APPLICATION OF OPENAI API FOR SEMANTIC CONTENT ANALYSIS IN INTELLIGENT E-LEARNING SYSTEMS

ЗАСТОСУВАННЯ OPENAI API ДЛЯ СЕМАНТИЧНОГО АНАЛІЗУ КОНТЕНТУ В ІНТЕЛЕКТУАЛЬНИХ СИСТЕМАХ ЕЛЕКТРОННОГО НАВЧАННЯ

by Mykola Rybak

Presented in Partial Fulfillment of the Requirements for the Degree

Master of Software Engineering

American University Kyiv
2024

APPROVED BY:

Sergiy Tytenko, Ph.D., Faculty Mentor

Abstract

The thesis aims to describe the practical aspects of using OpenAI API in intelligent e-learning systems. The proposed work contains three main parts.

The first part covers a background overview, theoretical aspects of a Large Language Model (LLM), transformer-based Natural Language Processing (NLP) architecture description, and use cases of different transformers. Introduces the term semantic unit and defines its main elements and connection with the capstone project.

The second part concerns the rationale of the chosen model's type for the capstone project and its description and implementation. This part covers requirements for the capstone project, C4 architecture diagrams, an overview of essential modules, libraries, and code examples of the implemented solution leveraging OpenAI API.

The final part demonstrates further research on architecture improvements for better semantic content analysis and interaction with intelligent e-learning systems, including a combination of encoder-decoder and embedding models in the solution.

Contents

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
ASR	Architecturally Significant Requirement
AWS	Amazon Web Services
BERT	Bidirectional Encoder Representations from Transformers
C4	Model name for visualizing software architecture
CI/CD	Continuous Integration and Continuous Delivery
GPT	Generative Pre-trained Transformer
LLM	Large Language Model
LSTM	Long Short-Term Memory Network
POC	Proof of Concept
POV	Point of View
RAG	Retrieval Augmented Generation
NER	Named Entity Recognition
NLP	Natural Language Processing
RNN	Recurrent Neural Network
SME	Subject Matter Expert
SPA	Single Page Application

Introduction

The emergence of new information technologies, especially ones in the field of Artificial Intelligence (AI) and Natural Language Processing (NLP) in particular, makes a significant impact on countless areas of our lives. The research *focuses on* applying the Large Language Model (LLM) for semantic content analysis in intelligent e-learning systems.

The educational field constantly seeks innovative methods to enhance the learning process. LLM can dramatically help improve learning and testing processes, benefiting everyone involved — from creating learning materials and curriculum to the overall student's educational experience. The traditional methods of preparing educational content often entail a substantial investment of time and effort from educators and subject matter experts (SMEs). Moreover, these conventional approaches may need more dynamic adaptability to cater to students' diverse learning curves and educational pathways [1]. In that case, instead of spending much time with SMEs on the initial data analysis and educational content preparation for a specific program or topic, such tasks can be automated and delegated to the LLM part. It will allow SMEs to focus on tasks requiring their expertise and attention instead of working on routine tasks. Such leveraging of LLM in the educational process should improve the general effectiveness of individual researchers, educational institutions, and particular tools like e-learning systems or their parts.

Based on the author’s comprehensive experience in software engineering and cloud technologies, we focus on the digital transformation of the existing e-learning system, the Semantic Portal [2], to *show concrete integration* with the LLM via OpenAI API. We use different libraries and tools, particularly the D3.js library for the visualization part, LangChain [3], and OpenAI API to interact with LLM for that task. The implemented application allows users to apply GPT-3 and GPT-4 LLMs provided by the OpenAI organization, demonstrating that the difference in final results depends on the complexity of the model involved in data processing. For continuous integration and delivery (CI/CD), we use the AWS ecosystem, applying AWS CodePipeline, AWS Codebuild, AWS CodeDeploy, and AWS Fargate as a serverless service to run the application. The codebase of the capstone project is publicly available on the Bitbucket repository.

Also, the paper *explains* a) the chosen libraries and LLMs used in the capstone project with code examples b) and a further vision of how the application architecture can be

extended and improved based on additional requirements for the application and different types of LLMs involved in the different stages of the data processing.

The *aims* of the research are:

a) to explain the purpose of different types of modern LLMs and their core capabilities;
b) to implement proof of concept to demonstrate the integration of the existing e-learning platform without AI capabilities with the LLM of choice in the context of data decomposition (semantic content analysis);
c) to demonstrate possible improvements in the solution architecture and directions for future work based on this research.

The thesis is an application of gained knowledge during education in disciplines SDT 514 “Natural Language Processing”, SDT 515 “The basics of machine learning”, SDT 509 “Solution Architecture”, SDT 505 “Data Visualization”, SDT 502 “Software Design and Architecture”, SDT 501 “Software engineering: Principles and Concepts”, GEN 500 “Professional English”.

Some of the research output was published in the article “Leveraging ChatGPT for Educational Text Analysis Aiming Assessment Generation” (2023) in the international scientific peer-reviewed journal “Modern Engineering and Innovative Technologies” [1].

The paper consists of 5 parts:

Introduction: introduces a brief overview of the topic, the author’s motivation, and the paper’s aims.

Chapter 1: explains background overview with current research on the topic, theoretical aspects and description of transformer-based models, and use cases of different transformers; introduces the term semantic unit and defines its main elements and connection with the capstone project.

Chapter 2 presents the rationale of the chosen model’s type for the capstone project, integration with the existing e-learning platform with an architectural description in the C4 model representation, an overview of modules, selected libraries, and code examples of the capstone project.

Chapter 3 describes further research on the improved architecture and system extension based on the previous chapters to demonstrate additional areas to investigate.

Conclusions: includes the thesis’s summary, conclusions, and areas for improvement.

Chapter 1. Foundations of Transformer-Based NLP: Exploring Large Language Models Applicability in E-learning Systems

1.1. History of Natural Language Processing and Large Language Models

Performing textual data analysis usually means applying different natural language processing techniques (NLP). There is no single definition or standard of NLP, and different researchers define it from different aspects:

1. A part of artificial intelligence (AI): “is a subfield of artificial intelligence that focuses on the interaction between computers and humans through language. The ultimate objective of NLP is to read, decipher, understand, and make sense of human language in a valuable way” [4].

2. A discipline: “is a discipline that combines the power of computers with the nuances of human language, allowing machines to read text, hear speech, interpret it, measure sentiment, and determine which parts are important” [4].

3. A software algorithm: “is any computer manipulation of natural language, including the understanding of the sentence essence to provide a relevant output” [5].

4. An interdisciplinary discipline: “is a subfield of artificial intelligence and computer science that focuses on the tokenization of data – the parsing of human language into its elemental pieces. By combining computational linguistics with statistical machine learning techniques and deep learning models, NLP enables computers to process human language in the form of text or voice data” [6].

Different terms are used regarding NLP depending on its applied field: speech recognition in electrical engineering, computational psycholinguistics in psychology, computational linguistics in linguistics, and natural language processing in computer science [7]. But the common part in all definitions is about the ability of a system of any kind to understand the natural human language of different types (textual or verbal representation) in terms of context, semantics, and tone.

1.1.1. Early History of NLP

Before diving into the technical aspects of modern NLP approaches, which are essential to explain why we chose a certain LLM for the capstone project, we summarized critical points of NLP history.

1940s-1950s: Theoretical foundations. Formation of the formal language theory field results from the appearance of the first model of a neuron (e.g., McCulloch-Pitts neuron). During that period, the first probabilistic algorithms for language processing were developed, and concepts from other disciplines were borrowed (e.g., the entropy concept from thermodynamics to measure the information content of a language) [7].

1950s-1960s: Rule-based systems. Focus on parsing and machine translations. The first Ph.D. thesis in Mechanical Translation was prepared at Harvard. During that period, many international conferences were held in the USA, East Germany, France, Japan, and the Soviet Union (e.g., a national conference in Chernivtsi, Ukraine, in 1960 with the participation of leading researchers in the field) [8].

1970s-1990s: Extensive usage of statistical methods. Development of speech recognition algorithms. The main paradigms during that period were stochastic, logic-based, natural language understanding, and discourse modeling [7]. During that period, research shifted towards statistical methods, leveraging probabilistic models for various NLP tasks [9]. Also, the first theoretical aspects of RNNs and LSTMs were introduced without practical implementation due to their complexity at that period [10].

Thus, the mentioned periods created a foundation for further NLP research, and starting from the 2000s, a new era of neural models began.

1.1.2. Modern History of NLP

There are three main periods in the modern history of NLP.

2000s: Evolving of machine learning models. During that period, a dramatic increase in publicly available data required new approaches to processing data and led to the development of supervised and supervised machine learning. The other important thing was hardware evolution in that period with increased computational capacity [7]. In 2001, introduced the first neural language model that could predict future words in a sentence using a feed-forward neural network [10].

2010s: Deep learning. Practical implementation of RNNs and LSTMs due to Ph.D. research on “Training recurrent neural networks” by Ilya Sutskever [11]. In 2013, introduced the Word2Vec library, which is used for translating words into word embeddings, an essential technique in the transformer-based paradigm [12].

Late 2010s-Present: Transformer-based models and LLMs. Introduction of Bidirectional Encoder Representations from Transformer (BERT) and Generative Pre-trained Transformer (GPT) models as a first implementation of pre-trained language models [13, 14].

The transformer-based approach is the most recent and modern paradigm in NLP. In the next section, we will analyze it in detail because the capstone project uses the implementation of the pre-trained transformer-based model: GPT model.

1.2. Foundations of Transformer-Based Models as latest paradigm in NLP

In the transformer-based paradigm, there are two important parts:

- ***Encoder***, which is responsible for reading textual data. It outputs a fixed-length hidden state of the encoder.
- ***Decoder,*** which generates a variable length output sequence. It takes a fixed-length state and produces a variable-length result.

1.2.1. Essence of Self-Attention Mechanism in Transformers

The primary concept of transformers is ***attention*** [13]. In 2018, pre-trained models, like T5, BERT, and GPT, were introduced based on that concept, particularly ***self-attention***. It is a specific type of attention that allows associating any token from input with any other tokens, finding the best match, and making a relationship between all tokens.

If we extrapolate to the real-life example of how different kinds of models work with tokens, it seems reasonable to represent that as:

- In sequential processing, the model reads tokens in sequence, meaning it doesn't t know the next context before reaching it. It mimics *usual human reading patterns* to read text from the beginning to the end.

- In self-attention processing, the model makes connections between all tokens, which allows the creation of context with tokens no matter where they are located: at the beginning or the end of the input. It increases the overall "understanding" of the data. It is similar to the *"skimming and scanning" technique* on high-level comparison, which is popular in language exams like IELTS to increase the overall understanding of the context of long texts. However, the self-attention mechanism is

more comprehensive and deeply explains the context, while the “skimming and scanning” technique is about general understanding.

As mentioned, before transformer-based architecture, textual data was processed sequentially, which had issues understanding long sequences of tokens. The comparison of sequential and transformer-based self-attention processing is demonstrated in Table 1.1.

	Sequential processing (RNN/LSTM)	Transformer-based self-attention processing (GPT, BERT)
Processing nature	Sequential (the context resolves based on the previous tokens)	Parallel (create association between all tokens regardless of the sequence due to self-attention mechanism)
Level of context understanding	Low, as there is a *limitation to handle long-term dependencies*, so if tokens appear much earlier or later from the target token the context will be lost.	High, as it supports short-term (remember information from previous tokens) and long-term dependencies
Vanishing gradient problem	Partially solved in LSTM models, but the issue is stills may occur in case of long sequence of tokens, which decrease the context understanding	Solved by applying positional encoding technique, which guarantee that the model preserve tokens’ order, which is vital for context understanding

Table 1.1. Comparison of sequential and transformer-based self-attention tokens processing.

1.2.2. Transformer Types and Applicability Use Cases

As mentioned earlier, the transformer-based architecture includes encoder and decoder parts, and different models implement mostly one or combine both parts. It is shown in Figure 1.1.

The aforementioned means a model used in the solution should be chosen based on the main purpose of the transformer type underlying such a model. Thus, the main use cases for transformers-based models are:

- The encoder type is designed to process input data and understand the context. Use cases: text classification, information retrieval, named entity recognition (NER).

- The decoder type is designed to generate new content or transform the input into another format. Use cases: text generation, machine translations.
- The encoder-decoder type is designed to understand the context of input and generate output based on the encoder output. Use cases: text summarization, speech recognition.

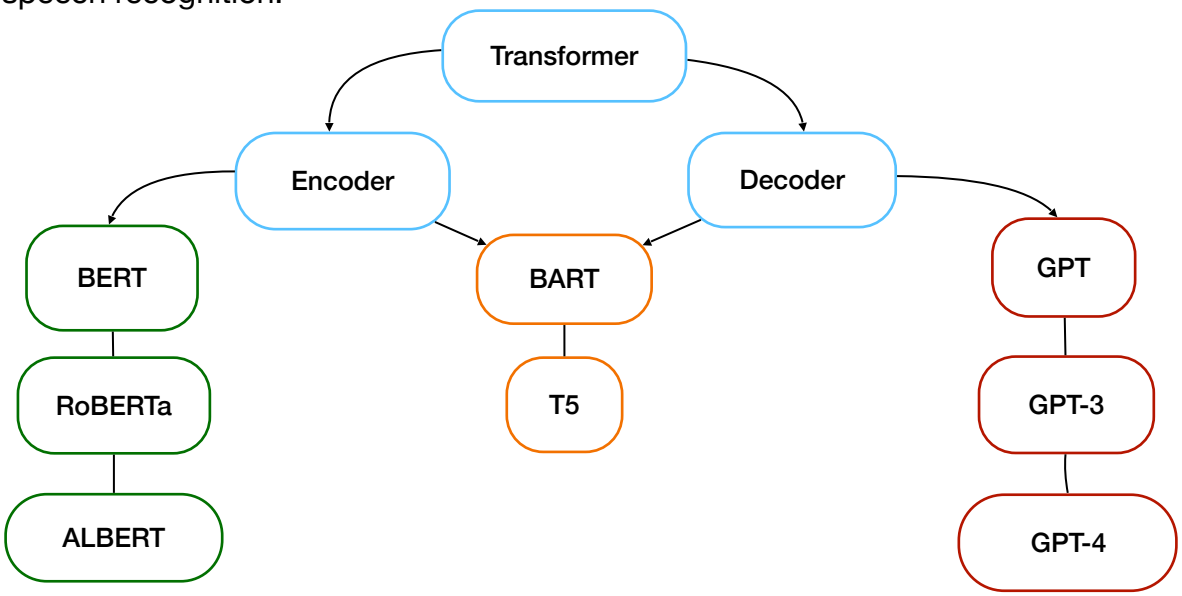

Figure 1.1. Transformer-based models.

Having a basic theoretical understanding of transformers, we are ready to move forward to discuss their applicability in e-learning systems.

1.3. Application of Large Language Models in E-learning Systems

Based on the available data, the interest in ChatGPT, one of the most popular publicly available tools for using the GPT model provided by the OpenAI organization, particularly in education, has risen significantly. Following the recent statistics, the estimated number of visits to ChatGPT is reaching 2 billion points, and an estimated 80% of major Fortune 500 companies are expected to utilize ChatGPT in their business operations permanently [15]. This says that AI technologies, particularly LLMs, are becoming inevitable in all areas of our lives, including education.

1.3.1. AI Tools in Education: Current State

Different AI tools are widely implemented in educational processes, and the growth of AI technologies will impact the future of education. The article [16] stated that

integrating OpenAI's GPT-3 model into educational products led to a 25% improvement in student engagement and learning outcomes. GPT-3 facilitates personalized, efficient, and effective learning experiences for students. The United States, India, and China are the leading adopters of GPT-3 in education, accounting for 60% of the global adoption [16].

Therefore, various researchers investigate possibilities of leveraging IA in the educational domain. Among the different resources analyzed, we defined the following areas of application of the AI solution in education.

1.3.2. Concept Maps and Automated Assessments

One of the most popular representations of information is concept maps, which show information in a hierarchical structure. It can be used in different areas, from the tool to evaluate student's knowledge in some topics to information representation for content exploration.

In the article [17], the authors introduced the Artificial Intelligence-based Student Learning Evaluation tool (AISLE), which is based on concept maps to evaluate the understanding of different topics. The main idea of the proposed concept is to allow students to describe the educational content through a concept map. Each topic starts from some primary node and then expands to multiple levels. Roughly said, the level of hierarchy and number of concepts students identified explains how deep the understanding of a particular topic is.

Others pay attention to the problem of concept map creation overall from unstructured data [18, 19, 20]. The authors investigate how to decompose the data to retrieve its essential parts for further analysis. For instance, in article [18], the author suggested dividing educational material into smaller and more manageable components called Concepts and Theses—that breakdown the author used to support a more organized and coherent assessment generation process. Based on different schemas for assessment generation, it can be created around concepts or theses. These schemas play a crucial role in generating alternative answers for the tests. They are sourced from correct and incorrect answers and serve as the backbone for creating diverse options. This not only aids in crafting dynamic and maintainable tests but also provides a system to inform users about their knowledge gaps in specific areas or content portions. The outlined hierarchical relation of Theses and Concepts simplifies the maintenance of tests and ensures their alignment with the educational content, thereby fostering a more effective and meaningful assessment process. The creation of concept maps and semantic content analysis can be automated using LLM and supervised by the SME on the final step of the result verification.

1.3.3. AI Assistant

Transformers, particularly decoder-based LLMs, are designed to generate content and ideally fit the assistant role. In the article “Teaching CS50 with AI” [21], the authors described the following areas of usage:

- code explanation;
- code style evaluation;
- chatbot to answer course-related inquiries.

The important thing that the authors raised is the issue of academic integrity [21, 22], which requires having a wrapper above the main LLM, instructing it to prevent direct answers but giving a hint so students will find the solution by themselves. In the case of the GPT model, it is solved by providing the system prompt, the special instructions, and the rules that LLM initially consumes before the actual prompt from the user. In Figure 1.2. provided an example of such flow.

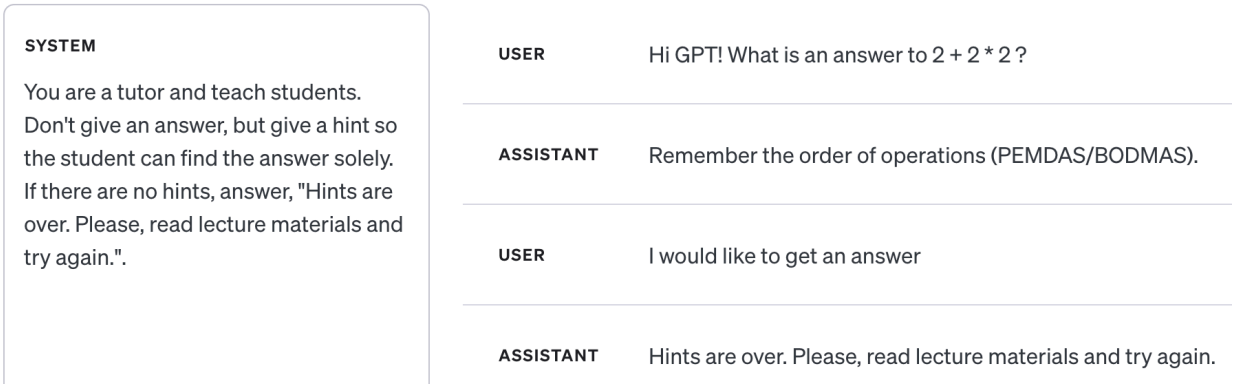

Figure 1.2. Example of system message usage on OpenAI playground.

The peculiarities of prompt engineering will be discussed in the scope of the capstone project in Chapter 2.

1.3.4. Content Generation

Decoder-based models, like GPT, can create new educational content, curriculum, or assignments. The user should provide LLM with a prompt with a blueprint for the desired content, and then, LLM will expand it with actual content based on the existing knowledge base. Connecting external data sources to enrich data is also possible; for example, the LangChain library provides the Agent module [3, 30]. More detail on that feature will be discussed in Chapter 3.

In Figure 1.3. demonstrates the high-level flow for assessment generation.

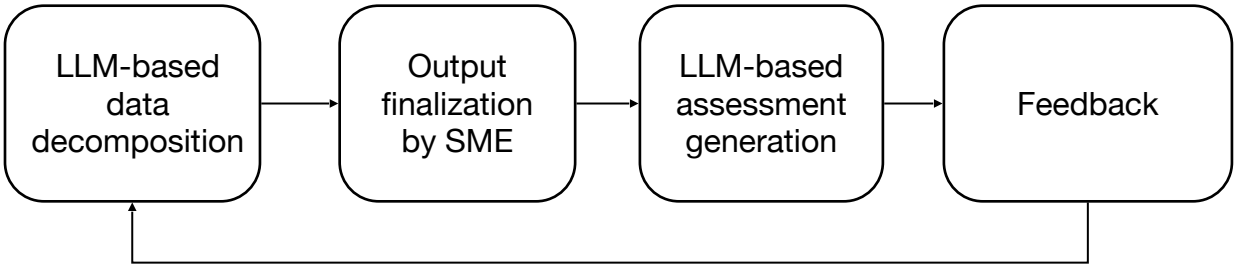

Figure 1.3. The high-level flow of LLM-based assessment generation.

LLM doesn't work fully independently in the mentioned flow; a human (SME) always validates the output. The other important thing is a feedback loop to interact with LLM and teach it in case of invalid results.

1.4. Semantic Content Analysis: Theoretical Aspects

As mentioned before in Section 1.3.2. semantic content analysis is one of the aspects of e-learning systems that can be delegated to LLM. That task is the aim of the Capstone project, so it is important to describe what we mean by semantic content analysis before moving into the application overview in Chapter 2.

Semantic content analysis is a process to understand the meaning, interpretation and relationships between words, phrases, and sentences in a given context [32].

To better understand the content, the common approach is to define three core elements of the text: a) Topic, b) Main ideas, and c) Supportive ideas. We propose using the category ***Semantic unit*** to refer to these elements. The topic is a general theme of the whole content and the highest element in the semantic unit's hierarchy, which contains the Main ideas. The main idea describes some aspects of the topic, and their collection defines the Topic. The Supportive idea is the lowest unit in the semantic unit's hierarchy, which argues the main idea.

The aforementioned approach we use in the Capstone project, but with a different terminology. The topic is represented by *a* ***Branch*** from the Semantic Portal e-learning system, which represents some educational content around that branch/topic. The next semantic unit is a ***Concept***, which refers to the Main idea. The branch may have multiple concepts that describe a topic. Each Concept *should have at least one* ***Theses***, a Supportive idea that describes different aspects of the Concept. Also, it is important to mention that the Concept may have nested concepts if the parent concept can be split into smaller units.

The Theses *may have* different examples, and we introduce a new semantic unit called ***Example***, which different types, such as HTML, code with the name of the programming language, link, etc, can represent. The selection of the example type is on the LLM side, and the application user can alter it later (see Section 2.4.3). The purpose of the LLM in the scope of the Capstone project is to define the Concepts, Theses, and Examples. The system's user provides the Branch into LLM as an argument, so LLM doesn't need to define it.

Therefore, the final semantic unit hierarchy for the semantic content analysis in the Capstone project is represented in Figure 1.4.

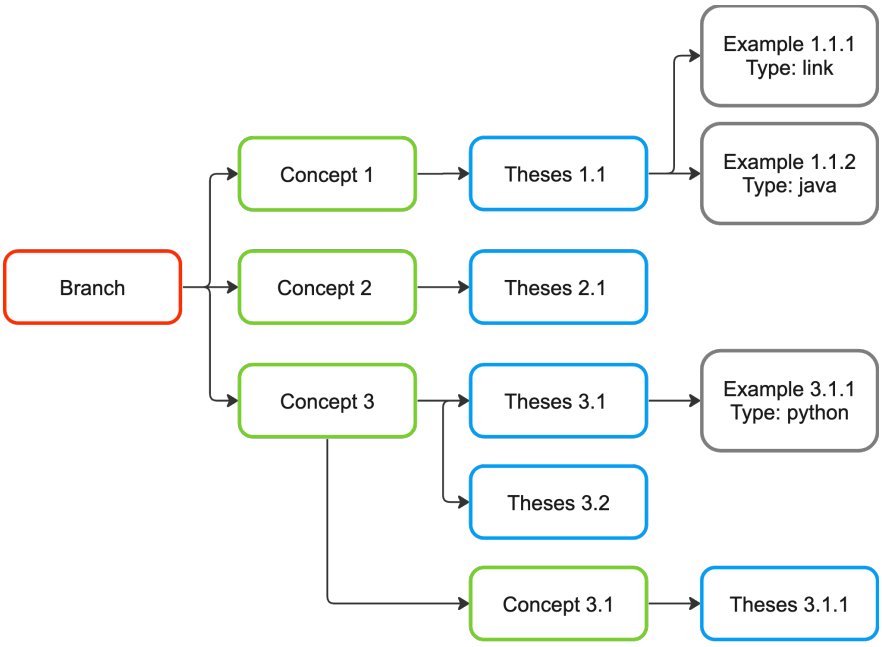

Figure 1.4. The semantic unit hierarchy for semantic content analysis in the Capstone project.

The practical usage of the mentioned approach to decompose a text into different semantic units is described in Chapter 2, specifically in Section 2.4.1. in the scope of the implementation features of prompt engineering in the project.

1.5. Summary of Chapter 1

Modern NLP solutions use transformer-based architecture. There are three types of transformers: 1) encoder, 2) decoder, and 3) encoder-decoder. The encoder is focused on consuming the input data to understand the context and is used mostly for classification or NER tasks. The decoder focuses on text generation and is helpful for

content creation tasks, like translating input data into another format. An encoder-decoder transformer is a combination of both types.

The core concept of transformers is a self-attention mechanism. It allows for better context understanding due to the association between all tokens and defining better matches between them.

We defined three main areas of LLM applicability in e-learning systems: 1) concept map creation and automated assessment tasks, 2) as AI assistant, and 3) content generation tasks.

Mentioned that semantic content analysis can be delegated to the LLM side and supervised by the SME in the final processing stage. Introduced a term semantic unit and highlighted its components from the capstone project perspective: Branch, Concept, Theses, Example. Narrowed the purpose of the LLM in the scope of the Capstone project as a definition of Concepts, Theses, and Examples during semantic content analysis.

Chapter 2. Text Decomposer: Application for Semantic Analysis of Learning Content

2.1. Application Overview

Innovative solutions in the educational field are essential to accelerate its progress and increase its efficiency. The capstone project, AI Text Decomposer, is designed to add AI capabilities to the existing e-learning system for semantic content analysis.

As an e-learning system, we selected the Semantic Portal, which aims to help students and professionals learn quickly via the modern ontology-oriented interface [2].

2.1.1. Business Goals of AI Text Decomposer

The current stage of the evolution of the Semantic Portal requires the manual decomposition of the educational content by SMEs to represent data as a graph. The capstone project aims to power the Semantic Portal with an AI solution to decompose data via a transformer model of NLP and eventually reduce SMEs' time on that task.

Business drivers:

- Business processes optimization in the context of data decomposition.
- Time reduction to prepare new educational content in graph representation.
- Positive feedback growth from students and professionals.

Business goals:

- Increase the number of new users in the Semantic Portal.
- Decrease time for business processes to decompose educational content.

Business objectives:

- Integrate LLM-powered companion application (POC) for Semantic Portal to perform semantic content analysis and export the final result in JSON format.

2.1.2. Project Stakeholders

Name	Role	Group	Contacts	Architectural Work Request Reason
Mykola Rybak	Solution Architect	Project Team	mykola.rybak@auk.edu.ua	Improvements, new features
Sergiy Tytenko	Product Owner	Project Team	sergiy.tytenko@auk.edu.ua	Improvements, new features

Table 2.1. Stakeholder list of capstone project.

2.1.3. Essential Project Resources

For the project purpose, the following resources were used:

- Bitbucket: https://bitbucket.org/mykolarybak/text-decomposer/src/develop/
- AWS CodePipeline: https://aws.amazon.com/codepipeline/
- Semantic Portal API: http://semantic-portal.net/
- LangChain library: https://www.langchain.com/
- OpenAI API: https://platform.openai.com/

2.2. Requirements for AI Text Decomposer

The requirements of the capstone project, or also referring to architecturally significant requirements (ASR), are divided into three types:

- Functional, which represents application features.
- Quality attributes, also known as non-functional requirements, represent requirements for the system overall, like scalability, security, maintainability, etc.
- Constraints, which describe the system's limitations.

The project ASRs are represented in Table 2.3.

	ASR	Rationale	Type	Priority
1	Authentication via Semantic Portal	To work with Semantic Portal data the capstone project should proxy authentication to Semantic Portal	Functional	H
2	Decompose educational content from Semantic Portal via LLM	Initial decomposition process should be done using LLM to reduce interaction with SME	Functional	H
3	Modify the LLM's result in-app	Application should allow user to modify the result from LLM in application before exporting it	Functional	H
4	Visualize result as a graph	Application should visualize result in a graph for better assessment of decomposition quality on high-level (number of Concepts)	Functional	H
5	Show content with decomposed data per each defined Concept on a separate sidebar	For better usability the system should show a Concept's node content on a separate sidebar for better assessment of decomposition quality on low-level (number of Theses and examples to them)	Functional	M
6	Show initial educational content	For better usability the capstone application should allow user to read the initial educational content in application without switching to Semantic Portal	Functional	L
7	Export	Final result should be exported in JSON format for further usage outside the application	Functional	H
8	Scalability	System should be flexible for scaling in case of increasing usage rate, and open for changes	Quality attribute	H
9	Reliability	System should have built-in mechanism to restart in case of failure	Quality attribute	H
10	Data limits	LLM used for the solution has restrictions regarding input / output token size, which should be considered	Constraints	H

Table 2.3. ASR list.

The capstone project is built with the following assumptions:

- Semantic Portal is available for API calls.
- The user for Semantic Portal is created with an editor role and is active.

- Max tokens for the prompt processing are expected to be less than 4096 tokens, a limitation for a GPT model by OpenAI.

2.3. Target Solution Architecture of the AI Text Decomposer

This section includes an architectural framework for the “AI Text Decomposer" capstone project, detailing how the system is structured and interacts with external systems to meet the project's objectives and requirements.

2.3.1. Identified Risks of the Project

The application relies on OpenAI API and uses two LLMs by user’s selection:

- GPT-3;
- GPT-4.

The usage of the mentioned LLMs is paid, and costs vary depending on the API usage rate (input/output token size). For demonstration purposes of the capstone project, the application account has enough funds to maintain interaction with LLMs.

In Table 2.6. demonstrated pricing schema for GPT-3 and GPT-4 models.

Model	Input	Output
gpt-3.5-turbo-1106	$0.0010 / 1K tokens	$0.0020 / 1K tokens
gpt-4-1106-preview	$0.01 / 1K tokens	$0.03 / 1K tokens

Table 2.6. Pricing schema for GPT-3 and GPT-4 models [23].

From the above table, the GPT-4 model is more expensive than the GPT-3 model; on the other hand, the quality of the result is potentially higher due to the more complex training of the model.

In case of funds insufficiency, the application will not decompose the educational content.

2.3.2. Dependency Selection Rationale for the AI Text Decomposer

To interact with LLM in the capstone project, we ***use the LangChain library*** as a "flexible abstractions and AI-first toolkit" [3]. It allows using any LLMs via interface wrappers, making it easy to replace one LLM with another without breaking changes in the application.

We decided to ***use the GPT model***, a decoder-type transformer, for the project's purpose. The capstone project's task has two parts:

- Categorize data to retrieve Concepts, Theses, and their examples.
- Prepare output in JSON format.

As mentioned in Section 1.2.2., the context understanding and categorization task is mainly responsibility for encoder type of transformers (BERT) or encoder-decoder types (T5, BART). Although GPT (3,4) is mainly a decoder model, its transformer architecture allows for some hybrid functionality. This architecture enables it to understand the context of the input text before generating an output, giving it some capabilities of the encoder model. This understanding allows it to perform tasks that require a degree of classification or categorization.

GPT models by OpenAI require minimal engineering prompting to achieve the desired result. Also, OpenAI API gives a convenient way to interact with LLM, which convinced us to use the GPT model.

For CI/CD purposes, it was decided to ***use the cloud-based infrastructure AWS*** and its ***serverless service Fargate*** to run the application's docker containers.

The backend part is implemented in ***Python*** as the main language for any AI-based applications, particularly the LangChain library.

The ***Semantic Portal*** was selected due to the public accessibility to API, which the capstone project consumes for processing.

2.3.3. High-Level Solution Structure of AI Text Decomposer (C4)

Below are architecture diagrams following the C4 model guidance.

2.3.3.1. System Context Diagram

The System Context provides a high-level overview of the software system, illustrating its interactions with external entities such as users and other systems.

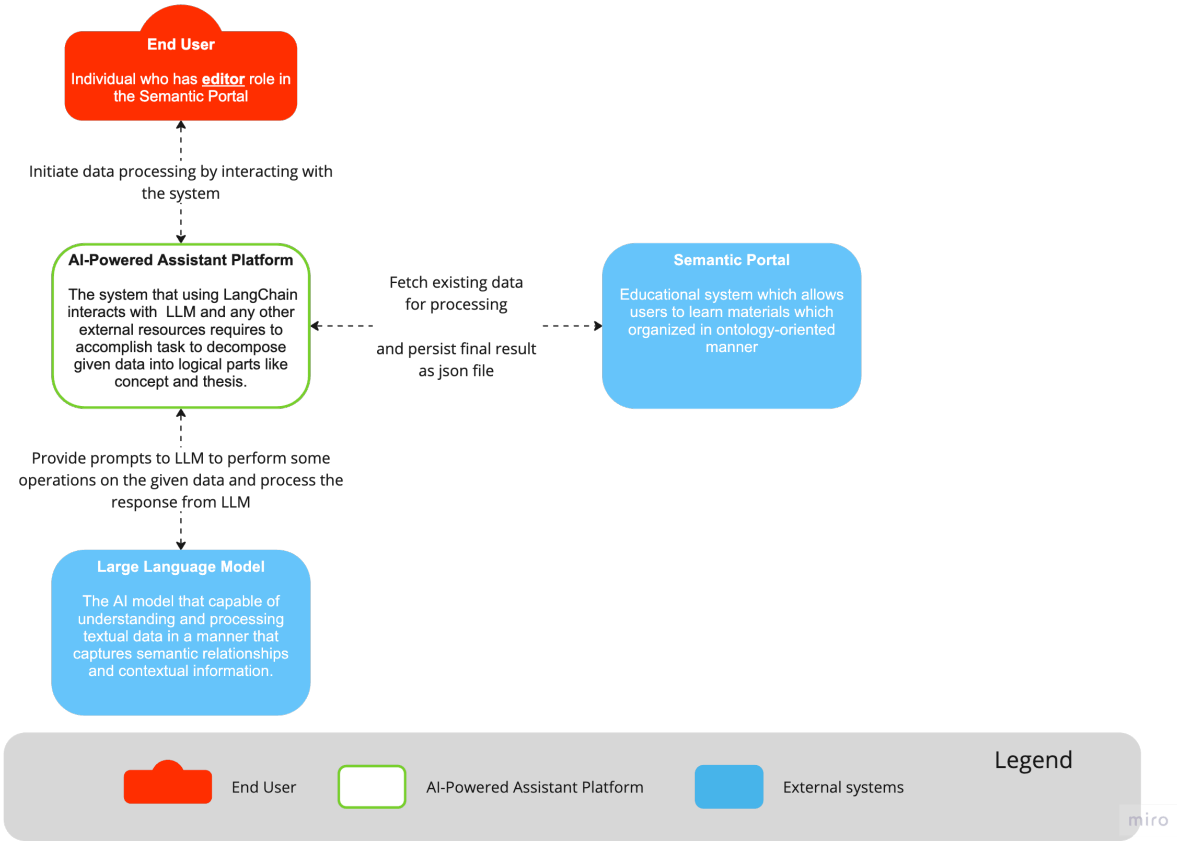

Figure 2.1. System context diagram of AI Text Decomposer.

AI Text Decomposer is a companion application for the Semantic Portal, which receives educational content from the Semantic Portal and, using the user's selected options prepares a prompt to interact with LLM via REST call and then return the result to the user for further work with it on the UI side. A more detailed explanation of the responsibilities of different parts of the application is explained in the Container Diagram.

2.3.3.2. Container Diagram

The Container Diagram below depicts the architecture's high-level shape and how responsibilities are distributed across it.

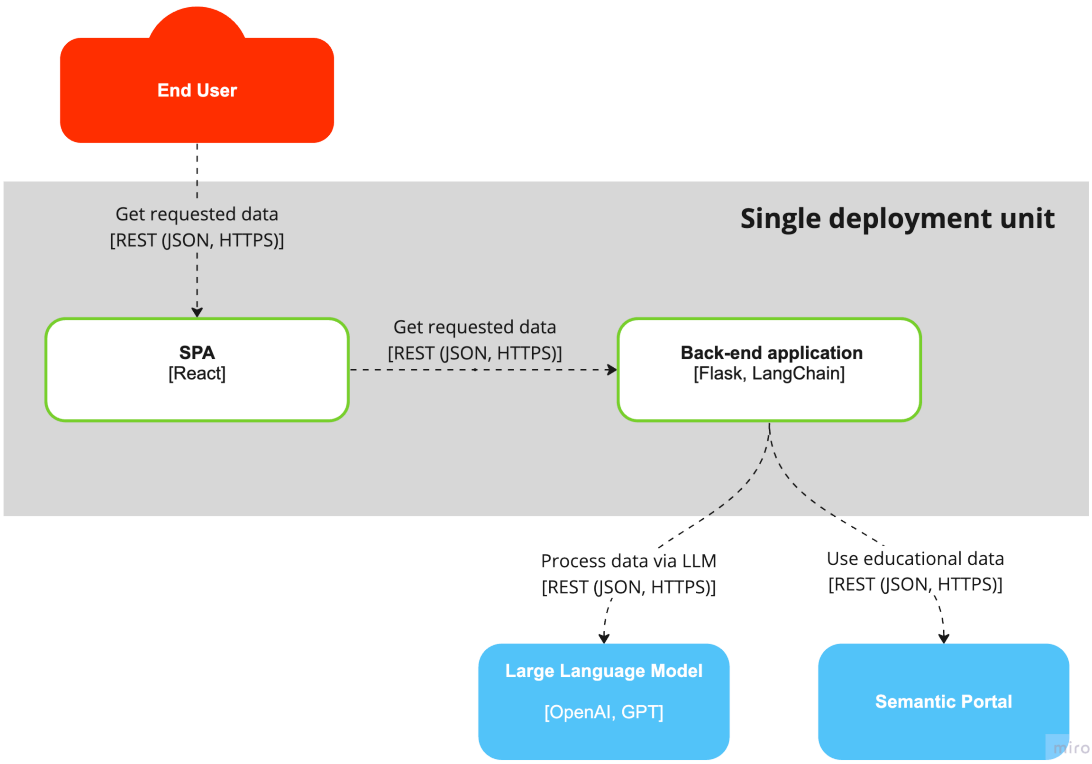

Figure 2.2. Container diagram of AI Text Decomposer.

The frontend and backend parts of the application are a part of a single deployment unit. It means they are in the same repository and start up as a single unit. It was done to simplify the deployment process and reduce additional settings for service discovery.

Description	Application to interact with the Semantic Portal and OpenAI API. It exposes all required API for external usage
Technology stack	Python, Flask, LangChain, beautifulsoup4, pydantic
Related components	SPA
Covered functional requirements	ASR 1, ASR 2

Table 2.7. Backend application technical details.

Description	Application to interact with user and backend, allows to modify LLM output on UI side, shows visualization and summarized contents
Technology stack	JavaScript, React, D3.js, @pgrabovets/json-view
Related components	Backend application
Covered functional requirements	ASR 3, ASR 4, ASR 5, ASR 6, ASR 7

Table 2.8. SPA technical details.

2.3.3.3. Component Diagram

The Component Diagram is intended to decompose each container further to identify the major structural building blocks and their interactions.

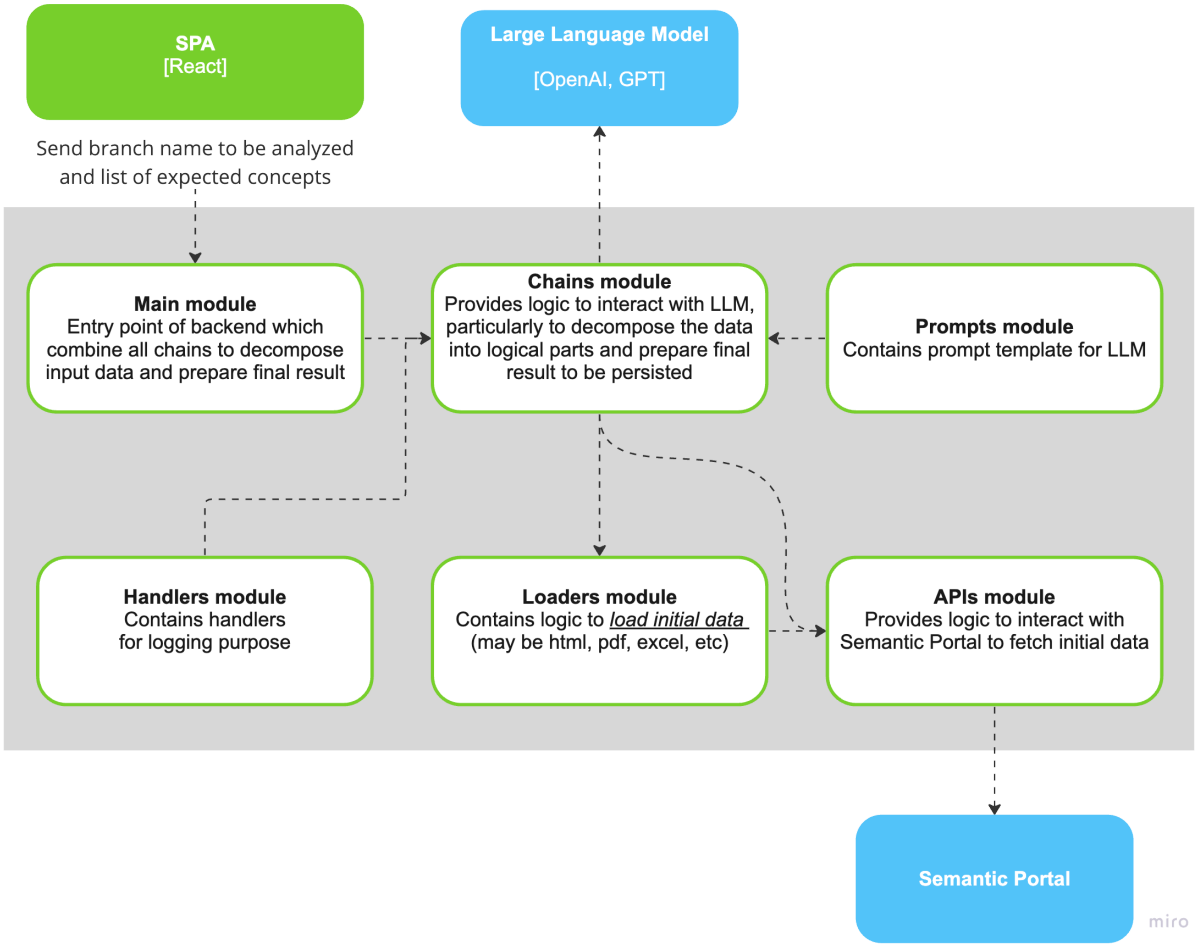

Figure 2.3. Backend component diagram of AI Text Decomposer.

2.3.3.4. Deployment Diagram

The deployment diagram focuses on the CI/CD flow demonstration.

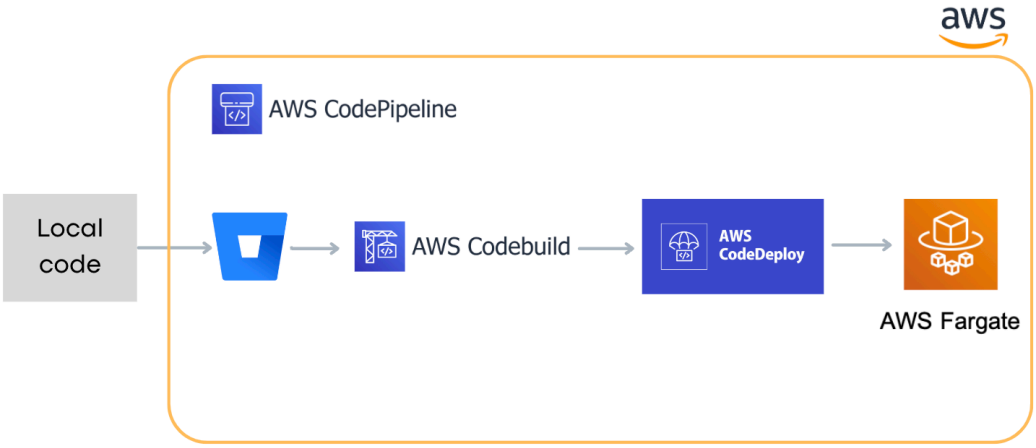

Figure 2.4. Deployment diagram of AI Text Decomposer.

CI/CD process is handled by the AWS platform, precisely by the AWS CodePipeline. Each push into the develop branch of the project on the Bitbucket repository triggers a build on the AWS Codebuild and auto-deployment, in case of a successful build, on Fargate (Figure 2.5). Such flow allows us to accomplish functional requirements ASR 8 and ASR 9, as the application's life-cycle and autoscaling (upscale/downscale) are managed by the AWS platform.

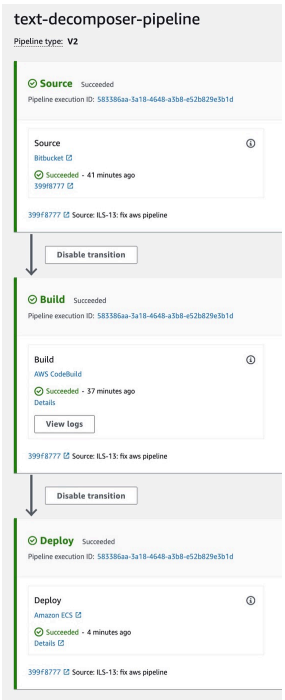

Figure 2.5. AWS code pipeline deployment flow of AI Text Decomposer.

2.4. Analysis of Implementation Features of AI Text Decomposer

We implemented project-specific logic with appropriate prompt engineering and features like data streaming, in-app editing, and visualization to achieve the capstone project objectives. In the following sections, we will describe key peculiarities of implementation.

2.4.1. Prompt Engineering Features

A prompt is key in communication with LLM to perform some actions. Incorrect or ambiguous prompts lead to unexpected responses from LLM. There are six strategies for better prompt engineering defined in OpenAI documentation [24]:

1. Writing clear instructions.
2. Providing a reference text to instruct LLM.
3. Splitting complex tasks into simpler subtasks.
4. Giving the model time to “think”.
5. Using external tools to handle tasks that the LLM cannot handle well.
6. Reviewing changes periodically.

Figure 2.6. is a prompt we designed for a capstone project. For a better “understanding” of the prompt by the GPT model, we defined different paragraphs:

- Definitions. It is a block where we define the main categories of our task: Concepts, Theses, and examples.
- Instructions. It is a block with a list of actions we expect from the LLM.
- Format. Is a definition in which format we are expecting the output sequence.
- Default block. The latest paragraph contains the default answer if output cannot be generated. It can happen if the input sequence is too small to perform semantic content analysis by LLM or if there is a networking issue with OpenAI API (e.g., there is insufficient balance for the application token, and LLM doesn’t respond due to this).

In prompt uses special placeholders wrapped by opening and closing curly brackets, `{}`.

Some are defined based on the user selection, like `{topic_output}` and `{expected_concepts_prompt}` and defined at runtime as a variable; others, like `{output_format}`, resolve immediately because that value is static. The output format has a specific representation. For its definition, we used the pydantic python package, and in code, it represents a class of objects, which we expect to get as a result (Figure 2.7).

```
### DEFINITIONS ##
CONCEPT is core ideas or patterns identified within the GIVEN RESOURCE.
THESES is an explanation of a particular concept. It refers only to one concept.
Every Concept MAY contain multiple theses and include other concepts, children.
Theses may contain examples as html snippets, code snippets, etc.
### DEFINITIONS ##

### INSTRUCTIONS ###
Topic: {topic_name}.
You MUST categorize from GIVEN RESOURCE Concepts and Theses. {expected_concepts_prompt}
Concept SHOULD be flexible to accommodate NESTED Concepts if necessary.
Each concept SHOULD has AT LEAST one theses!
GENERATE your own examples for theses if it is absent in GIVEN RESOURCE.
Add example in field "examples".
All code examples put into "examples", NOT in "description".
For code examples prettify the code for better representation following google styles.
Try to put a correct code type, like java, javascript, etc.
SHOULD USE special characters like \\n, \\t for better readability!!!
Set id as null.
More nested concepts identified the better result is.
### INSTRUCTIONS ###

### FORMAT ###
{output_format}
### FORMAT ###

DON'T reply the FORMAT schema itself, if you don't have result from the GIVEN RESOURCE.
In case if answer cannot be generated send "Sorry, provided information is not enough, check input data. Thank you!".
```

Figure 2.6. Prompt to perform semantic content analysis.

```python
from pydantic import BaseModel, Field
from typing import List, Optional


1 usage  Mykola Rybak
class ThesesExample(BaseModel):
    type: str = Field(description="Type of example: javascript, html, bash, relevant link etc.")
    example: str = Field(description="Example of the given type")


1 usage  Mykola Rybak
class Theses(BaseModel):
    id: str = None
    type: str = "THESES"
    description: str
    examples: List[ThesesExample]


5 usages  Mykola Rybak
class Concept(BaseModel):
    id: str = None
    type: str = "CONCEPT"
    name: str
    theses: Optional[List[Theses]] = None
    children: Optional[List['Concept']] = None


Concept.update_forward_refs()
```

Figure 2.7. Output format schema definition with pydantic package.

However, the Pydantic Python package is not an LLM-specific package. To translate the Pydantic object into the prompt-friendly format, there is a special wrapper from LangChain - `PydanticOutputParser`. For its usage, we need to wrap the pydantic object by that class, like:

```
PydanticOutputParser(pydantic_object=Concept)
```

After this, the available method `get_format_instructions()` to prepare the schema for the prompt.

The final definition of the prompt template is shown in Figure 2.8.

```
decompose_prompt = ChatPromptTemplate(
    messages=[
        HumanMessagePromptTemplate.from_template(decompose_template),
    ],
    partial_variables={
        "output_format": concept_model.get_format_instructions()
    },
    input_variables=["expected_concepts_prompt", "topic_name"]
)
```

Figure 2.8. Prompt template definition in LangChain.

, where partial_variables and input_variables hold placeholder definitions, and the message accepts the prompt from Figure 2.6.

Before proceeding, it is worth mentioning that LangChain defines three types of messages:

- System Message: This is an initial instruction that LLM sees before starting any actions with the prompt.
- Human Message. This is a user's / application's defined prompt.
- AI Message. This is a response from LLM.

Under the hood, LangChain replaces all variables with actual values before interacting with LLM and adds additional system messages, as demonstrated in Figure 2.9.

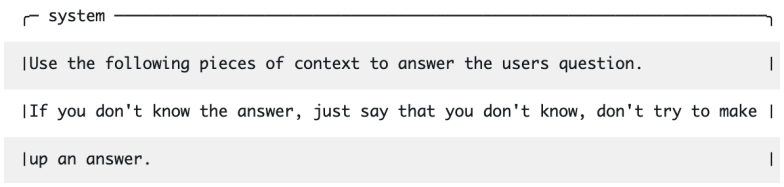

Figure 2.9. A system message that is added automatically by LangChain.

After that system message, LangChain adds the application-defined prompt, Human Message.

Thus, LangChain works like a useful bridge to easily communicate with LLP, adding additional processing to prompts and enriching them with system messages if needed.

2.4.2. Data Streaming Implementation Features

As was mentioned earlier, one of the strategies for better prompting is to give LLM time to process the prompt. But it may take some time, and to make a user-friendly application, it is important to send users the available chunks of data during such processing so it will be possible to check data in run time as soon as it is available from LLM. OpenAI API supports streaming operation, but LangChain, by default, doesn't process streaming data as expected and is waiting for the whole data by default. To fix that issue there some adjustments are required in chain settings.

Firstly, when defining an OpenAI client, we should enable streaming:

```
ParametrizedChatOpenAI(
        model_name=llm_model, streaming=True, callbacks=[handler]
)
```

This will tell OpenAI API to stream data. After this, we need to instruct LangChain to process streaming data. For this aim, we should define a custom chain. This contains three steps:

1. Define the streaming handler as demonstrated in Figure 2.10.

```python
class StreamingHandler(BaseCallbackHandler):
    Mykola Rybak
    def __init__(self, queue):
        self.queue = queue
        self.streaming_run_ids = set()

    Mykola Rybak
    def on_chat_model_start(self, serialized, messages, run_id, **kwargs):
        if serialized["kwargs"]["streaming"]:
            self.streaming_run_ids.add(run_id)

    Mykola Rybak
    def on_llm_new_token(self, token, **kwargs):
        self.queue.put(token)

    Mykola Rybak
    def on_llm_end(self, response, run_id, **kwargs):
        if run_id in self.streaming_run_ids:
            self.queue.put(None)
            self.streaming_run_ids.remove(run_id)

    Mykola Rybak
    def on_llm_error(self, error, **kwargs):
        self.queue.put(None)
```

Figure 2.10. A streaming handler definition in LangChain.

2. Define a custom streamable chain as demonstrated in Figure 2.11, where we use the handler from the previous step:

```python
class StreamableChain:
    Mykola Rybak
    def stream(self, input):
        queue = Queue()
        handler = StreamingHandler(queue)

        Mykola Rybak
        def task(app_context):
            app_context.push()
            self(input, callbacks=[handler])

        Thread(target=task, args=[current_app.app_context()]).start()

        while True:
            token = queue.get()
            if token is None:
                break
            yield token
```

Figure 2.11. A streamable chain definition in LangChain.

3. Define custom retrieval chain extending streamable chain from the previous step:

```
class StreamingRetrievalQAChain(StreamableChain, RetrievalQA):
    pass
```

4. Create a new chain based on the class from the previous step:

```
def build_content_decomposition_chain(llm, retriever):
    return StreamingRetrievalQAChain.from_chain_type(
        llm=llm,
        retriever=retriever,
        chain_type="stuff"
    )
```

Figure 2.12. A streamable chain creation in LangChain.

In the setting for the chain, we used chain type “stuff.” It means that we send initial data for processing as it is without any additional processing with that data. That type is the most cost-effective compared with “map-reduce” or “refine” strategies, as it requires only one query to process by LLM.

5. Finally, use the stream method from the chain to start streaming processing:

```
return build_content_decomposition_chain(llm=chat, retriever=retriever).stream(
    decompose_prompt.format(expected_concepts_prompt=expected_concepts_prompt, topic_name=branch))
```

Figure 2.13. A streamable chain execution in LangChain.

After that step, we achieve streaming processing, and the user will get data in chunks as soon as possible when they are available from LLM.

2.4.3. In-App Editing Feature

As demonstrated in Figure 1.3. in Chapter 1, regardless of the output from LLM, there still should be the possibility to check the result and correct it if needed. For that purpose, in the “AI Text Decomposer”, we added the feature to update the response from LLM if it is in the JSON format. If data from LLM is not in JSON format, the processing cannot be performed properly, and we get a plain string with an error message in such cases.

To transform a javascript object into an editable JSON object on the UI side, we used the library @pgrabovets/JSON-view, which allows us to update any information in the JSON alongside adding/deleting nodes. Also, that library has a built-in feature for auto-formatting with styles similar to the ones we have in code editors (Figure 2.14).

```
<ReactJson
    src={jsonResponse}
    name={false}
    iconStyle="circle"
    displayDataTypes={false}
    displayObjectSize={true}
    onEdit={onAnyUpdate}
    onAdd={onAnyUpdate}
    onDelete={onAnyUpdate}
    theme="tomorrow"
/>
```

Figure 2.14. Component definition to edit JSON object on UI.

In Figure 2.14, the src variable is a JSON object, and onEdit/onAdd/onDelete are callbacks to perform modifiable operations with the object.

That implementation brings users the possibility to manipulate the resulting object before exporting the output. In the next section, we will show the client side of the application in the scope of the User Guide.

2.5. AI Text Decomposer User Guide

The capstone project is designed as a companion application to the existing solution, the Semantic Portal. To use API for Semantic Portal in the AI Text Decomposer, we raised the same credentials to log into the system.

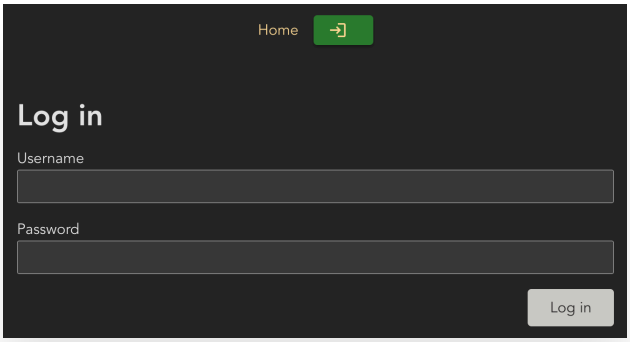

Figure 2.15. Login page in the AI Text Decomposer.

A new tab, “AI Assistant,” is available when the user is logged in. On that screen, the user has the following options:

- *choose the LLM model* for data processing. There are available ***GPT 3 Turbo*** (gpt-3.5-turbo-1106) and ***GPT 4 Turbo*** (gpt-4-1106-preview) models provided by OpenAI API. The analysis of the price schema for that model is described in Section 2.4.1.

- Select a topic with the Semantic Portal's educational content (branch).

- Add a list of expected concepts that LLM will use during a semantic analysis of educational content to extract Concepts and Theses.

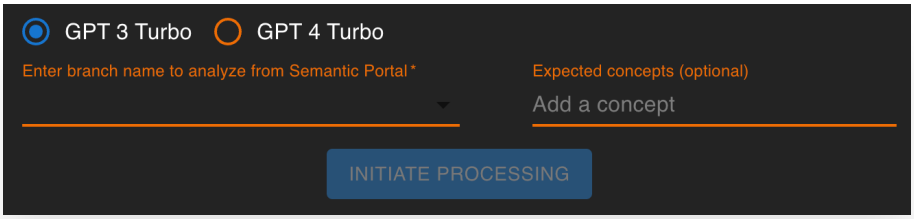

Figure 2.16. Form with processing options in the AI Text Decomposer.

When the branch is selected, the system fetches the educational content, and the user can read the material from the sidebar on the right side by clicking on the purple button “READ”.

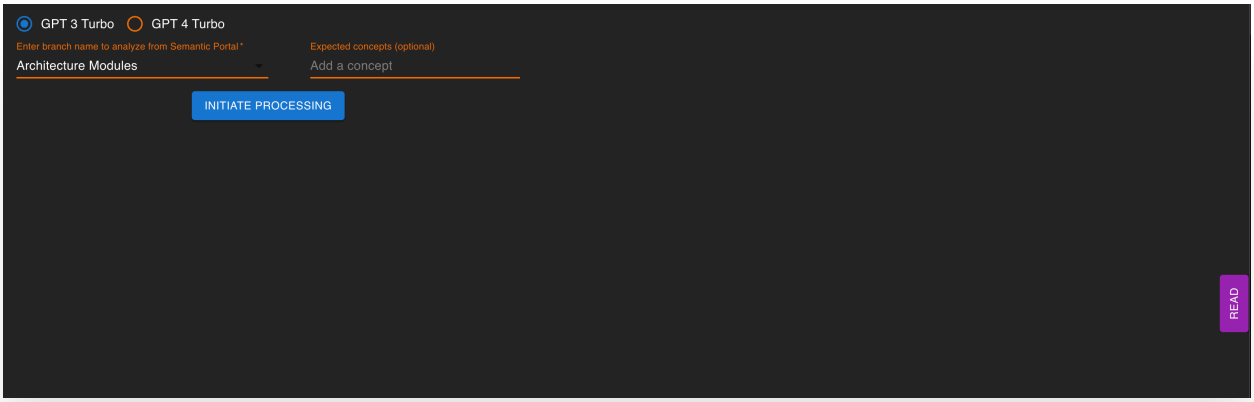

Figure 2.17. Page with READ purple button on the right side in the AI Text Decomposer.

To close the sidebar, a button “CLOSE” is on the left side (Figure 2.18).

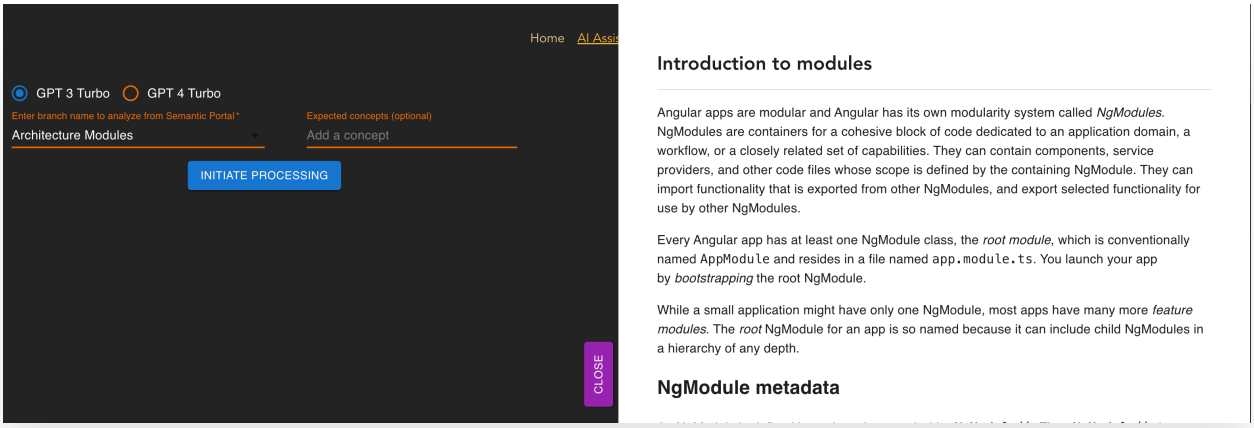

Figure 2.18. Page with opened sidebar and button CLOSE in the AI Text Decomposer.

By clicking “INITIATE PROCESSING,” the semantic analysis is initiated, and the backend will interact with the selected GPT model via OpenAI API during the processing. During the processing, the backend streams the output from LLM, and the user can check progress instantly.

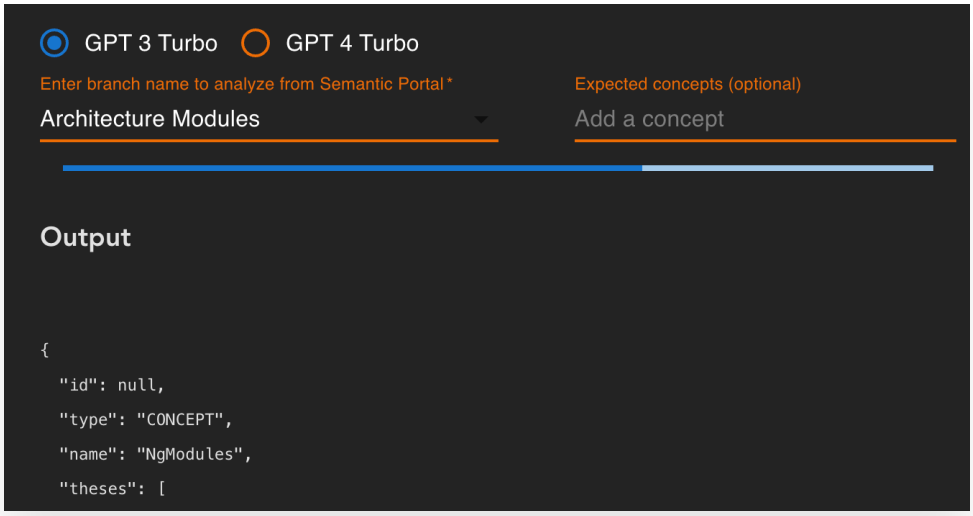

Figure 2.19. Page which demonstrates in progress event for processing action with partially consumed response in the AI Text Decomposer.

When streaming is finished and the whole output is consumed, the retrieved JSON is transformed into editable JSON available on the left side of the screen, where the user can update the result (Figure 2.20).

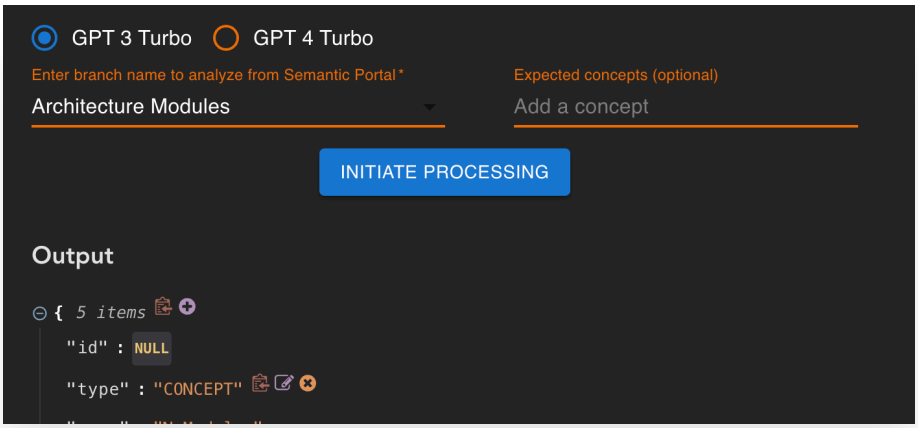

Figure 2.20. Page with consumed response from LLM in editable component for JSON object in the AI Text Decomposer.

On the left side, Concepts are visualized as a graph (Figure 2.21). Each node of the graph has encoded information in format {NUMBER}TC, like 4TC, where:

- {NUMBER}T is a count of theses of the Concept;

- C indicates that Concept has code snippets.

Also, on the top right corner, the user can redirect to the selected branch in the Semantic Portal by clicking “Check on Semantic Portal.”

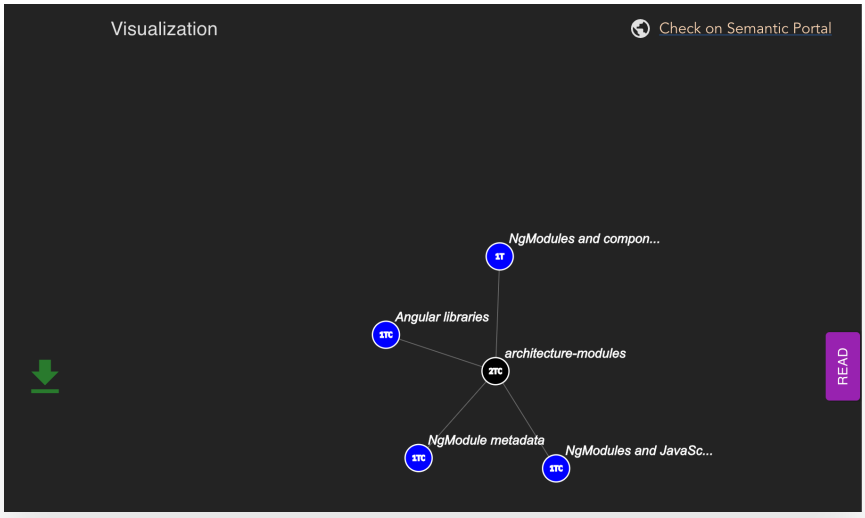

Figure 2.21. Page with graph visualization in the AI Text Decomposer.

Depending on the hierarchy level, nodes can be colored differently (Figure 2.22):

- root level: black;
- Level 1: blue;
- Level 2: red;
- Level 3: green;
- Level 4: yellow;
- Level 5: pink;
- Level 6: brown;
- Level 7: orange;
- Default color: white.

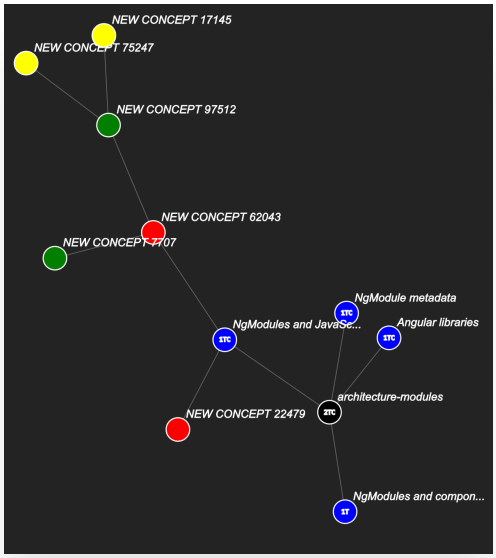

Figure 2.22. Visualization with multi-level hierarchy in the AI Text Decomposer.

When a user clicks on any node, the information of that node (Concept) displays on the sidebar on the left side of the screen (Figure 2.23). To close such a sidebar, click outside it.

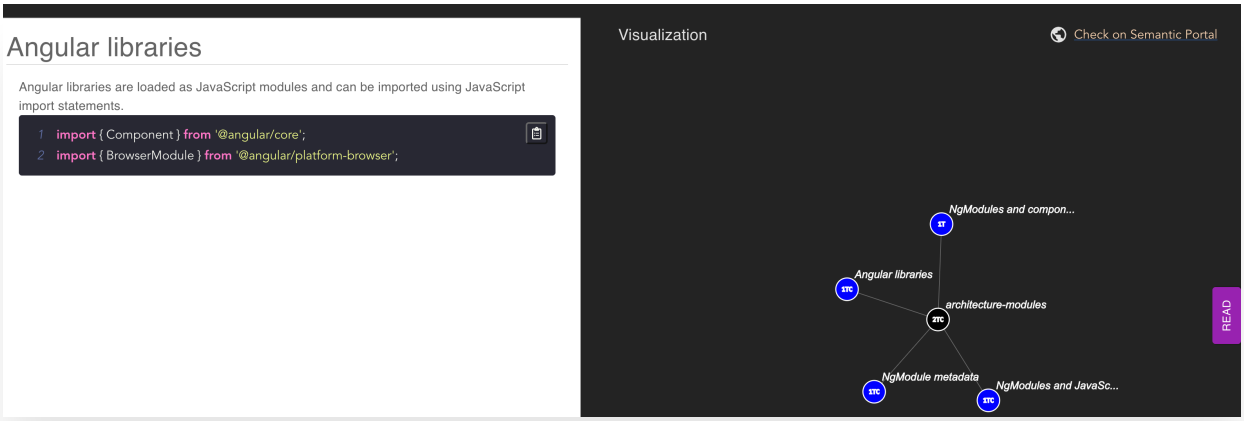

Figure 2.23. Page with left sidebar with concept details in the AI Text Decomposer.

As a final step in working with the application, the user exports the final result in JSON format by clicking the green icon “DOWNLOAD” in the middle of the page (Figure 2.24).

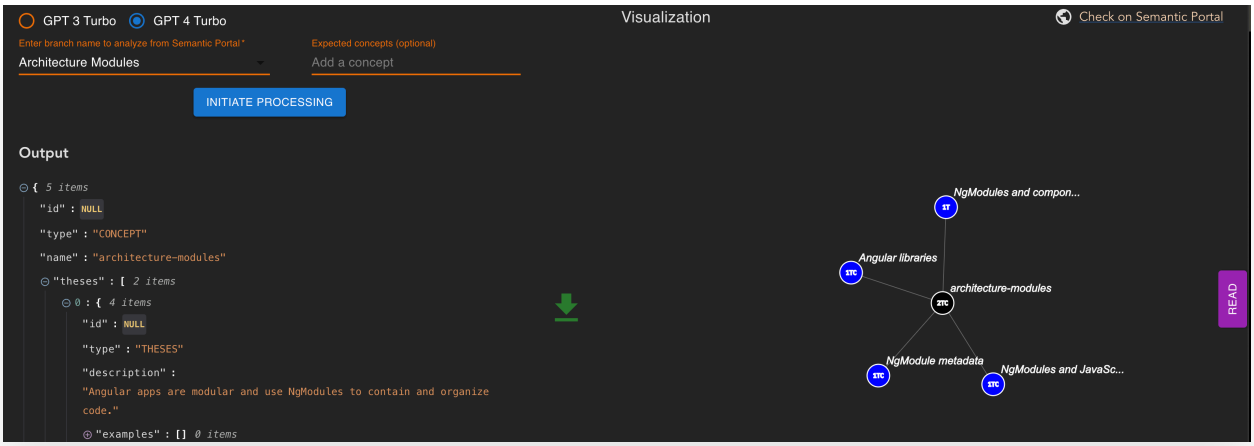

Figure 2.24. Visualization with multi-level hierarchy in the AI Text Decomposer.

The final JSON has the following structure:

```
{
  "id": "concept1",
  "type": "CONCEPT",
  "name": "Example Concept",
  "theses": [
    {
      "id": "thesis1",
      "type": "THESES",
      "description": "A brief description of the thesis",
      "examples": [
        {
          "type": "javascript",
          "example": "console.log('Hello World');"
        }
      ]
    }
  ],
  "children": [
    {
      "id": "concept2",
      "type": "CONCEPT",
      "name": "Child Concept",
      "theses": null,
      "children": null
    }
  ]
}
```

Thus, the “AI Text Decomposer” application allows the performance of semantic content analysis and automates the process of defining Concepts, Theses, and examples by using the GPT model via OpenAI API. The final result is available for downloading in JSON format for further integration.

Chapter 3. Future Outlook

In the following chapter, we are going to discuss different areas for improvement of the architecture and extension of future application capabilities.

3.1. Choosing the Suitable LLM Type for Semantic Content Analysis

As mentioned before, NLP transformers can be of two types: encoder and decoder. In Chapter 2, we used the GPT model, a decoder-type transformer. That type of model is pre-trained for generative tasks with some capabilities for classification or NER tasks, which is not its primary feature.

From that point, to increase the quality of the semantic content analysis, the NER part of processing should be done by encoder or encoder-decoder transformers, as was mentioned in Chapter 1.2.2. it may be BERT, BART, or T5 models. Such models are pre-trained and optimized to capture the context around each word and better understand the context and semantics of words in the input sequence. But, as said before, there is nothing wrong with using decoders for NER tasks, but the internal work and the quality of the result will be different.

From that point of view, let's capture the main features of the capstone project in the context of semantic content analysis:

- NER feature to decompose input sequence (educational content) and define Concepts, Theses, and other components (analysis part).
- Apply auto-formatting for output results of NER processing, especially valuable to examples in the form of code snippets (generative part).
- Transform the final result into JSON format using a predefined schema (generative part).

Thus, the capstone project combines encoder and decoder tasks, and the optimal model for the capstone application is a combined type of transformers represented by BART or T5 models. Such models allow to perform properly both tasks: semantic content analysis (NER) and generative tasks. The additional model required for similarity analysis is an embedding model, which will be discussed in the following section.

3.2. Utilizing Similarity Analysis in AI Text Decomposer

In Section 2.3.3. the high-level solution architecture was described. That solution doesn't cover the following things:

- Analyze the existing data regarding the target topic in the e-learning system.
- Automatic integration of the final result with the target system.

Generally, the capstone project has two core aims:

1. Automate the semantic analysis process.
2. Integrate the final result into the main system, the e-learning platform.

As the output of semantic content analysis, the application captures entities of different types: Concepts and Theses and their components like examples. Ideally, these entities should not be duplicated, so the system should perform a similarity analysis logic to dedupe entities. For that purpose, LLM should know additional application-related context, which was not included during the general model training. There is a technique called Retrieval Augmented Generation (RAG) to achieve that. That approach means sharing a custom knowledge base with LLM to be applied during the processing [25].

The embedding is the technique "used to perform a similarity analysis in NLP. Embeddings are numerical representations of real-world objects that machine learning (ML) and artificial intelligence (AI) systems use to understand complex knowledge domains like humans do" [26]. To work with embeddings, there are specialized types of databases: vector databases. An example of such databases is ChromaDB [27] or the cloud-based solution Pinecone [28].

OpenAI offers different embedding models to perform similarity analysis; one such model is text-embedding-ada-002 [29].

To inject a similarity analysis feature, OpenAI provides a special type of instruction: Function [30]. It allows to connect LLM with external tools, which provide additional capabilities to LLM. In addition, LangChain allows users to apply functions to the chain processing via Agents [31], which convert the described tools into OpenAI API format. Tools are instructions with business logic defined on the application level. To run agents, the LangChain uses the Agent Executor class.

The naive flow of similarity analysis leveraging LLM is demonstrated in Figure 3.1.

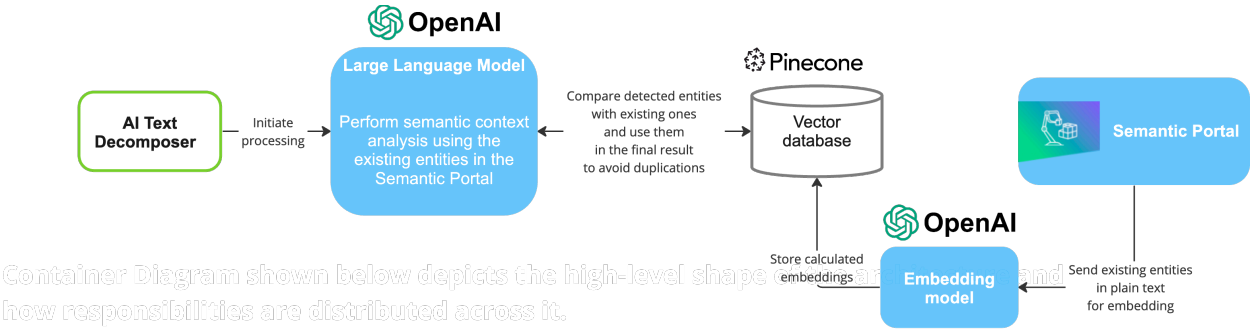

Figure 3.1. Naive flow of similarity analysis.

Therefore, to apply similarity analysis. two things should be done:

1. Load existing entities (Concepts, Theses, examples) into the vector database. During the loading of plain text, the vector database automatically creates an embedding representation of that data.
2. Set up the interaction between LLM and vector database to perform similarity analysis.

Internally, there is no direct interaction between LLM and the vector database. The backend application works as a bridge or proxy to connect LLM with the vector database in that flow. In Figure 3.2. demonstrated the high-level data flow for similarity analysis.

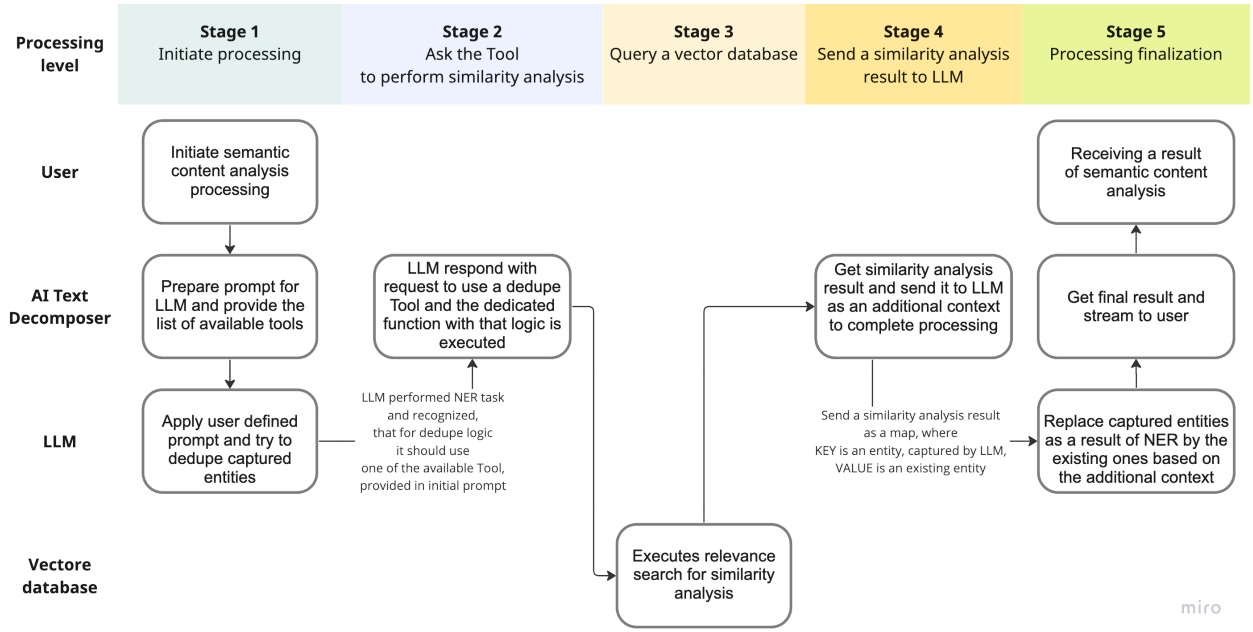

Figure 3.2. Similarity analysis data flow.

The described flow shows the application of the RAG pattern and how LLM is enriched with additional context.

On the application level, the implementation of the Tool is shown in Figure 3.3, where args_schema argument describes the schema of the expected arguments for the Tool.

```python
def perform_similarity_analysis(entities):
    """
    Perform a similarity analysis by querying a vector database
    :param entities: a grouped entities by entity type (Concept, Theses, Example)
    :return: a dictionary of capture entities with similar existing ones
    """


1 usage
class EntitiesArgsSchema(BaseModel):
    entities: dict[str, str]

similarity_analysis_tool = Tool.from_function(
    name="perform_similarity_analysis",
    description="""
        Given a grouped entities by entity type (Concept, Theses, Example),
        return the mapping of capture entities with similar existing ones.
    """,
    func=perform_similarity_analysis,
    args_schema=EntitiesArgsSchema
)
```

Figure 3.3. Tool definition in LangChain.

The perform_similarity_analysis function contains actual logic for how to interact with the vector database. The result of such a function returns a mapping between the detected entities by LLM and the corresponding similar value in the system. LLM uses existing ones for the final output, as described in the prompt. The function provided to Tool via argument func.

In Figure 3.4. demonstrates how to define an Agent with the defined tool and which prompt is expected to be used for proper processing by LLM. In the prompt, we mentioned a function that we expect to use for similarity analysis, and its name is the same as we defined in the Tool with the name argument: “perform_similarity_analysis.”

```
prompt = ChatPromptTemplate(
    messages=[
        SystemMessage(content=(
            """
            ...
            ### TOOL USAGE INSTRUCTION ###
                Replace captured entities with the existing ones in the system if possible.
                Use for that the "perform_similarity_analysis" function.
                That function expects as an argument a dictionary,
                where KEY is an entity type (CONCEPT, THESES, EXAMPLE),
                and VALUE is a list of captured entities of the corresponding types.
            ### TOOL USAGE INSTRUCTION ###
            """
        )),
    ]
)


agent = OpenAIFunctionsAgent(
    llm=chat,
    prompt=prompt,
    tools=[similarity_analysis_tool]
)
```

Figure 3.4. Agent definition in LangChain.

During the processing, LLM will perform a semantic content analysis to identify Concepts, Theses, and their examples following the prompt instructions. Finally, LLM will use the tool “perform_similarity_analysis” to replace some entities in the final result with similar ones in the system.

3.3. Summary of Chapter 3

Depending on the transformer type that LLM implements, some tasks may be applicable to decoders, but others are more efficient by encoders or mixed-type models. For similarity analysis, the core concept is embedding, which requires specialized embedding models.

Any LLM is pre-trained on some limited dataset and cannot do everything. There is an RAG technique that describes how to enrich the capabilities of the LLM. The implementation of RAG in the OpenAI ecosystem is represented via the Function concept and in the LangChain library via Tool and Agent abstractions.

Conclusions

During the project, we analyzed the purpose of different types of transformers as the latest paradigm in NLP. We implemented the AI-powered application to perform semantic content analysis to replace the manual process in the existing e-learning system, the Semantic Portal.

All aims of the research are accomplished:

1. Described the peculiarities of different kinds of transformers and defined their use cases. The encoder-based models (BERT, ALBERT) are focused on context understanding and NER operations, whereas the decoder-based models (GPT) are specialized in text generation tasks. The encoder-decoder models (BART, T5) combine the capabilities of encoder and decoder models.

2. Introduced a definition of the semantic unit and defined its main elements: Branch, Concept, Theses, and Example. Highlighted that the Concept contains at least one Theses and can contain nested concepts. The Theses can contain multiple Examples of different types, which LLM defines during the semantic content analysis.

3. Implemented the application that uses LLM for semantic content analysis. For the project purpose, the usage of the GPT model was argued. The application interacts with the Semantic Portal as an e-learning system and with LLM via OpenAI API. Provided detailed architecture in C4 format and demonstrated system context, container, component, and deployment diagrams alongside an overview of essential modules, libraries, and code examples. Differences in the usage of different models are shown in the example of the GPT-3 and GPT-4 models.

4. Described in detail the implementation of essential features of the capstone project, like prompt engineering, data streaming, and in-app editing. Mentioned that to improve the quality of LLM's output, the prompt should have clear and conscious instructions without ambiguous interpretation.

5. Provided rationale for the usage of different dependencies, namely LangChain, AWS, and OpenAI API. Shown how the LangChain simplifies the interaction with LLM and provides an additional abstraction layer to work with any providers. The AWS platform provides an infrastructure for continuous integration and deployment process.

6. Shown how the implemented work can be extended and improved using the encoder-decoder model for content analysis purposes, the RAG approach to enrich the LLM capabilities. Argued in detail the importance of similar analysis and described the key stages of its implementation with code examples.

Bibliography

1. Rybak, M., & Tytenko, S. (2023). Leveraging ChatGPT for educational text analysis aiming assessment generation. Modern Engineering and Innovative Technologies, 1(29-01), 112–116. https://doi.org/10.30890/2567-5273.2023-29- 01-045.

2. Semantic Portal. (n.d.). Educational project. Retrieved from http://semantic-portal.net/about.

3. LangChain. (n.d.). Official site. Retrieved from https://www.langchain.com.

4. Eastgate Software. (n.d.). What is natural language processing? Retrieved from https://eastgate-software.com/what-is-natural-language-processing.

5. O'Reilly. (n.d.). Natural language processing. Retrieved from https://learning.oreilly.com/library/view/natural-language-processing/9780596803346/pr02.html#I_sect1_d1e168.

6. IBM. (n.d.). What is natural language processing? IBM Watson natural language processing solutions. Retrieved from https://www.ibm.com/natural-language-processing.

7. Jurafsky, D., & Martin, J. (2008). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Retrieved from https://www.researchgate.net/publication/200111340_Speech_and_Language_Processing_An_Introduction_to_Natural_Language_Processing_Computational_Linguistics_and_Speech_Recognition/citations.

8. Hutchins, W. J. (2000). Early years in machine translation: Memoirs and biographies of pioneers. Retrieved from https://archive.org/details/earlyyearsinmach0000unse/page/n7/mode/2up and https://www.google.ca/books/edition/Early_Years_in_Machine_Translation/3dU5AAAAQBAJ?hl=en&gbpv=1&pg=PA1&printsec=frontcover.

9. Manning, C. D., & Schütze, H. (1999). Foundations of statistical natural language processing. Retrieved from https://nlp.stanford.edu/fsnlp.

10. O'Reilly. (n.d.). Introduction to transformer. Retrieved from https://learning.oreilly.com/videos/introduction-to-transformer/9780137923717/9780137923717-TMN1_01_01_01/.

11. Sutskever, I. (n.d.). Training recurrent neural networks. PhD Thesis. Retrieved from https://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf.

12. Goldberg, Y. (2017). Neural network methods for natural language processing. Retrieved from https://aclanthology.org/J18-1008.pdf.

13. Vaswani, A., et al. (2017). Attention is all you need. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

14. Devlin, J., et al. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding.

15. Cass, J. (2023). ChatGPT statistics: Users, growth, revenue & more. Retrieved from https://justcreative.com/chatgpt-statistics.

16. Cubettech. (2023). Unleashing the potential of GPT-3 in the education space: How product companies can elevate their learning solutions. Retrieved from https://cubettech.com/resources/blog/unleashing-the-potential-of-gpt-3-in-the-education-space-how-product-companies-can-elevate-their-learning-solutions.

17. Jain, G., Gurupur, V., Schroeder, J., & Faulkenberry, E. (2014). Artificial intelligence-based student learning evaluation: A concept map-based approach for analyzing a student's understanding of a topic. IEEE Transactions on Learning Technologies, 7, 267-279. https://doi.org/10.1109/TLT.2014.2330297.

18. Tytenko, S. V. (2019). Generation of test tasks in the system of distance learning based on the model of formalization of didactic text. Scientific News of NTUU "KPI", No1(63), 47–57. Retrieved from http://www.setlab.net/?view=Tytenko-test-generation.

19. Zubrinic, K., Kalpic, D., & Milicevic, M. (2012). The automatic creation of concept maps from documents written using morphologically rich languages. Expert Systems with Applications, 39(16), 12709–12718. https://doi.org/10.1016/j.eswa.2012.04.065.

20. Fedenko, V. A., Polienova, V. A., & Tytenko, S. V. (2022). Educational mobile application based on concept maps. Modern Engineering and Innovative Technologies, Issue 23, Part 1. https://doi.org/10.30890/2567-5273.2022-23-01-014.

21. Malan, D. J., & Co. (n.d.). Teaching CS50 with AI. Retrieved from https://cs.harvard.edu/malan/publications/V1fp0567-liu.pdf.

22. Nikolic, S., Daniel, S., Haque, R., Belkina, M., Hassan, G. M., Grundy, S., Lyden, S., Neal, P., & Sandison, C. (2023). ChatGPT versus engineering education

assessment: A multidisciplinary and multi-institutional benchmarking and analysis of this generative artificial intelligence tool to investigate assessment integrity. European Journal of Engineering Education, 48:4, 559-614. https://doi.org/10.1080/03043797.2023.2213169.

23. OpenAI. (n.d.). API Pricing. Retrieved from https://openai.com/pricing.

24. OpenAI. (n.d.). Six strategies for getting better results. Retrieved from https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results.

25. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459-9474. https://doi.org/10.48550/arXiv.2005.11401.

26. AWS. (n.d.). What are embeddings in machine learning? Retrieved from https://aws.amazon.com/what-is/embeddings-in-machine-learning.

27. ChromaDB. (n.d.). Official site. Retrieved from https://www.trychroma.com.

28. Pinecone. (n.d.). Official site. Retrieved from https://www.trychroma.com.

29. OpenAI. (n.d.). What are embeddings? OpenAI documentation. Retrieved from https://platform.openai.com/docs/guides/embeddings/what-are-embeddings.

30. OpenAI. (n.d.). Function calling. OpenAI documentation. Retrieved from https://platform.openai.com/docs/guides/function-calling.

31. LangChain. (n.d.). Agents. LangChain official documentation. Retrieved from https://python.langchain.com/docs/modules/agents.

32. Liza U. (n.d.). Semantic Analysis: What Is It, How & Where To Works. Retrieved from https://www.questionpro.com/blog/semantic-analysis.

33. Rybak M. (2024). Text Decomposer. Bitbucket repository. Retrieved from https://bitbucket.org/mykolarybak/text-decomposer/src/develop.