

COSC 3P99 Report

Name:- Vivek Salwan

Student ID:- 6951826

1. INTRODUCTION

In this report, I will explain everything that I have learned over time in my COSC 3P99 course under the guidance of Professor Naser Ezzati-Jivan. This report will cover all the research papers and articles that I have read during the course, and I will explain in detail what I have learned from them. Furthermore, I will explain the project that I have been working on, which uses large language models to generate documentation for coding projects.

2. RESEARCH PAPERS AND ARTICLES

1st - Augmenting Intelligent Document Processing (IDP) Workflows with Contemporary Large Language Models (LLMs)

The paper "Augmenting Intelligent Document Processing (IDP) Workflows with Contemporary Large Language Models (LLMs)" explores the integration of Large Language Models (LLMs) into the Intelligent Document Processing (IDP) workflow" by author Shreekanth Mandvikar. This paper talks about the challenges faced by the growing volume of documents and company data that an organization needs to manage in order to function properly. This research paper suggests that the integration of LLMs into IDP workflows can significantly improve the extraction of data from these documents.

This paper has provided a really good introduction to LLMs as it talks about how LLMs are predominantly built on a class of deep learning architectures called transformer networks. It also talks about how LLMs work as explained below:-

2.1. How Large Language Model Works

Large Language Models (LLMs) are predicated on statistical methodologies that encode a probabilistic relationship among sequences of words [4], [5]. These models use a probability distribution denoted as $P(w_1, \dots, w_L)$, which aims to approximate the empirical distribution observed in a substantial corpus of text in a specific language. The simplest form of such a model is the "1-gram" model, as given below, which operates under the assumption that words are independently distributed [6], [7].

$$P(w_1, \dots, w_L) = \prod_{i=1}^L P(w_i); \quad P(W) = \frac{n(w)}{W}$$

Where $n(w)$ and W represent the number of occurrences of w in the corpus and the total number of words in the corpus.

Where the probability of a word sequence $P(w_1, \dots, w_L)$ is computed as the product of the probabilities of individual words $P(w_i)$, the individual word probabilities are determined by the frequency of each expression in the corpus relative to the total number of words in that corpus [8], [9]. To assess the effectiveness of a language model, the standard metric employed is cross entropy, which quantifies how well the model's probability distribution mirrors the empirical

distribution observed in the corpus [10], [11]—the cross-entropy.

L is calculated as a sum of logarithmic terms related to the conditional probabilities of word sequences, often denoted as:

$$L = -\frac{1}{N} \sum_{i=1}^{N-n} \log P(W_{i+n} | W_i, W_{i+1}, \dots, W_{i+n-1})$$

Where the perplexity is represented as $\exp(-L)$. The equation is an objective function to minimise during the training phase. Various optimization techniques, such as backpropagation, can be employed.

In this paper, we can also see a detailed comparison among traditional Machine Learning (ML), Deep Learning (DL), and Large Language Models (LLMs) based on their training size data, model complexity, performance, cost, adaptability, etc. This information provides us with a lot of insights into how different models are suitable for different tasks and how LLMs are efficient at certain tasks like extracting meaningful information etc.

Table 1. Comparison among traditional Machine Learning (ML), Deep Learning (DL), and Large Language Models (LLMs)			
Comparison	Traditional ML	Deep Learning	Large Language Models
Training Data Size	Large (Thousands to Millions)	Large (Thousands to Millions)	Very Large (Billions+)
Feature Engineering	Manual (Domain expertise features)	Automatic (Self-learns features)	Automatic (Self-learns required)
Model Complexity	Limited (Linear, Tree-based)	Complex (Convolutional, Recurrent)	Very Complex (Up to billions of parameters)
Interpretability	Good (Transparent algorithms)	Poor (Black-box nature)	Poorer (Extremely complex, black-box nature)
Performance	Moderate (Sufficient for simpler tasks)	High (Effective for complex for multiple tasks)	Highest (State-of-the-art tasks)
Hardware Requirements	Low (CPUs sufficient)	High (GPUs often required)	Very High (Multiple GPUs, TPUs often required)
Computational Cost	Low to Moderate	High	Very High
Real-Time Capabilities	Often Suitable	Less Suitable (due to Complexity)	Generally Not Suitable
Adaptability	Lower (Fine-tuning often leaning)	Moderate (Some transfer learning)	High (Very adaptable with needed)
Software Libraries	Scikit-learn, Stats models	TensorFlow, PyTorch	Hugging Face Transformers, GPT-specific implementations

Furthermore, it also talks about the different classes of LLMs, each with their unique capabilities for different use cases, as shown below:-

Table 2. Overview of different classes of large language models			
Model Class	Short Description	Typical Use Cases	Example Models
Autoregressive	Specialized in generating coherent and contextually relevant text.	Content creation, Conversational agents	GPT-3 (Generative Pretrained Transformer 3)
Encoder-only	Optimized for understanding and representing text rather than generating it.	Text classification, Sentiment analysis	BERT (Bidirectional Encoder Representations from Transformers)
Encoder-Decoder	Capable of both understanding and generating text, offering versatility.	Machine translation, Summarization	T5 (Text-to-Text Transformer), BART (Bidirectional and Autoregressive Transformers)
Specialized	Designed for specific tasks that require high performance but lower computational costs.	Efficient text classification, Information retrieval	ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately)

This table shows Autoregressive models, such as GPT-3 (Generative Pretrained Transformer 3), are compelling at generating human-like text, making them ideal for tasks such as content creation and another class Encoder-only models like BERT (Bidirectional Encoder Representations from Transformers), are designed to understand and represent textual data rather than generate it. A third class comprises Encoder-Decoder models like T5 (Text-to-Text Transformer) or BART (Bidirectional and Auto-Regressive Transformers), which are versatile in both understanding and generating text. Moreover, specialized models like ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) are designed for efficiency and are suitable for tasks that demand lower computational resources but require high performance.

Below we can explain the experiment in 4 stages

3.1 Document Classification Stage

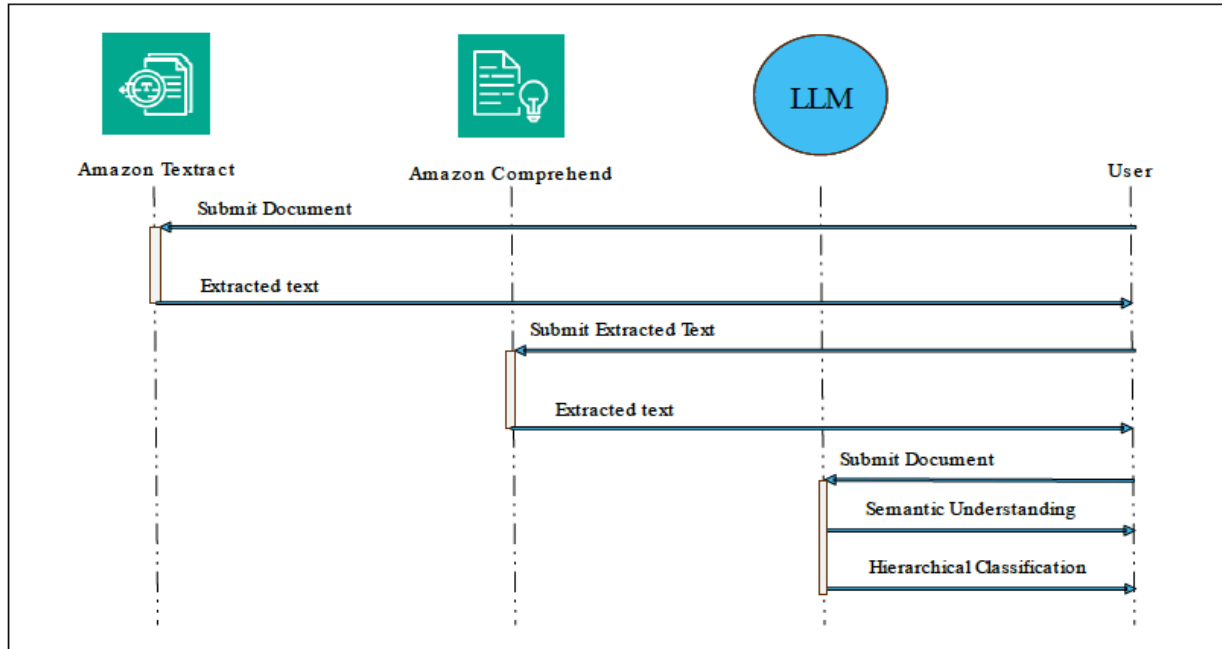


Fig. 1 Contributions of LLMs in the classification stage of IDP

In this stage, we make use of Amazon Web Services (AWS) as AWS provides Amazon Textract and Amazon Comprehend. Amazon Textract helps extract text and other data from scanned documents. In the first stage, the user needs to submit a document to Amazon Textract, which begins the IDP process. After extracting text and data, this includes recognizing text from various document formats and layouts. It sends the extracted text back to the user for the next service in the workflow. The extracted document is then submitted to Amazon Comprehend, which uses Natural Language Processing (NLP) techniques to analyze the context and semantic details of the extracted text, and in parallel, the extracted text is also submitted to an LLM. The LLM employs advanced techniques for a more nuanced document classification like using Semantic understanding and Hierarchical classification. In the end, the LLM sends the processed document to the user, concluding its role in the process. The output includes a more refined categorization of the document.

3.2 Document Extraction Stage

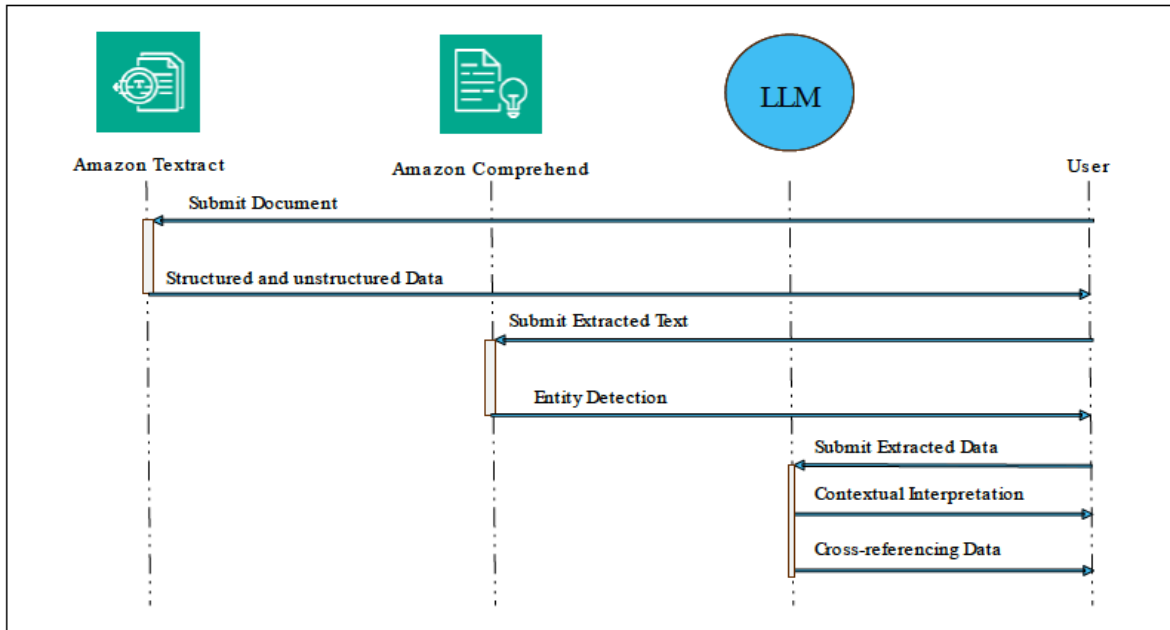


Fig. 2 Contributions of Large Language Models (LLMs) in the extraction stage of IDP

In this stage, we perform all the same steps as the previous stage as you can see in the above figure but when we feed the data into the LLM we improve it by using contextual interpretation to improve the understanding of extracted data by analyzing its context and we use Cross-reference data as LLMs link related data across the document ensuring consistent interpretation and reducing errors in data extraction. This stage is important for transforming raw data into structured formats that can be easily used in databases or other applications.

3.3 Review and Validation Stage

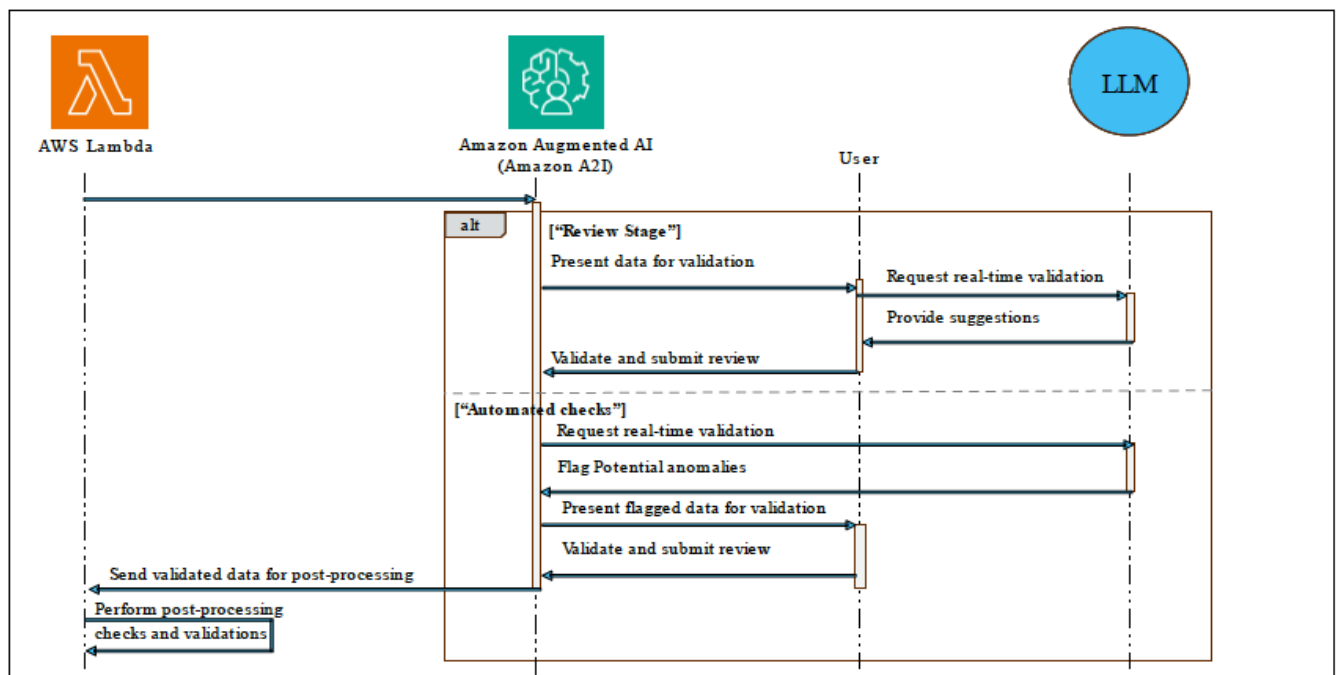


Fig. 3 Contributions of Large Language Models (LLMs) in the review/validation stage of IDP

In this stage, review and validation take place where the process begins with AWS Lambda sending the extracted data to Amazon Augmented AI (A2I) for review. Amazon A2I facilitates the integration of human reviewers in the workflow. In this process, Human reviewers assess the extracted data to validate its accuracy and completeness. In parallel to the manual review, LLMs are also used for automated anomaly detection. LLMs analyze the data for any inconsistencies or unusual patterns that may indicate errors. Once the review process is complete, the validated data is sent back to AWS Lambda for post-processing. The final step involves AWS Lambda finalizing the data processing, incorporating all the corrections and validations from the review stage. The data is now ready for further use in applications or for entry into databases.

3.4 Document Enrichment Stage

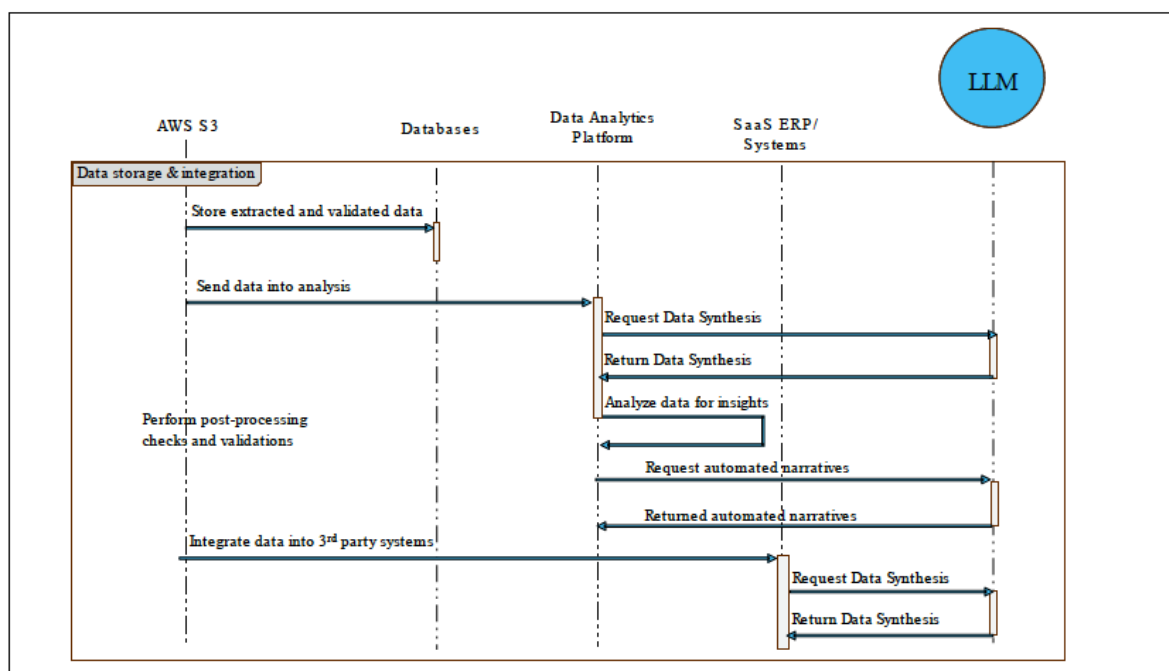


Fig. 4 Contributions of Large Language Models (LLMs) in the review/validation stage of IDP

In the final stage, After the data has been extracted, validated, and possibly enriched in previous stages, it is stored in Amazon Simple Storage Service (S3) and possibly other databases. LLMs are utilized to synthesize the stored data to ensure consistency and coherence across different data types and sources. The synthesized data is then sent to Data Analytics platforms, where it is prepared for detailed analysis. As part of making the data more accessible and understandable, LLMs are employed to generate automated narratives or summaries based on the analyzed data. Following analysis, the data is integrated into third-party applications or systems, such as Enterprise Resource Planning (ERP) systems or other Software as a Service (SaaS) platforms. Once integrated, the data is ready for use within the organization's broader ecosystem, supporting various business functions, from customer relationship management to financial analysis and beyond.

In conclusion, this research paper effectively explains how Large Language Models (LLMs) can be integrated into the Intelligent Document Processing (IDP) workflow to enhance each stage, from classification and extraction to review, validation, and integration. By using LLMs large organizations can significantly improve their methods of document processing. LLMs provide advanced capabilities such as semantic understanding, hierarchical classification, contextual interpretation, and automated narrative generation, which exceed traditional document processing methods. However, we should also consider that the implementation of LLMs also introduces complexities related to computational demands, potential biases, and the need for sophisticated model management. Organizations must consider all these factors to fully leverage the potential of LLMs in enhancing IDP workflows.

2nd - Automatic Code Documentation Generation Using GPT-3

The research paper "Automatic Code Documentation Generation Using GPT-3" by Junaed Younus Khan and Gias Uddin explores the use of OpenAI's Codex, a GPT-3-based model, for generating code documentation automatically. Codex is trained on both natural and programming languages and shows promise in improving the efficiency of software development by automating the documentation process.

The paper begins by addressing the importance of well-maintained code documentation in software development, especially highlighted during the shift to remote work due to COVID-19. Traditional documentation practices are costly, often outdated, and unpopular among developers, prompting a need for automation. Here we use Codex, a model derived from GPT-3 that is specially tuned for understanding and generating programming content and is trained on a vast dataset that includes GitHub contributions across multiple programming languages. This paper talks about an experiment that is carried out using Codex which is explained below:-

1. Data Collection and Preprocessing

Automatic Code Documentation Generation Using GPT-3

Table 1: Statistics of CodeSearchNet [25]

Language	Train	Valid	Test
Java	164,923	5,183	10,955
Python	251,820	13,914	14,918
PHP	241,241	12,982	14,014
GO	167,288	7,325	8,122
JavaScript	58,025	3,885	3,291
Ruby	24,927	1,400	1,261

The researchers used the CodeSearchNet dataset, which is widely employed for various software engineering tasks involving code understanding and generation. This dataset includes pairs of code and corresponding documentation for six programming languages: Java, Python, PHP, GO, JavaScript, and Ruby. To perform the experiment properly the preprocessing steps include Removing comments from the code to ensure that the model's output was purely based on the code's content rather than comments. Filtering out examples where the code could not be parsed into an abstract syntax tree, the documentation was too short or too long or contained non-English language or special tokens like tags.

2. GPT-3 Model and Parameter Setup

```
Code:
def add(x, y):
    return x+y
Documentation: Adds two numbers.
Code:
def subtract(x, y):
    return x-y
Documentation: [To be generated by Codex]
```

Figure 2: Sample prompt format for one-shot learning

The Codex model, a variant of GPT-3 specifically designed for understanding and generating code, was utilized. This model is pre-trained on a large corpus of natural language and programming code from sources like GitHub. The model was tested with zero-shot and one-shot learning approaches. In zero-shot learning, the model received only a task description without any examples. In one-shot learning, it received one example of a code-documentation pair before being asked to generate documentation for a new piece of code. Parameters like 'Temperature' and 'Top-p' were adjusted to control the randomness of the generated output, with 'Temperature' set low to reduce randomness and ensure more predictable outputs.

3. Evaluation of Generated Documentation

Table 2: Results on documentation generation (BLEU score)

Model	Ruby	JavaScript	GO	Python	Java	PHP	Overall
Seq2Seq [48]	9.64	10.21	13.98	15.93	15.09	21.08	14.32
Transformer [54]	11.18	11.59	16.38	15.81	16.26	22.12	15.56
RoBERTa [31]	11.17	11.90	17.72	18.14	16.47	24.02	16.57
CodeBERT [17]	12.16	14.90	18.07	19.06	17.65	25.16	17.83
PLBART [5]	14.11	15.56	18.91	19.30	18.45	23.58	18.32
CoTexT (2-CC) [40]	13.07	14.77	19.37	19.52	19.1	24.47	18.38
CoTexT (1-CC) [40]	14.02	14.96	18.86	19.73	19.06	24.58	18.55
REDCODER [38]	-	-	-	21.01	22.94	-	N/A
REDCODER-EXT [38]	-	-	-	20.91	22.95	-	N/A
Codex (0-shot)	5.41	9.83	15.80	18.93	13.59	13.32	12.81
Codex (1-shot)	16.04	16.58	20.94	22.28	22.81	25.13	20.63

The effectiveness of the generated documentation was measured using both quantitative and qualitative metrics:

Quantitative Metrics: BLEU Score is a standard metric for evaluating machine-generated text based on n-gram similarity between the generated text and a reference text. The study used a smoothed version of the BLEU score to account for the variability in documentation length and content. Moreover, we observed that though Codex with zero-shot learning could not achieve satisfactory results (mostly because it fails to learn the expected documentation format), the performance greatly improves with one-shot learning. Codex (with one-shot) shows the best overall performance among all approaches with an average BLEU score of 20.63 while the nearest competitor CoTexT achieves 18.55 (11.21% improvement).

Java <pre>protected ImageIcon createImageIcon(String path, String description) { java.net.URL imgURL = getClass().getResource(path); if (imgURL != null) { return new ImageIcon(imgURL, description); } else { System.err.println("Couldn't find file: " + path); return null; } }</pre> <p>Actual: Returns an ImageIcon, or null if the path was invalid. Codex: Returns an ImageIcon, or null if the path was invalid.</p>
Python <pre>def check_for_file(self, file_path): try: files = self.connection.glob(file_path, details=False, invalidate_cache=True) return len(files) == 1 except FileNotFoundError: return False</pre> <p>Actual: Check if a file exists on Azure Data Lake. Codex: Check if a file exists on the remote machine.</p>
PHP <pre>public static function base256ToDec(\$number) { Validate::canCastAsString(\$number, 'number'); \$result = 0; \$base = 1; for (\$i = strlen(\$number) - 1; \$i >= 0; \$i--) { \$result = bcadd(\$result, bcmul(ord(\$number[\$i]), \$base)); \$base = bcmul(\$base, 256); } return \$result; }</pre> <p>Actual: Convert base 256 number to decimal number. Codex: Converts a base 256 number to a base 10 number.</p>
JavaScript <pre>function validateProtocolOptions(protocolOptions) { if (!protocolOptions) { throw new TypeError('protocolOptions not defined in options'); } const version = protocolOptions.maxVersion; if (version && (typeof version !== 'number' !types.protocolVersion.isSupported(version))) { throw new TypeError(util.format('protocolOptions.maxVersion provided (%s) is invalid', version)); } }</pre> <p>Actual: Validates the protocol options. Codex: Validate protocol options.</p>
Ruby <pre>def remove_file(*paths) path = File.join(*paths) log.debug(log_key) { "Removing file {path}" } FileUtils.rm_f(path) path end</pre> <p>Actual: Remove the file at the given path. Codex: Removes the file located at the given path. If the path is not present a noop is performed.</p>
GO <pre>func (os *orderedSet) append(s string) { if _, ok := os.included[s]; !ok { os.list = append(os.list, s) os.included[s] = struct{}{} } }</pre> <p>Actual: append adds s to the end of os, only if it is not included already. Codex: append adds s to the orderedSet if it is not already present.</p>

Figure 3: Examples of documentation by Codex (1-shot)

Qualitative Metrics:

Documentation length and readability is measured by the Flesch-Kincaid Grade Level, which assesses the readability based on sentence length and word complexity. Its Informativeness is evaluated using TF-IDF scores to compare the information content of the generated documentation against the actual documentation.

In conclusion, the experiment emphasizes the potential of using advanced language models, like Codex, to automate the generation of code documentation. It showcases the model's ability to adjust to different programming languages and create top-notch documentation with minimal input. This paves the way for its practical use in software development and maintenance tasks.

3rd - Enabling BLV Developers with LLM-driven Code Debugging

The research paper titled "Enabling BLV Developers with LLM-driven Code Debugging" by Clark Saben and Prashant Chandrasekar talks about BLVRUN, a command-line tool designed to assist blind and low-vision (BLV) developers by summarizing complex error messages into concise, easily understandable summaries. This tool is used to fine-tune a large language model to enhance accessibility and streamline the debugging process for developers who face significant challenges using traditional debugging tools.

Below we can explain the experiment carried out in the paper:-

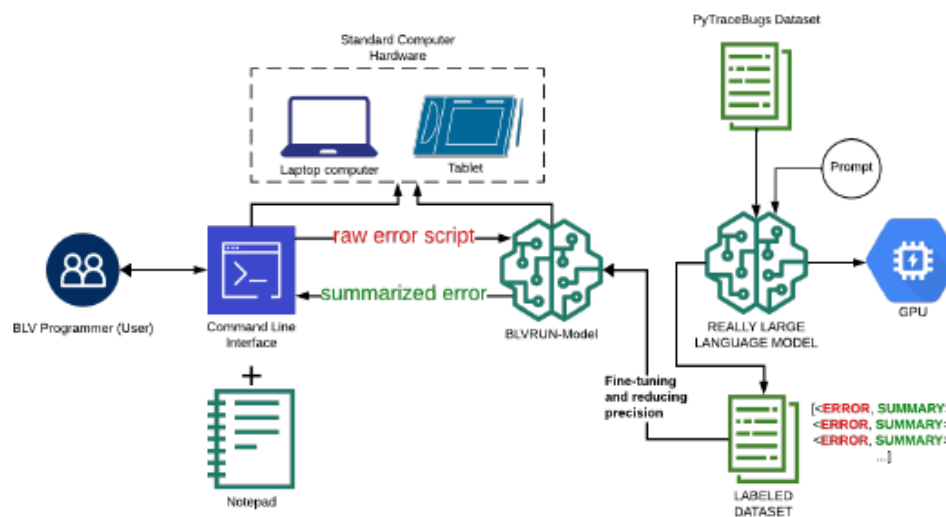


Fig. 1. Architecture and Development Components of BLVRUN. Starting from the left, a BLV programmer, who using CLI and text buffers executes their Python code. When an error is produced, BLVRUN's script captures the verbose and unstructured text and only presents the user with a concise and accurate description of the error. This is possible because BLVRUN's model is fine-tuned using a dataset we created from PyTraceBugs. Finally, BLVRUN is optimized to run on any machine, thereby not requiring BLV programmers to depends on IDEs and/or switch contexts with ChatGPT-like solutions.

This experiment focuses on evaluating the effectiveness of the BLVRUN tool, which is designed to aid blind and low-vision (BLV) developers by summarizing traceback errors into understandable summaries. This evaluation aims to assess how well BLVRUN performs in real-world debugging scenarios compared to standard models and according to specific metrics.

1. Dataset Utilization

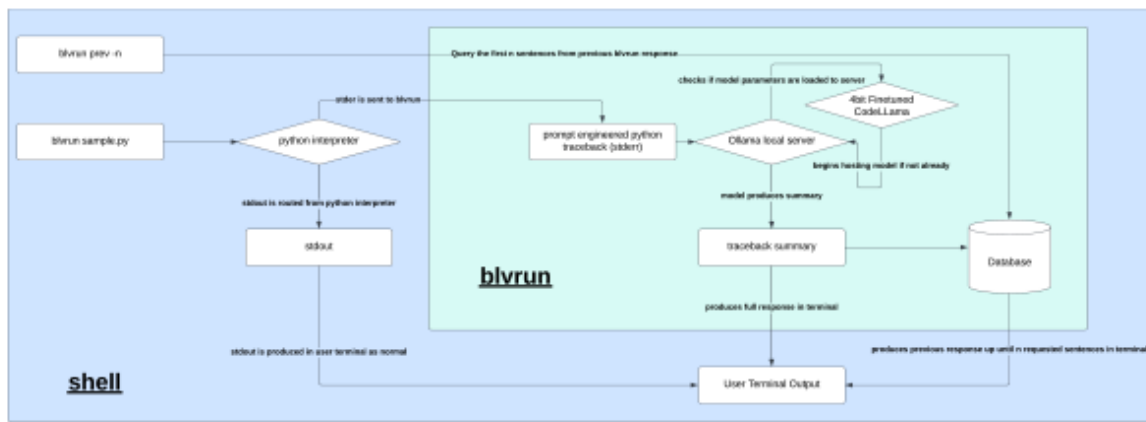
The PyTraceBugs Dataset is used for both training and testing. It includes a variety of traceback errors from Python programs, which are crucial for fine-tuning the model to accurately summarize common errors encountered by developers.

2. Fine Tuning

For fine tuning BLVRUN uses a version of the CodeLlama model, fine-tuned specifically on the PyTraceBugs dataset. This fine-tuning process adapts the model to better understand and generate summaries for Python traceback errors. Moreover, to enhance performance, the model's precision is reduced, by a process known as *quantization*. This step is crucial for ensuring that the model runs efficiently on standard consumer hardware.

3. User Interface

3.5 User Interface



The above diagram explains the BLVRUN interface, When `blvrn sample.py` is executed in the shell, the prompt is sent to our model that is hosted on a Ollama server. Our model produces a traceback summary that is sent back to the terminal and saved in a database. BLV programmers can see previously generated summaries using the `blvrn prev -n` command.

4. Evaluation

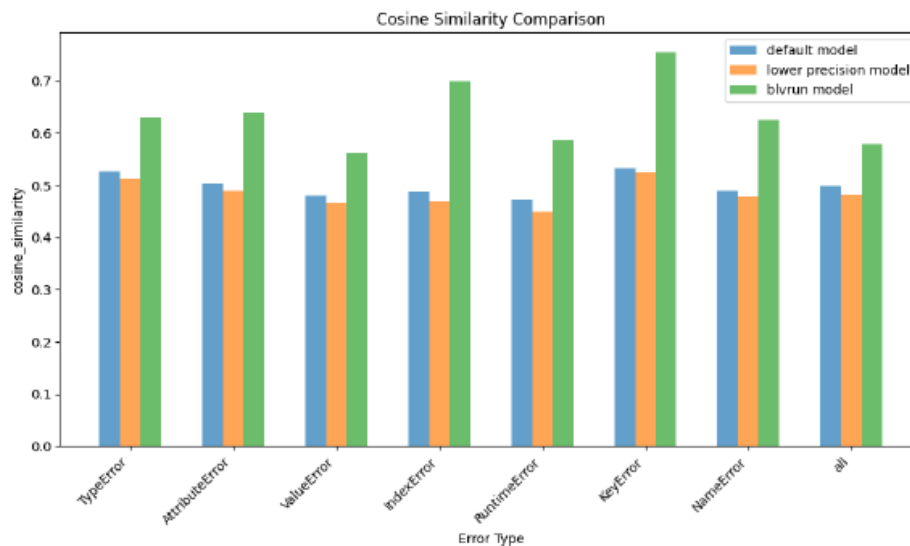


Fig. 4. Cosine similarity scores of summaries generated by (1) base model (with no fine-tuning or lowered precision), (2) base model (with lowered precision), (3) BLVRUN's fine-tuned and optimized model compared against "gold standard"

Cosine Similarity: This metric is used to evaluate the similarity between the error summaries generated by BLVRUN and a "gold standard" set of summaries. The gold standard is presumably generated using the most accurate settings or another high-performing model.

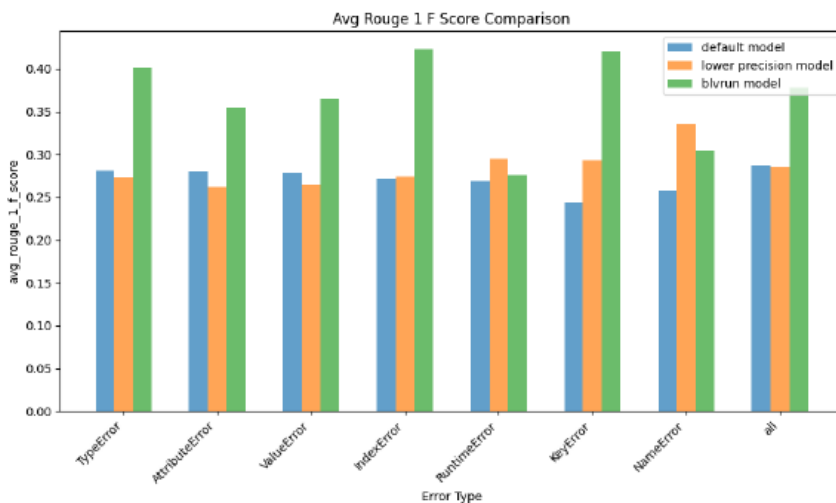


Fig. 5. ROUGE-1 f-scores of summaries generated by (1) base model (with no fine-tuning or lowered precision), (2) base model (with lowered precision), (3) BLVRUN's fine-tuned and optimized model compared against "gold standard"

ROUGE-1 Score: This metric measures the overlap of unigrams between the generated summaries and the gold standard. It assesses how many of the same words appear in both texts, which is an indicator of textual similarity and relevance.

The results, shown in figures within the paper (e.g., Figures 4 and 5), demonstrate that the fine-tuned and optimized BLVRUN model outperforms the base models in terms of both cosine similarity and ROUGE-1 scores. This indicates that the summaries it produces are both accurate and relevant, closely matching the gold standard.

In conclusion, this experiment shows that BLVRUN, with its specific optimizations and fine-tuning, provides significant benefits over standard models, making it an effective tool for helping BLV developers understand and resolve errors in their code more efficiently.

4th - CodePlan: Repository-level Coding using LLMs and Planning

The document titled "CodePlan: Repository-level Coding using LLMs and Planning" by Microsoft Research, India presents a detailed exploration of how Large Language Models (LLMs) can be effectively used to automate extensive coding tasks across entire repositories, rather than just localized coding issues. Traditional LLM applications like GitHub Copilot excel in localized coding scenarios but struggle with repository-wide tasks where changes need to be coordinated across files and may depend on each other.

Experimental Setup

1. Datasets

Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D C, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet

Repositories	Migration (C#)			Temporal Edits (Python)		
	Int-1	Int-2	Ext-1	T-1	T-2	T-3
Number of files	91	168	55	21	137	4
Lines of code	8853	16476	8868	3883	20413	1874
Number of files changed	47	97	21	2	2	3
Number of seed changes	41	63	42	2	1	1
Number of derived changes	110	375	16 0	8	3	10
Size of diff (lines)	1744	4902	1024	104	15	39
Size of seed edits (lines)	242	242	379	76	4	1
Prompt template size (lines)	81	81	81	75	75	75
URL	-	-	[7]	[8]	[1]	[3]

Table 2. Dataset statistics. Int-1,2 are internal (proprietary) repositories, Ext-1 and T-1,2,3 are external (public GitHub) repositories.

Multiple repositories were chosen for the experiment to cover a range of coding tasks. These included internal proprietary repositories from a large product company and publicly available repositories from GitHub. Each repository varied in size and complexity, ensuring a robust test environment. The tasks were designed to reflect real-world software development challenges, such as migrating from one logging framework to another or making a series of code edits dictated by evolving requirements or dependency updates.

2. CodePlan Operations

CodePlan was configured with initial seed edits, which are the starting point for each task. These edits are predefined modifications that simulate typical changes a developer might make, such as updating method calls to a new API version. The system then generated a plan based on these seed edits, using its planning algorithm to determine all necessary subsequent edits throughout the repository. This planning accounted for dependencies and potential impacts across the repository's codebase. For each planned edit, CodePlan used an LLM to generate the actual code changes. The LLM was provided with specific prompts that included relevant code context, ensuring that the suggested edits were applicable and syntactically correct.

3. Evaluation Metrics

CodePlan: Repository-level Coding using LLMs and Planning

Dataset	Approach	Matched Blocks	Missed Blocks	Spurious Blocks	Diff BLEU	Levenshtein Distance	Validity Check
C# Migration Task on Internal (Proprietary) Repositories							
Int-1 (Logging)	CodePlan (Iter 1)	151	0	0	0.99	60	✗ (4)
	CodePlan (Iter 2)	151	0	0	1.00	0	✓ (0)
	Build-Repair	82	69	13	0.81	6465	✗ (46)
Int-2 (Logging)	CodePlan (Iter 1)	438	0	0	0.99	90	✗ (6)
	CodePlan (Iter 2)	438	0	0	1.00	0	✓ (0)
	Build-Repair	337	101	25	0.66	7496	✗ (68)
C# Migration Task on External (Public) Repositories							
Ext-1 (NUnit-XUnit)	CodePlan (Iter 1)	58	0	0	1.00	0	✓ (0)
	Build-Repair	52	6	1	0.94	530	✗ (8)
Python Temporal Edit Task on External (Public) Repositories							
T-1	CodePlan (Iter 1)	8	2	0	0.90	1044	✗
	Pyright-Repair	5	5	0	0.76	1090	✗
	Pyright-Strict-Repair	8	2	0	0.90	1045	✗
	Coeditor-CodePlan	8	2	0	0.90	1160	✗
	Coeditor-Pyright-Repair	5	5	0	0.66	1205	✗
	Coeditor-Pyright-Strict-Repair	5	5	0	0.65	1139	✗
T-2	CodePlan (Iter 1)	4	0	0	0.86	248	✓
	Pyright-Repair	1	3	0	0.00	344	✗
	Pyright-Strict-Repair	1	3	0	0.00	344	✗
	Coeditor-CodePlan (Iter 1)	3	1	0	0.82	254	✗
	Coeditor-Pyright-Repair	1	3	0	0.00	344	✗
	Coeditor-Pyright-Strict-Repair	1	3	0	0.00	344	✗
T-3	CodePlan (Iter 1)	11	0	0	0.92	358	✓
	Pyright-Repair	1	10	0	0.00	798	✗
	Pyright-Strict-Repair	1	10	0	0.00	798	✗
	Coeditor-CodePlan (Iter 1)	10	1	0	0.78	717	✗
	Coeditor-Pyright-Repair	1	10	0	0.00	798	✗
	Coeditor-Pyright-Strict-Repair	1	10	0	0.00	798	✗

Table 3. Comparison of CodePlan’s repository edit metrics with Build-Repair baseline. Higher values of Matched Blocks, Diff BLEU, and lower values of Missed Blocks, Spurious Blocks, Levenshtein Distances are better.

The primary metrics for evaluation were the correctness and completeness of the code changes. Correctness was measured by the ability of the repositories to compile and pass tests after the edits. Completeness was assessed by comparing the final state of the repository against a ground truth of expected outcomes after all edits were applied. CodePlan’s performance was compared against baseline methods that did not use the planning approach. These baseline methods were typical LLM-driven code editors that handled each edit independently without considering broader repository-level impacts.

4. Results and Analysis

CodePlan demonstrated a superior ability to handle complex coding tasks correctly and completely across repositories. It successfully managed dependencies and interactions between different parts of the codebase, significantly reducing errors compared to the baselines.

The experiment also tested CodePlan's adaptability by introducing changes in the tasks and observing how well the system adjusted its planning and execution strategy. This included handling unexpected changes and errors introduced during the coding process.

In Conclusion, the results of the experiment validated the efficacy of using a planning-based approach with LLM integration for managing large-scale, complex coding tasks in software repositories. CodePlan not only outperformed traditional LLM-based code editors but also demonstrated its potential to significantly automate and improve the software development lifecycle. This experiment highlights the advantages of combining AI-driven planning with code generation technology to enhance productivity and accuracy in software engineering tasks.

Workshops: -

Along with studying research paper I have also attended different sessions like:-



This session was hosted by the computer science club at Brock university. I found the engineering session on promoting to be very informative. We focused on the basics of promoting and had a hands-on experience. We learned about the history of promoting, how to use them effectively, and how to create prompts that achieve the desired results. During the session, we used different LLMs to craft prompts and get various answers. We then compared our answers to check the accuracy of each other's promotes. The session also covered different types of promoting techniques like one-shot and two-shot promoting, which were used in some of the research papers I have mentioned above.

PROJECT

For this project, I have created four different classes that interact with OpenAI's GPT-3 Model to perform different tasks like parsing and it generates a basic level of documentation based on the given input and stores its output in an Excel file. below I have explained the functionality of the different classes:-

The Parser Class:- is used for parsing Java source files and extracting information about classes and methods for documentation generation. It utilizes the Java Compiler API to process Java files and the Java Compiler Tree API to traverse the abstract syntax tree (AST) of the parsed files. The extracted information is then used to generate documentation in CSV format.

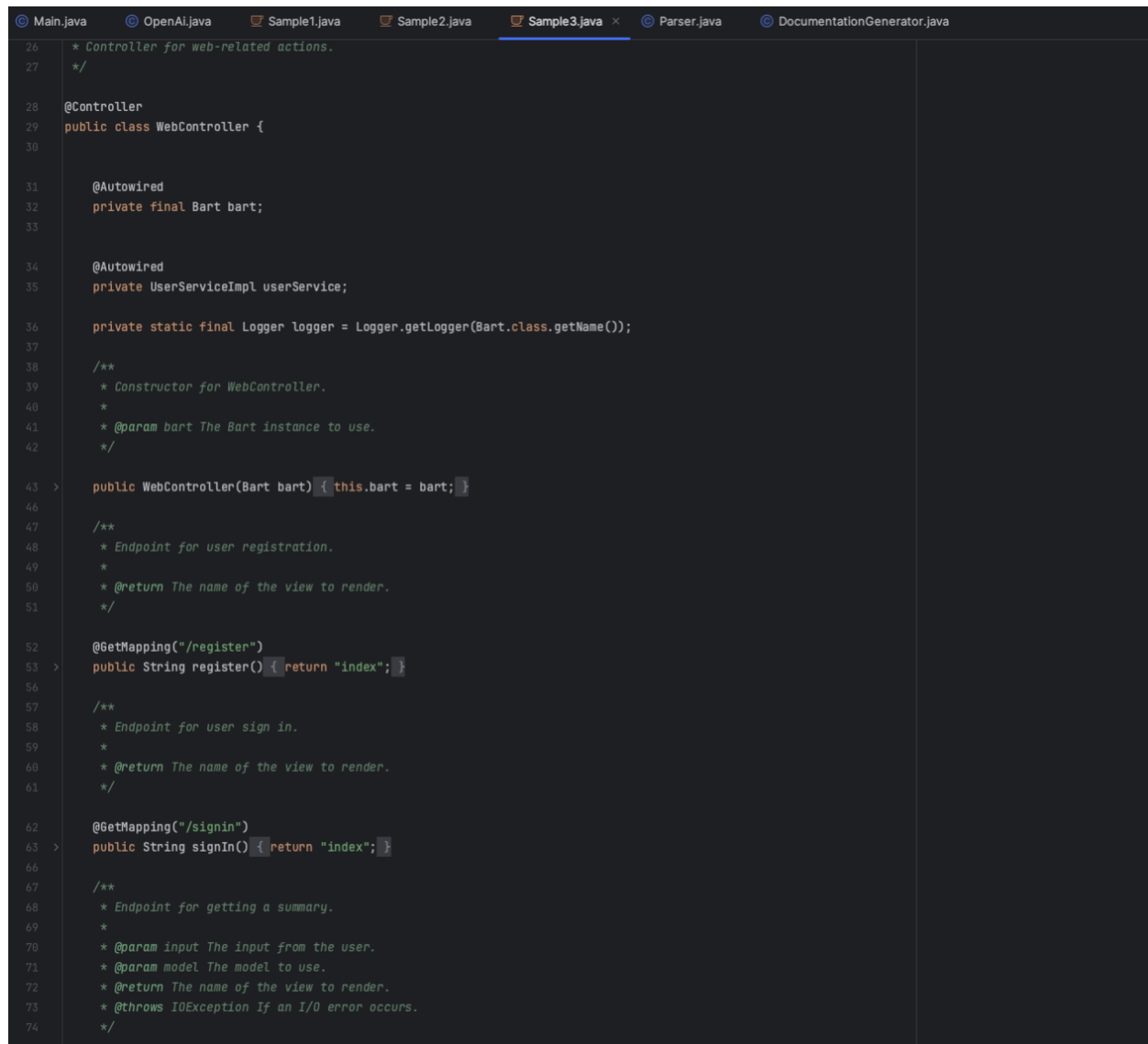
The OpenAI Class:- is used controller for interacting with the OpenAI API. It is annotated with `@RestController`, indicating that it handles web requests. The class includes methods for querying the OpenAI model with prompts and retrieving the responses as JSON objects. The OpenAI model is utilized to generate text completions or responses based on the provided prompts.

The Main Class:- serves as the entry point for the application. It contains the main method, which initializes and runs various components of the project. This includes creating new instances of the Parser class to parse Java source files and interacting with the OpenAI API to query the AI model.

The documentation generator Class:- is used to generate CSV documentation from provided data. Its `generateCSV()` method accepts a list of String arrays, formats them as comma-separated lines, and writes them to a file named "documentation.csv". It handles file writing and error handling internally, providing a streamlined approach for generating documentation.

The following images illustrate the input code (Sample3.java) and the corresponding output stored in the CSV file for our program. We have included some sample test codes in the `sampleTestCodes` folder which we used to test the program.

	A	B	C	D	E
1	Class Name	Method Name	Return Type	Parameters	
2	WebController	<init>	Constructor	Bart bart	
3	WebController	register	String		
4	WebController	signIn	String		
5	WebController	getSummary	String	@RequestParam(value = "input")	
6	String input	Model model			
7	WebController	createUser	String	@ModelAttribute	
8	User user	HttpSession session			
9	WebController	isValidURL	boolean	String urlStr	
10					
11					
12					
13					



```
26  * Controller for web-related actions.
27  */
28  @Controller
29  public class WebController {
30
31      @Autowired
32      private final Bart bart;
33
34      @Autowired
35      private UserServiceImpl userService;
36
37      private static final Logger logger = Logger.getLogger(Bart.class.getName());
38
39      /**
40       * Constructor for WebController.
41       * @param bart The Bart instance to use.
42       */
43  > public WebController(Bart bart) { this.bart = bart; }
44
45      /**
46       * Endpoint for user registration.
47       * @return The name of the view to render.
48       */
49
50      @GetMapping("/register")
51  > public String register() { return "index"; }
52
53      /**
54       * Endpoint for user sign in.
55       * @return The name of the view to render.
56       */
57
58      @GetMapping("/signin")
59  > public String signIn() { return "index"; }
60
61      /**
62       * Endpoint for getting a summary.
63       * @param input The input from the user.
64       * @param model The model to use.
65       * @return The name of the view to render.
66       * @throws IOException If an I/O error occurs.
67       */
68
69      // ...
70
71      // ...
72
73      // ...
74
75      // ...
76
77      // ...
78
79      // ...
80
81      // ...
82
83      // ...
84
85      // ...
86
87      // ...
88
89      // ...
90
91      // ...
92
93      // ...
94
95      // ...
96
97      // ...
98
99      // ...
100
101      // ...
102
103      // ...
104
105      // ...
106
107      // ...
108
109      // ...
110
111      // ...
112
113      // ...
114
115      // ...
116
117      // ...
118
119      // ...
120
121      // ...
122
123      // ...
124
125      // ...
126
127      // ...
128
129      // ...
130
131      // ...
132
133      // ...
134
135      // ...
136
137      // ...
138
139      // ...
140
141      // ...
142
143      // ...
144
145      // ...
146
147      // ...
148
149      // ...
150
151      // ...
152
153      // ...
154
155      // ...
156
157      // ...
158
159      // ...
160
161      // ...
162
163      // ...
164
165      // ...
166
167      // ...
168
169      // ...
170
171      // ...
172
173      // ...
174
175      // ...
176
177      // ...
178
179      // ...
180
181      // ...
182
183      // ...
184
185      // ...
186
187      // ...
188
189      // ...
190
191      // ...
192
193      // ...
194
195      // ...
196
197      // ...
198
199      // ...
200
201      // ...
202
203      // ...
204
205      // ...
206
207      // ...
208
209      // ...
210
211      // ...
212
213      // ...
214
215      // ...
216
217      // ...
218
219      // ...
220
221      // ...
222
223      // ...
224
225      // ...
226
227      // ...
228
229      // ...
230
231      // ...
232
233      // ...
234
235      // ...
236
237      // ...
238
239      // ...
240
241      // ...
242
243      // ...
244
245      // ...
246
247      // ...
248
249      // ...
250
251      // ...
252
253      // ...
254
255      // ...
256
257      // ...
258
259      // ...
260
261      // ...
262
263      // ...
264
265      // ...
266
267      // ...
268
269      // ...
270
271      // ...
272
273      // ...
274
275      // ...
276
277      // ...
278
279      // ...
280
281      // ...
282
283      // ...
284
285      // ...
286
287      // ...
288
289      // ...
290
291      // ...
292
293      // ...
294
295      // ...
296
297      // ...
298
299      // ...
300
301      // ...
302
303      // ...
304
305      // ...
306
307      // ...
308
309      // ...
310
311      // ...
312
313      // ...
314
315      // ...
316
317      // ...
318
319      // ...
320
321      // ...
322
323      // ...
324
325      // ...
326
327      // ...
328
329      // ...
330
331      // ...
332
333      // ...
334
335      // ...
336
337      // ...
338
339      // ...
340
341      // ...
342
343      // ...
344
345      // ...
346
347      // ...
348
349      // ...
350
351      // ...
352
353      // ...
354
355      // ...
356
357      // ...
358
359      // ...
360
361      // ...
362
363      // ...
364
365      // ...
366
367      // ...
368
369      // ...
370
371      // ...
372
373      // ...
374
375      // ...
376
377      // ...
378
379      // ...
380
381      // ...
382
383      // ...
384
385      // ...
386
387      // ...
388
389      // ...
390
391      // ...
392
393      // ...
394
395      // ...
396
397      // ...
398
399      // ...
400
401      // ...
402
403      // ...
404
405      // ...
406
407      // ...
408
409      // ...
410
411      // ...
412
413      // ...
414
415      // ...
416
417      // ...
418
419      // ...
420
421      // ...
422
423      // ...
424
425      // ...
426
427      // ...
428
429      // ...
430
431      // ...
432
433      // ...
434
435      // ...
436
437      // ...
438
439      // ...
440
441      // ...
442
443      // ...
444
445      // ...
446
447      // ...
448
449      // ...
450
451      // ...
452
453      // ...
454
455      // ...
456
457      // ...
458
459      // ...
460
461      // ...
462
463      // ...
464
465      // ...
466
467      // ...
468
469      // ...
470
471      // ...
472
473      // ...
474
475      // ...
476
477      // ...
478
479      // ...
480
481      // ...
482
483      // ...
484
485      // ...
486
487      // ...
488
489      // ...
490
491      // ...
492
493      // ...
494
495      // ...
496
497      // ...
498
499      // ...
500
501      // ...
502
503      // ...
504
505      // ...
506
507      // ...
508
509      // ...
510
511      // ...
512
513      // ...
514
515      // ...
516
517      // ...
518
519      // ...
520
521      // ...
522
523      // ...
524
525      // ...
526
527      // ...
528
529      // ...
530
531      // ...
532
533      // ...
534
535      // ...
536
537      // ...
538
539      // ...
540
541      // ...
542
543      // ...
544
545      // ...
546
547      // ...
548
549      // ...
550
551      // ...
552
553      // ...
554
555      // ...
556
557      // ...
558
559      // ...
560
561      // ...
562
563      // ...
564
565      // ...
566
567      // ...
568
569      // ...
570
571      // ...
572
573      // ...
574
575      // ...
576
577      // ...
578
579      // ...
580
581      // ...
582
583      // ...
584
585      // ...
586
587      // ...
588
589      // ...
590
591      // ...
592
593      // ...
594
595      // ...
596
597      // ...
598
599      // ...
600
601      // ...
602
603      // ...
604
605      // ...
606
607      // ...
608
609      // ...
610
611      // ...
612
613      // ...
614
615      // ...
616
617      // ...
618
619      // ...
620
621      // ...
622
623      // ...
624
625      // ...
626
627      // ...
628
629      // ...
630
631      // ...
632
633      // ...
634
635      // ...
636
637      // ...
638
639      // ...
640
641      // ...
642
643      // ...
644
645      // ...
646
647      // ...
648
649      // ...
650
651      // ...
652
653      // ...
654
655      // ...
656
657      // ...
658
659      // ...
660
661      // ...
662
663      // ...
664
665      // ...
666
667      // ...
668
669      // ...
670
671      // ...
672
673      // ...
674
675      // ...
676
677      // ...
678
679      // ...
680
681      // ...
682
683      // ...
684
685      // ...
686
687      // ...
688
689      // ...
690
691      // ...
692
693      // ...
694
695      // ...
696
697      // ...
698
699      // ...
700
701      // ...
702
703      // ...
704
705      // ...
706
707      // ...
708
709      // ...
710
711      // ...
712
713      // ...
714
715      // ...
716
717      // ...
718
719      // ...
720
721      // ...
722
723      // ...
724
725      // ...
726
727      // ...
728
729      // ...
730
731      // ...
732
733      // ...
734
735      // ...
736
737      // ...
738
739      // ...
740
741      // ...
742
743      // ...
744
745      // ...
746
747      // ...
748
749      // ...
750
751      // ...
752
753      // ...
754
755      // ...
756
757      // ...
758
759      // ...
760
761      // ...
762
763      // ...
764
765      // ...
766
767      // ...
768
769      // ...
770
771      // ...
772
773      // ...
774
775      // ...
776
777      // ...
778
779      // ...
780
781      // ...
782
783      // ...
784
785      // ...
786
787      // ...
788
789      // ...
790
791      // ...
792
793      // ...
794
795      // ...
796
797      // ...
798
799      // ...
800
801      // ...
802
803      // ...
804
805      // ...
806
807      // ...
808
809      // ...
810
811      // ...
812
813      // ...
814
815      // ...
816
817      // ...
818
819      // ...
820
821      // ...
822
823      // ...
824
825      // ...
826
827      // ...
828
829      // ...
830
831      // ...
832
833      // ...
834
835      // ...
836
837      // ...
838
839      // ...
840
841      // ...
842
843      // ...
844
845      // ...
846
847      // ...
848
849      // ...
850
851      // ...
852
853      // ...
854
855      // ...
856
857      // ...
858
859      // ...
860
861      // ...
862
863      // ...
864
865      // ...
866
867      // ...
868
869      // ...
870
871      // ...
872
873      // ...
874
875      // ...
876
877      // ...
878
879      // ...
880
881      // ...
882
883      // ...
884
885      // ...
886
887      // ...
888
889      // ...
890
891      // ...
892
893      // ...
894
895      // ...
896
897      // ...
898
899      // ...
900
901      // ...
902
903      // ...
904
905      // ...
906
907      // ...
908
909      // ...
910
911      // ...
912
913      // ...
914
915      // ...
916
917      // ...
918
919      // ...
920
921      // ...
922
923      // ...
924
925      // ...
926
927      // ...
928
929      // ...
930
931      // ...
932
933      // ...
934
935      // ...
936
937      // ...
938
939      // ...
940
941      // ...
942
943      // ...
944
945      // ...
946
947      // ...
948
949      // ...
950
951      // ...
952
953      // ...
954
955      // ...
956
957      // ...
958
959      // ...
960
961      // ...
962
963      // ...
964
965      // ...
966
967      // ...
968
969      // ...
970
971      // ...
972
973      // ...
974
975      // ...
976
977      // ...
978
979      // ...
980
981      // ...
982
983      // ...
984
985      // ...
986
987      // ...
988
989      // ...
990
991      // ...
992
993      // ...
994
995      // ...
996
997      // ...
998
999      // ...
1000
1001      // ...
1002
1003      // ...
1004
1005      // ...
1006
1007      // ...
1008
1009      // ...
1010
1011      // ...
1012
1013      // ...
1014
1015      // ...
1016
1017      // ...
1018
1019      // ...
1020
1021      // ...
1022
1023      // ...
1024
1025      // ...
1026
1027      // ...
1028
1029      // ...
1030
1031      // ...
1032
1033      // ...
1034
1035      // ...
1036
1037      // ...
1038
1039      // ...
1040
1041      // ...
1042
1043      // ...
1044
1045      // ...
1046
1047      // ...
1048
1049      // ...
1050
1051      // ...
1052
1053      // ...
1054
1055      // ...
1056
1057      // ...
1058
1059      // ...
1060
1061      // ...
1062
1063      // ...
1064
1065      // ...
1066
1067      // ...
1068
1069      // ...
1070
1071      // ...
1072
1073      // ...
1074
1075      // ...
1076
1077      // ...
1078
1079      // ...
1080
1081      // ...
1082
1083      // ...
1084
1085      // ...
1086
1087      // ...
1088
1089      // ...
1090
1091      // ...
1092
1093      // ...
1094
1095      // ...
1096
1097      // ...
1098
1099      // ...
1100
1101      // ...
1102
1103      // ...
1104
1105      // ...
1106
1107      // ...
1108
1109      // ...
1110
1111      // ...
1112
1113      // ...
1114
1115      // ...
1116
1117      // ...
1118
1119      // ...
1120
1121      // ...
1122
1123      // ...
1124
1125      // ...
1126
1127      // ...
1128
1129      // ...
1130
1131      // ...
1132
1133      // ...
1134
1135      // ...
1136
1137      // ...
1138
1139      // ...
1140
1141      // ...
1142
1143      // ...
1144
1145      // ...
1146
1147      // ...
1148
1149      // ...
1150
1151      // ...
1152
1153      // ...
1154
1155      // ...
1156
1157      // ...
1158
1159      // ...
1160
1161      // ...
1162
1163      // ...
1164
1165      // ...
1166
1167      // ...
1168
1169      // ...
1170
1171      // ...
1172
1173      // ...
1174
1175      // ...
1176
1177      // ...
1178
1179      // ...
1180
1181      // ...
1182
1183      // ...
1184
1185      // ...
1186
1187      // ...
1188
1189      // ...
1190
1191      // ...
1192
1193      // ...
1194
1195      // ...
1196
1197      // ...
1198
1199      // ...
1200
1201      // ...
1202
1203      // ...
1204
1205      // ...
1206
1207      // ...
1208
1209      // ...
1210
1211      // ...
1212
1213      // ...
1214
1215      // ...
1216
1217      // ...
1218
1219      // ...
1220
1221      // ...
1222
1223      // ...
1224
1225      // ...
1226
1227      // ...
1228
1229      // ...
1230
1231      // ...
1232
1233      // ...
1234
1235      // ...
1236
1237      // ...
1238
1239      // ...
1240
1241      // ...
1242
1243      // ...
1244
1245      // ...
1246
1247      // ...
1248
1249      // ...
1250
1251      // ...
1252
1253      // ...
1254
1255      // ...
1256
1257      // ...
1258
1259      // ...
1260
1261      // ...
1262
1263      // ...
1264
1265      // ...
1266
1267      // ...
1268
1269      // ...
1270
1271      // ...
1272
1273      // ...
1274
1275      // ...
1276
1277      // ...
1278
1279      // ...
1280
1281      // ...
1282
1283      // ...
1284
1285      // ...
1286
1287      // ...
1288
1289      // ...
1290
1291      // ...
1292
1293      // ...
1294
1295      // ...
1296
1297      // ...
1298
1299      // ...
1300
1301      // ...
1302
1303      // ...
1304
1305      // ...
1306
1307      // ...
1308
1309      // ...
1310
1311      // ...
1312
1313      // ...
1314
1315      // ...
1316
1317      // ...
1318
1319      // ...
1320
1321      // ...
1322
1323      // ...
1324
1325      // ...
1326
1327      // ...
1328
1329      // ...
1330
1331      // ...
1332
1333      // ...
1334
1335      // ...
1336
1337      // ...
1338
1339      // ...
1340
1341      // ...
1342
1343      // ...
1344
1345      // ...
1346
1347      // ...
1348
1349      // ...
1350
1351      // ...
1352
1353      // ...
1354
1355      // ...
1356
1357      // ...
1358
1359      // ...
1360
1361      // ...
1362
1363      // ...
1364
1365      // ...
1366
1367      // ...
1368
1369      // ...
1370
1371      // ...
1372
1373      // ...
1374
1375      // ...
1376
1377      // ...
1378
1379      // ...
1380
1381      // ...
1382
1383      // ...
1384
1385      // ...
1386
1387      // ...
1388
1389      // ...
1390
1391      // ...
1392
1393      // ...
1394
1395      // ...
1396
1397      // ...
1398
1399      // ...
1400
1401      // ...
1402
1403      // ...
1404
1405      // ...
1406
1407      // ...
1408
1409      // ...
1410
1411      // ...
1412
1413      // ...
1414
1415      // ...
1416
1417      // ...
1418
1419      // ...
1420
1421      // ...
1422
1423      // ...
1424
1425      // ...
1426
1427      // ...
1428
1429      // ...
1430
1431      // ...
1432
1433      // ...
1434
1435      // ...
1436
1437      // ...
1438
1439      // ...
1440
1441      // ...
1442
1443      // ...
1444
1445      // ...
1446
1447      // ...
1448
1449      // ...
1450
1451      // ...
1452
1453      // ...
1454
1455      // ...
1456
1457      // ...
1458
1459      // ...
1460
1461      // ...
1462
1463      // ...
1464
1465      // ...
1466
1467      // ...
1468
1469      // ...
1470
1471      // ...
1472
1473      // ...
1474
1475      // ...
1476
1477      // ...
1478
1479      // ...
1480
1481      // ...
1482
1483      // ...
1484
1485      // ...
1486
1487      // ...
1488
1489      // ...
1490
1491      // ...
1492
1493      // ...
1494
1495      // ...
1496
1497      // ...
1498
1499      // ...
1500
1501      // ...
1502
1503      // ...
1504
1505      // ...
1506
1507      // ...
1508
1509      // ...
1510
1511      // ...
1512
1513      // ...
1514
1515      // ...
1516
1517      // ...
1518
1519      // ...
1520
1521      // ...
1522
1523      // ...
1524
1525      // ...
1526
1527      // ...
1528
1529      // ...
1530
1531      // ...
1532
1533      // ...
1534
1535      // ...
1536
1537      // ...
1538
1539      // ...
1540
1541      // ...
1542
1543      // ...
1544
1545      // ...
1546
1547      // ...
1548
1549      // ...
1550
1551      // ...
1552
1553      // ...
1554
1555      // ...
1556
1557      // ...
1558
1559      // ...
1560
1561      // ...
1562
1563      // ...
1564
1565      // ...
1566
1567      // ...
1568
1569      // ...
1570
1571      // ...
1572
1573      // ...
1574
1575      // ...
1576
1577      // ...
1578
1579      // ...
1580
1581      // ...
1582
1583      // ...
1584
1585      // ...
1586
1587      // ...
1588
1589      // ...
1590
1591      // ...
1592
1593      // ...
1594
1595      // ...
1596
1597      // ...
1598
1599      // ...
1600
1601      // ...
1602
1603      // ...
1604
1605      // ...
1606
1607      // ...
1608
1609      // ...
1610
1611      // ...
1612
1613      // ...
1614
1615      // ...
1616
1617      // ...
1618
1619      // ...
1620
1621      // ...
1622
1623      // ...
1624
1625      // ...
1626
1627      // ...
1628
1629      // ...
1630
1631      // ...
1632
1633      // ...
1634
1635      // ...
1636
1637      // ...
1638
1639      // ...
1640
1641      // ...
1642
1643      // ...
1644
1645      // ...
1646
1647      // ...
1648
1649      // ...
1650
1651      // ...
1652
1653      // ...
1654
1655      // ...
1656
1657      // ...
1658
1659      // ...
1660
1661      // ...
1662
1663      // ...
1664
1665      // ...
1666
1667      // ...
1668
1669      // ...
1670
1671      // ...
1672
1673      // ...
1674
1675      // ...
1676
1677      // ...
1678
1679      // ...
1680
1681      // ...
1682
1683      // ...
1684
1685      // ...
1686
1687      // ...
1688
1689      // ...
1690
1691      // ...
1692
1693      // ...
1694
1695      // ...
1696
1697      // ...
1698
1699      // ...
1700
1701      // ...
1702
1703      // ...
1704
1705      // ...
1706
1707      // ...
1708
1709      // ...
1710
1711      // ...
1712
1713      // ...
1714
1715      // ...
1716
1717      // ...
1718
1719      // ...
1720
1721      // ...
1722
1723      // ...
1724
1725      // ...
1726
1727      // ...
1728
1729      // ...
1730
1731      // ...
1732
1733      // ...
1734
1735      // ...
1736
1737      // ...
1738
1739      // ...
1740
1741      // ...
1742
1743      // ...
1744
1745      // ...
1746
1747      // ...
1748
1749      // ...
1750
1751      // ...
1752
1753      // ...
1754
1755      // ...
1756
1757      // ...
1758
1759      // ...
1760
1761      // ...
1762
1763      // ...
1764
1765      // ...
1766
1767      // ...
1768
1769      // ...
1770
1771      // ...
1772
1773      // ...
1774
1775      // ...
1776
1777      // ...
1778
1779      // ...
1780
1781      // ...
1782
1783      // ...
1784
1785      // ...
1786
1787      // ...
1788
1789      // ...
1790
1791      // ...
1792
1793      // ...
1794
1795      // ...
1796
1797      // ...
1798
1799      // ...
1800
1801      // ...
1802
1803      // ...
1804
1805      // ...
1806
1807      // ...
1808
1809      // ...
1810
1811      // ...
1812
1813      // ...
1814
1815      // ...
1816
1817      // ...
1818
1819      // ...
1820
1821      // ...
1822
1823      // ...
1824
1825      // ...
1826
1827      // ...
1828
1829      // ...
1830
1831      // ...
1832
1833      // ...
1834
1835      // ...
1836
1837      // ...
1838
1839      // ...
1840
1841      // ...
1842
1843      // ...
1844
1845      // ...
1846
1847      // ...
1848
1849      // ...
1850
1851      // ...
1852
1853      // ...
1854
1855      // ...
1856
1857      // ...
1858
1859      // ...
1860
1861      // ...
1862
1863      // ...
1864
1865      // ...
1866
1867      // ...
1868
1869      // ...
1870
1871      // ...
1872
1873      // ...
1874
1875      // ...
1876
1877      // ...
1878
1879      // ...
1880
1881      // ...
1882
1883      // ...
1884
1885      // ...
1886
1887      // ...
1888
1889      // ...
1890
1891      // ...
1892
1893      // ...
1894
1895      // ...
1896
1897      // ...
1898
1899      // ...
1900
1901      // ...
1902
1903      // ...
1904
1905      // ...
1906
1907      // ...
1908
1909      // ...
1910
1911      // ...
1912
1913      // ...
1914
1915      // ...
1916
1917      // ...
1918
1919      // ...
1920
1921      // ...
1922
1923      // ...
1924
1925      // ...
1926
1927      // ...
1928
1929      // ...
1930
1931      // ...
1932
1933      // ...
1934
1935      // ...
1936
1937      // ...
1938
1939      // ...
1940
1941      // ...
1942
1943      // ...
1944
1945      // ...
1946
1947      // ...
1948
1949      // ...
1950
1951      // ...
1952
1953      // ...
1954
1955      // ...
1956
1957      // ...
1958
1959      // ...
1960
1961      // ...
1962
1963      // ...
1964
1965      // ...
1966
1967      // ...
1968
1969      // ...
1970
1971      // ...
1972
1973      // ...
1974
1975      // ...
1976
1977      // ...
1978
1979      // ...
1980
1981      // ...
1982
1983      // ...
1984
1985      // ...
1986
1987      // ...
1988
1989      // ...
1990
1991      // ...
1992
1993      // ...
1994
1995      // ...
1996
1997      // ...
1998
1999      // ...
2000
2001      // ...
2002
2003      // ...
2004
2005      // ...
2006
2007      // ...
2008
2009      // ...
2010
2011      // ...
2012
2013      // ...
2014
2015      // ...
2016
2017      // ...
2018
2019      // ...
2020
2021      // ...
2022
2023      // ...
2024
2025      // ...
2026
2027      // ...
2028
2029      // ...
2030
2031      // ...
2032
2033      // ...
2034
2035      // ...
2036
2037      // ...
2038
2039      // ...
2040
2041      // ...
2042
2043      // ...
2044
2045      // ...
2046
2047      // ...
2048
2049      // ...
2050
2051      // ...
205
```

REFERENCES

- [1] Mandvikar, S. (2023). Augmenting intelligent document processing (IDP) workflows with contemporary large language models (llms). *International Journal of Computer Trends and Technology*, 71(10), 80–91. <https://doi.org/10.14445/22312803/ijctt-v71i10p110>
- [2] Khan, J. Y., & Uddin, G. (2022). Automatic code documentation generation using GPT-3. *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. <https://doi.org/10.1145/3551349.3559548>
- [3] C. Saben and P. Chandrasekar, “Enabling BLV Developers with LLM-driven Code Debugging,” in *Proceedings of the x ’24*, ACM, New York, NY, USA, 2018, pp. 1-8. DOI: XXXXXXXX.XXXXXXX.
- [4] R. Bairi, A. Sonwane, A. Kanade, V. D. C., A. Iyer, S. Parthasarathy, S. Rajamani, B. Ashok, and S. Shet, “CodePlan: Repository-level Coding using LLMs and Planning,” in *Proceedings of the x ’24*, ACM, New York, NY, USA, 2023, pp. 1-8. DOI: 10.48550/arXiv.2309.12499.
- [5] OpenAI. “ChatGPT: Fine-Tuned from GPT-3.5 for Dialogue.” In *Proceedings of the x ’24*, ACM, New York, NY, USA, 2024, pp. 1-8. DOI: 10.48550/arXiv.2405.67890.
- [6] Grammarly. “Free AI Writing Assistance.” In *Proceedings of the x ’24*, ACM, New York, NY, USA, 2024, pp. 1-8. DOI: 10.48550/arXiv.2405.67890.