These help students understand complex concepts, apply the gained knowledge and develop skills for their future careers. In commercial contexts, models can provide valuable insights, enhance decision-making and drive strategic planning and innovation.

Improved and on-going training can significantly strengthen the performance of models. Models that are trained on diverse and comprehensive datasets are better equipped to understand and process information. This results in more accurate, relevant and contextually appropriate responses or predictions. It is suggested that the increased effectiveness of models in real-world applications depends on techniques and tools previously described.

Methodology

Large Language Models (LLMs) have many applications that need the incorporation of user-specific data that is not covered by the model's pre-existing training dataset, Retrieval Augmented Generation (RAG) being a key technique to help with this matter. To generate more accurate and contextually relevant replies, this technique involves retrieving external data and including it in the LLM during the generating step.

LangChain, a novel framework for working with Large Language Models, offers several useful features for applications involving Retrieval Augmented Generation. The retrieval process, the first stage in RAG, may seem simple at first, but there are several subtle intricacies involved.

Several essential LangChain components facilitate this process, increasing the reliability and effectiveness of the retrieval system and extending the LLM's capacity knowledge with personalized information. Figure 1 illustrates the main components of LangChain framework as well as the dataflow.

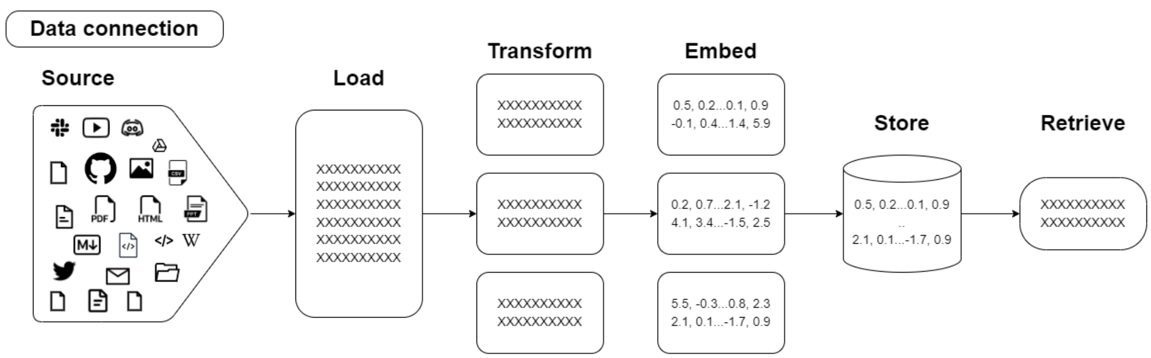

Figure 1. LangChain framework (LangChain, 2023)

The process of developing the Q&A application begins with the creation of vector embeddings from various document formats, such as Word or PDF files. Langchain helps with this process by analyzing the text in these documents using OpenAI's models and turning them into high-dimensional vector embeddings.

One important aspect of converting the documents to embeddings is splitting the large chunks of text into smaller semantically related pieces to fit into the Large Language Model's context window. By capturing the semantic core of the documents, the embedded content enhances the system with comprehension and data retrieval capabilities based on content similarity rather than keyword matching.

An innovative feature of our app is represented by the use of multiple LangChain retrieval tools, one for each possible prompt type sent by the user. As such, inside our app, we create two retrieval tools: one tool for retrieving product requirements and the other tool for retrieving technical guidance, as per the organization's guidelines. The code snippet shown in the figure below, we demonstrate creating the embeddings that will be used for one of the retrieval tools, the organization guidelines.


```
const fileNames = fs.readdirSync("org-docs");
 const langchainDocs = fileNames.map((fileName) => {
   const filePath = path.join("org-docs", fileName);
   const fileContent = fs.readFileSync(filePath, "utf8");
   return new Document({
     metadata: { fileName },
     pageContent: fileContent,
   });
 });

 const textSplitter = new RecursiveCharacterTextSplitter({
   chunkSize: 1000,
   chunkOverlap: 200,
 });
 const splitDocs = await textSplitter.splitDocuments(langchainDocs);
```

Figure 2. Creating embeddings for retrieval tools

Following their generation, the embeddings are kept in the vector database Pinecone. Pinecone's architecture is built to facilitate and speed up similarity searches and to efficiently handle data of large proportions. Leveraging the abilities of vector search to detect matches based on semantic similarity, the application may quickly and precisely retrieve the most relevant documents, or portions thereof, in answer to user queries by storing the document embeddings in Pinecone, figure 3 shows the code implementation.

```
const client = new Pinecone({
   apiKey: process.env.PINECONE_API_KEY_ORG_DOCS,
 });
 const pineconeIndex = client.Index(process.env.PINECONE_INDEX_ORG_DOCS);

 await PineconeStore.fromDocuments(
   splitDocs,
   new OpenAIEmbeddings({ openAIApiKey: process.env.OPENAI_API_KEY }),
   {
     pineconeIndex,
   }
 );
```

Figure 3. Implementation of Pinecone

The next step involves creating the two retrieval tools based on the presented embeddings, tools that the Langchain agent will use to retrieve the requested information. Upon receiving a query from the user and a decision from the agent on which tool to use, the application communicates with Pinecone over Langchain and does a similarity search to identify the most pertinent embeddings. To get the closest matching document embeddings, the query must first be converted into a vector embedding and then searched across the Pinecone database.

It is required to provide a natural language description of the retrieval tool, to help the agent decide on which tool to use depending on the user's prompt. The approach on how to create the retrieval tool for the organization guidelines is demonstrated in the code snippet, shown in the figure 4.


```
const createOrgDocsTool = async () => {
 const client = new Pinecone({
   apiKey: process.env.PINECONE_API_KEY_ORG_DOCS || "",
 });
 const pineconeIndex = client.Index(process.env.PINECONE_INDEX_ORG_DOCS || "");
  const vectorStore = await PineconeStore.fromExistingIndex(
   new OpenAIEmbeddings({ openAIApiKey: process.env.OPENAI_API_KEY }),
   { pineconeIndex }
 );

  const retriever = vectorStore.asRetriever();
  const retrieverTool = createRetrieverTool(retriever, {
   name: "org_docs",
   description:
     "Search for information regarding how to achieve certain technical tasks. If no info was found,
do not attempt to answer the questions, just state that there is no info in the docs for that specific
question. For any 'how to' type of questions, use this tool!",
 });

 return retrieverTool
}
```

Figure 4. Creating retrieval tools for organizations guidelines

Finally, based on the two retrieval tools that we create, we generate an intelligent Langchain agent that makes use of the OpenAI function calling to communicate its decisions on what actions to take and the GPT 3.5 model to facilitate the natural language dialogue between the user and the app, the code snippet is presented in figure below.

```
const createAgent = async () => {const llm = new ChatOpenAI({
 modelName: "gpt-3.5-turbo",
 temperature: 0,
});

const prompt = await pull("hwchase17/openai-functions-agent");

const tools = [
 await toolCreators.createOrgDocsTool(),
 await toolCreators.createProdReqTool(),
];

const agent = await createOpenAIFunctionsAgent({
 llm,
```

```
 tools,
 prompt,
});

const agentExecutor = new AgentExecutor({
 agent,
 tools,
});
```


Figure 5. Implementation of LangChain agent connected with ChatGPT

This workflow represents a powerful integration of cutting-edge technologies to create a Q&A application that leverages multiple embedded custom knowledge sets, generated from private documents. By leveraging Langchain's pre-built modules, combined with Pinecone's vector databases and also OpenAI's advanced natural language processing models, it is possible to build sophisticated applications capable of providing relevant answers to a wide range of user queries.

Results

In this section, our research findings are thoroughly analyzed, with the focus being the integration of embeddings with OpenAI's models and using LangChain as the orchestrator in AI text processing technologies. To support our research hypothesis that orchestrating AI models with embeddings using LangChain offers significant better results than off the shelf Large Language Models, we present two separate use cases, in the business and education sectors.

The results presented in this chapter are based on a use case that illustrates the capabilities of a Large Language Model that has been previously enriched with specific technical guides and product requirements documents using the methodology described above and is employed for an innovative software solution that aims to assist the organization's developers. The proposed solution takes natural language queries from the user as input and offers technical support in completing specific tasks, such as creating a validation schema for a form, as output. This application's main objective is to make it easier for developers to access relevant information, given a large documentation base, which has the potential to save an important amount of effort that otherwise would be required in order to manually search and locate the desired information, ultimately increasing productivity and streamlining the development process.

Several instances of user prompts and responses produced by the model are provided to illustrate the capabilities of the application that we developed. The comparison of the model's responses before and after embedding the contents for the two tools that our intelligent agent employs is the main focus of these examples.

As such, when prompted about technical recommendations regarding the steps and requirements for creating a form, the Large Language Model offers a general response, based on the pre-existing training datasets available for ChatGPT 3.5 turbo (figure 6).

```
{
   "input": "How do I create a form?",
   "output": "Creating a form, whether it's for a website, a survey, or data collection, involves
several steps and considerations depending on the platform you're using. Here are general
instructions for creating a form :\n
      1. Structure with HTML: Use <form> to create the form container. Inside, use <input>,
```

```
<textarea>, <select>, and <button> elements to create fields where users can enter data.\n
     2. Style with CSS: Use CSS to style the form elements for a more engaging user interface.\n
     3. Client-side Validation with JavaScript: Enhance your form with JavaScript for real-time
validation, giving users immediate feedback before submission.\n"
}
```

Figure 6. Creating a form - Response of general model based

However, after enriching the model's knowledge with custom documents (in this case, the organization's technical guides and guidelines), the model adapts its answer based on the findings from the custom knowledge, as can be observed in figure 7.

```
{
  "input": "How do I create a form?",
  "output": "The documentation provides a guide on how to create forms using Formik. Here are the
key steps involved:\n' +
  '\n' +
  '1. **Add the Model**: Define the model with a validation schema and initial values.\n' +
  '2. **Add the API Logic**: Implement API logic for form submission actions like creating,
updating, or deleting data.\n' +
  '3. **Add Mutations**: Implement mutations for handling form data changes.\n' +
  '\n' +
  'For detailed instructions and code examples, you can refer to the [Formik
documentation](https://formik.org).'"
}
```

Figure 7. Creating a form - Response of our custom model

As can be seen from the output above, the answer provided by our application is tailored to the developer's needs, according to the technical documentation that was embedded in the Large Language Model.

Regarding the second retrieval tool that our intelligent agent employs, the product requirements tool, the results are similar to the example above, except that the general answer provided by the model before embedding the custom knowledge is not helpful and may confuse developers regarding the actual product requirements, as can be observed below.

```
{
 "input": "What are the requirements for the admin create user account form?",
 "output": "Creating a user account form for an admin panel typically involves collecting essential
information that allows for the effective management and identification of users within a system.
The requirements can vary based on the application's specific needs and the level of security
desired. Here's a list of common requirements for an admin create user account form: [...]
Username, Password, First and Last Name [...]"
}
```

Figure 8. The answer of general model on creating an user account form

However, after embedding the product requirements documents into the appropriate retrieval tool, the model is now able to find the actual product requirements and return an accurate and invaluable response to the developer prompting the model, as shown in figure 9.

```
{
 "input": "What are the requirements for the admin create user account form?",
 "output": "The requirements for the admin create user account form include the following fields:\n'
+
   '- Role\n' +
   '- Name\n' +
   '- E-mail\n' +
   '- Phone Number\n' +
   '- Short bio\n' +
   '- Profile Picture\n' +
   '- Password\n' +
   '- Confirm Password\n' +
   '\n' +
   'These fields should be included in the form for creating a user account by an admin."
}
```

Figure 9. The answer of custom model on creating an user account form

The response returned to the user by our application is based on the real product requirements extracted from the embedded documents. As such, the necessary effort required to gather the information that one developer needs is drastically reduced, allowing developers to focus on the more important aspects of their duties.

Similar to the example above, we reproduced the behavior by creating a chatbot that answers questions asked by students enrolled in a software development course for a new technology. Figure 10 shows the answer of the general model to the question "How do I locate exceptions my VisionOS app?" and the answer provided by the general, untrained model.

```
{
 "input": "How do I locate exceptions my VisionOS app?",
 "output": "I'm sorry, but I couldn't find specific information on how to locate exceptions in a
VisionOS app. It's possible that the documentation doesn't cover this topic. I recommend checking
the official documentation or reaching out to the support team for assistance with locating
exceptions in your VisionOS app."
}
```

Figure 10. The answer of the general model on debugging an application

Furthermore, we have trained the model with the documentation of the used technology and asked again the same question. As can be seen in Figure 11, the chatbot provided a detailed and correct answer, helping the student obtain the desired information in just a fraction of the time.

```
{
 "input": "How do I locate exceptions my VisionOS app?",
 "output": "To locate exceptions in your VisionOS app, you can follow these steps:\n' +
    '\n' +
    '1. When your app experiences a crash, exception, or runtime issue, it can be challenging to
pinpoint the code causing the problem because the stack trace for the crash doesn't always point to
the line of code that causes the crash.\n' +
    '\n' +
```

```
    '2. Identify the problem characteristics based on the type of crash:\n' +
    '   - A crash that stops at main or highlights AppDelegate is frequently an Objective-C
exception.\n' +
    '   - A crash that is the result of a runtime issue also stops at main or highlights AppDelegate and
may have a message like: "Thread 8: EXC_BAD_INSTRUCTION (code=…)".\n' +
    '   - A crash that stops at an uncaught or unhandled Swift error displays a fatal error message and
indicates a Swift error.\n' +
    '\n' +
    '3. Add a breakpoint to your code in a location based on the identified problem characteristics.
When your app stops at the breakpoint, check the state of the code execution.\n' +
    '\n' +
    '4. To diagnose errors, attach the debugger, reproduce the error, and narrow down its root cause
by inspecting your variables at key points in your code while the app is running with breakpoints.\n'
+
    '\n' +
    '5. Configure a scheme's run action for debugging by using the Debug executable checkbox in
Info settings to automatically attach the debugger when the app uses the scheme.\n' +
    '\n' +
    '6. If you need to attach the debugger to a process that is already running, choose Debug →
Attach to Process and select your app's process from the list.\n' +
    '\n' +
    '7. Follow these steps to diagnose and resolve errors in your code, crashes, memory leaks, and
layout problems.\n' +
    '\n' +
    'By following these steps, you can effectively locate exceptions in your VisionOS app and debug
any issues that arise."
}
```

Figure 11. The answer of the trained model on debugging an application

Conclusions and future work

Firstly, our empirical evidence shows that using custom models with embeddings significantly improves the accuracy and contextual relevance of AI text processing. After carefully studying the current state of the art, the authors identified the main techniques, technologies and tool to develop the AI text processing models. One of the results consists in the validation of our premise that using embeddings, the model will gather a much better understanding of the semantic relationships, leading to a better reasoning.

Starting with embeddings, Retrieval Augmented Generation is used to further improve the results. LangChain was identified as the cutting-edge technology offering the premises for building advanced models for AI text processing. All these led to extending the knowledge base of Large Language Models with custom materials.

Further, our results are compared to the ones of the general model, which was not enriched or custom improved. In order to show its utility, our work was applied and tested in two domains, business and education, and both use cases present good results. There are some important pluses of the model described compared to the classical ones, such as providing a scalable and robust solution for complex AI text-processing tasks as well as having better performances.

There is significant room for model's improvement and development in the future. The integration of multiple chains and runnable sequences, as offered by LangChain, would improve the functionality of the application and allow for the handling of increasingly complex queries as well as the execution of more sophisticated information retrieval and processing tasks.

Furthermore, a GPT model that has been finely tuned and customized to a custom domain may produce replies that are far more accurate and relevant. In order to do this, the model will be provided with more information about the context of the user's requests, this kind of nuance and adjustment would improve the results and overall user experience.

The system's capacity to convert general questions into specific, actionable advice based on underlying organizational knowledge is highlighted by the illustrated real-world instances. In the future, in addition to the methods and techniques already used, we want to use new ones, primarily related to ML model training. However, the field of AI-powered text analysis has become so dynamic in the last couple of years that we expect new tools and techniques to quickly appear that can help improve the model.

References

Bonan Min, H. R. (2023). Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. *ACM Computing SurveysVolume 56(2) - https://doi.org/10.1145/3605943*, pp. 1-40.

Harsh Trivedi, N. B. (2023). *Questions, Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step*. Retrieved 2024, from ArXiv - Computer Science > Computation and Language: https://arxiv.org/abs/2212.10509

Izzidien, A. (2022). Word vector embeddings hold social ontological relations capable of reflecting meaningful fairness assessments. *AI & Society, https://doi.org/10.1007/s00146-021-01167-3*, 299–318.

Jacob Devlin, M.-W. C. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Association for Computational Linguistics.

Jeffrey Pennington, R. S. (2024). *GloVe: Global Vectors for Word Representation*. Retrieved 2024, from NLP Standford Edu: https://nlp.stanford.edu/projects/glove

Jiaoyan, C. P.-R.-R.-R.-R. (2021). OWL2Vec*: embedding of OWL ontologies. *Machine Learning 110*, 1813–1845.

Johnson, S. M. (2023). A detailed review on word embedding techniques with emphasis on word2vec. *Multimed Tools Appl. https://doi.org/10.1007/s11042-023-17007-z*.

LangChain. (2023). *Retrieval*. Retrieved 2024, from LangChain.com: https://js.langchain.com/docs/modules/data_connection

Lisowski, E. (2023). *Best AI Text Analysis Tools Comparison (2023)*. Retrieved 2024, from Addepto.com: https://addepto.com/blog/best-ai-text-analysis-tools-comparison-2023/

Madanchian, H. T. (2023). Artificial Intelligence and Sentiment Analysis: A Review in Competitive Research. *Computers, 12(2), 37; https://doi.org/10.3390/computers12020037*.

Nuthakki, D. R. (2023). *Chapter 2 – Natural Language Processing and Text Analytics: Techniques and Applications*. Retrieved 2024, from San International - Scientific Publications: https://submissions.nobelonline.in/chapter-2-natural-language-processing-and-text-analytics-techniques-and-applications/

Ray, P. P. (2023). ChatGPT A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope, ISSN 2667-3452. *Internet of Things and Cyber-Physical Systems*, 121-154.

Roumeliotis, K. I., & Tselikas, N. D. (2023). ChatGPT and Open-AI Models: A Preliminary Review. *Future Internet, 15(6), 192*.

SILVA, V. S., FREITAS, A., & HANDSCHUH, S. (2019). *On the semantic interpretability of artificial intelligence models.* Retrieved from ArXiv preprint: https://arxiv.org/abs/1907.04105

Wei Jin, A. O. (2010). BERT: a tool for behavioral regression testing. *FSE '10: Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering* (pp. 361-362). https://doi.org/10.1145/1882291.1882348.

Yang, B. L. (2023). Recent Progress on Text Summarisation Based on BERT and GPT. In: Jin, Z., Jiang, Y., Buchmann, R.A., Bi, Y., Ghiran, AM., Ma, W. (eds) Knowledge Science, Engineering and Management. KSEM 2023. *KSEM, Computer Science, vol 14120. Springer, Cham*.

Yunfan Gao, Y. X. (2023). *Retrieval-Augmented Generation for Large Language Models: A Survey*. Retrieved 2024, from Arxiv Preprint: https://arxiv.org/abs/2312.10997