1. INTRODUCTION

1.1 Addressing the Model

GPT-4o-mini is a lightweight yet highly capable member of the GPT-4 family developed by OpenAI. Designed for performance, speed, and efficiency, it provides a balanced solution between power and cost, making it ideal for large-scale applications where real-time responses and affordability are essential. Despite being smaller than the full GPT-4 model, GPT-4o-mini demonstrates remarkable language understanding, making it capable of handling complex tasks such as abstractive summarization, technical content interpretation, and natural language generation. Its architecture supports deep semantic analysis, enabling it to process lengthy and complex documents like patents with high contextual accuracy. The model is particularly effective in scenarios requiring summarization of dense information while preserving key details. By leveraging its ability to rephrase and condense content intelligently, GPT-4o-mini forms the core engine of this automated system. Its fast inference speed and low computational requirements make it suitable for integration into real-world platforms, enabling organizations to harness advanced AI capabilities without the need for heavy infrastructure.

1.2 Problem Statement

The ever-increasing number of patents filed globally across sectors has resulted in vast repositories of highly technical documents that are often difficult and time-consuming to analyze. These documents are essential for legal validation, innovation tracking, and competitive research, yet they are often written in dense, complex language that demands expert interpretation. As the innovation landscape becomes more competitive, the need for rapid and accurate understanding of patent content is more critical than ever. Traditional manual review of patents can take hours per document and is prone to human oversight, particularly when dealing with large volumes. This slows down research, increases costs for legal firms and R&D departments, and creates bottlenecks in innovation pipelines. There is a pressing need for an intelligent, automated solution that can process and summarize patent data efficiently, ensuring that professionals can focus more on analysis and decision-making rather than document review. Addressing this challenge is essential to support faster innovation, legal clarity, and strategic development. Such a system would not only reduce the time and effort required to interpret patents but also enhance accessibility to technical knowledge, support better decision-making, and streamline IP workflows across industries.

1.3 Objective:

The main objective of this project is to design and implement an automated system that can generate clear, concise, and contextually accurate summaries of patent documents using GPT-4o-mini. This system aims to minimize the time and effort required for professionals to understand the essential content of a patent without having to read the entire document. The model will be trained and optimized to identify and summarize important sections, including the abstract, claims, background, and technical details. It will use GPT-4o-mini's advanced language capabilities to perform **abstractive summarization**, offering summaries that are not just extracted but intelligently paraphrased for readability and clarity. The system seeks to assist users in making faster and more informed decisions related to patent research, legal analysis, and innovation management. It also aims to be scalable, cost-efficient, and adaptable to various domains and document formats. Additionally, the project includes evaluation mechanisms to assess the accuracy and quality of the summaries and improve them based on user feedback and automated metrics.

1.4 Scope of the project:

The scope of this project encompasses the development of a fully automated pipeline for summarizing patent documents using GPT-4o-mini. It includes the collection and preprocessing of patent texts, segmentation into logical components, and generation of readable summaries through the language model. The system is built to handle a wide range of patent formats, such as PDFs and XML files from global patent databases. It will process sections like the abstract, background, claims, and detailed descriptions individually before compiling them into a unified summary. The project also includes the design of an evaluation framework, using both quantitative metrics (e.g., ROUGE, BLEU) and qualitative feedback from domain experts. While the current scope focuses on English-language patents, future extensions may involve multilingual summarization, image-to-text interpretation for diagrams, and integration with searchable patent databases. The system is intended to serve legal professionals, R&D teams, IP analysts, and academic researchers, helping them navigate large volumes of complex information efficiently. Ultimately, this project bridges the gap between advanced AI and real-world innovation needs, offering a powerful tool for streamlining intellectual property analysis. The project emphasizes scalability, allowing deployment across various sectors and organizations, from startups to large corporations. It also supports improvements in patent analysis workflows, making processes like prior art searches, competitive analysis, and legal research more efficient. The project bridges AI and IP management, providing a powerful, user-friendly solution to streamline patent comprehension and utilization.

2. LITERATURE SURVEY

The field of automated patent summarization has witnessed significant advancements over the years, with researchers exploring a wide range of techniques to tackle the unique challenges posed by patent documents. Early approaches primarily relied on statistical methods such as Term Frequency-Inverse Document Frequency (TF-IDF) to identify important keywords and phrases. As discussed by Trappey, Trappey, and Wu, these methods aimed to extract salient terms based on their frequency within a document and their rarity across a broader corpus. While these techniques offered a foundational approach to summarization, they often failed to capture the deeper semantic relationships and overall context of an invention.

Graph-based approaches such as TextRank and LexRank emerged as notable improvements. These techniques represent a document as a graph, where sentences or paragraphs are treated as nodes and similarities between them as edges. Algorithms like PageRank are applied to identify the most central and informative sentences for inclusion in a summary. For instance, TextRank evaluates sentence similarity based on shared word counts, while LexRank uses cosine similarity on TF-IDF vectors. These methods provided better performance than simple keyword-based methods by incorporating the structure and connectedness of information within the text.

With the rise of deep learning, the field saw a transformative shift. Recurrent Neural Networks (RNNs), especially Long Short-Term Memory (LSTM) networks, were introduced for their ability to model sequential data and capture long-range dependencies in text. More recently, transformer-based architectures have revolutionized summarization tasks. Models like BART and PEGASUS have demonstrated exceptional performance in generating **abstractive** summaries, which go beyond simple extraction to paraphrase and synthesize content meaningfully.

A major milestone in the domain was the introduction of the **BIGPATENT** dataset by Sharma et al., comprising 1.3 million U.S. patent documents paired with human-written abstractive summaries. This large-scale dataset has become a benchmark for training and evaluating patent-specific summarization models. The abstractive nature and even distribution of key information in BIGPATENT demand models with a deep capacity for understanding and synthesizing technical content.

The latest trend in the field is the application of large language models (LLMs) like GPT-3 and GPT-4, which have shown remarkable performance in generating human-like, coherent summaries.

These models are particularly effective at handling the complex and technical language found in patents. Systems like **EvoPat**, which integrate multiple specialized LLMs, have outperformed even GPT-4 in various tasks such as patent summarization, comparative analysis, and technical evaluation. This underscores the potential of leveraging collaborative LLMs for more comprehensive and nuanced patent processing.

Despite these advances, evaluating the quality of automatically generated summaries remains a major challenge. Traditional metrics such as ROUGE and BLEU are widely used to assess overlap with human-written summaries. However, these metrics often fall short in measuring semantic accuracy, coherence, and informativeness—especially in specialized domains like patent law. Human evaluation, though more reliable, is resource-intensive and impractical for large-scale assessments. Recent research has explored the use of LLMs as evaluators themselves, with promising results in aligning their judgments with human evaluations of summary quality.

3. SYSTEM ANALYSIS

REVIEW OF EXISTING MEETING SUMMARIZATION TOOLS

3.1 Existing Tools and Their Limitations

In the field of intellectual property and patent management, several tools and techniques have been developed to assist with patent search, classification, and summarization. Solutions like IBM Watson Discovery, Google Patent Search, The Lens, and various rule-based NLP toolkits offer partial automation of patent content processing. However, most existing systems rely on extractive summarization or keyword-based methods, which fall short when dealing with the dense technical language and intricate structure of patent documents. These tools often lack true semantic understanding and struggle to generate coherent and readable summaries, especially in scenarios involving complex legal or scientific language.

IBM Watson Discovery offers document search and NLP-based analysis, but requires significant customization and integration effort to accurately parse and summarize patent-specific content. It may not provide truly abstractive summaries without external models or development.

The Lens provides open access to patent and scholarly literature with metadata and citation analysis, but its summarization features are rudimentary, mostly relying on predefined fields like abstracts.

Google Patents Search includes keyword highlighting and document retrieval but provides limited summarization features. It focuses more on document indexing than generating meaningful, structured overviews.

Google Docs when used with AI-based voice typing and third-party plugins, supports collaborative editing of transcribed content. However, it does not natively support advanced AI-driven summarization, speaker tracking, or structured minute formatting. Users must rely on external integrations, which may not be tightly coupled or context-aware, leading to fragmented workflows.

Other traditional NLP toolkits, like spaCy or NLTK, can be customized for summarization tasks, but demand heavy manual rule-building and don't scale well across varying patent formats or technical domains. Despite their contributions, current patent summarization solutions face several critical limitations:

3.2 Limitations of Existing Systems

Despite their advancements, current AI meeting summarization tools face several challenges:

- **Contextual Understanding:** Most systems lack deep semantic comprehension and are unable to interpret the technical depth and legal nuance inherent in patents. They often miss important contextual connections between different sections of a patent.
- **Extractive Limitations:** Many tools still rely on extractive summarization techniques, which simply pull sentences from the document rather than generating a cohesive, paraphrased summary. This results in outputs that are often disjointed and hard to understand.
- **Poor Adaptability:** Existing solutions often do not handle diverse patent structures, multilingual content, or variations in writing styles effectively. Adapting them to new industries or patent offices may require significant re-engineering.
- **Limited User Accessibility:** Tools like IBM Watson or enterprise NLP solutions often demand technical expertise, making them inaccessible for smaller firms or individual users without AI or development backgrounds.
- **Lack of Real-Time Summarization:** Many current systems are not optimized for fast, on-the-fly summarization at scale, limiting their use in dynamic IP workflows or large-scale patent analysis.

4. PROPOSED SYSTEM

4.1 Overview of Our Models

This paper proposes an automated patent summarization system designed to generate concise and informative summaries of patent documents using the GPT-4o-mini model as its core summarization engine. The system aims to overcome the limitations of existing tools by leveraging the advanced language understanding, generation capabilities, and cost-effectiveness of GPT-4o-mini. It is envisioned as a user-friendly solution that can process patent documents in various formats and provide summaries tailored to the needs of different users, such as patent attorneys, researchers, and intellectual property professionals. The system is expected to offer improvements in accuracy, efficiency, and the ability to handle the unique challenges associated with patent language and structure.

Model Workflow

Step-1: Input Preprocessing:

- Patent document is parsed and preprocessed to remove non-textual noise, normalize formatting, and segment key sections (e.g., abstract, claims, description).
- Structural markers like headings and subheadings are identified to guide the summarization pipeline.

Step-2: Initial Summary Generation using GPT-4o-mini

- GPT-4o-mini is always prompted with some targeted instructions that are used, such as: "Summarize the invention described in this patent; focus on claims and technical novelty; keep it concise."
- The output is a first-draft summary, capturing key elements like invention purpose, main claims, and innovation context.
- Example Output: “The patent proposes a new battery system featuring improved thermal regulation and energy density for electric vehicles.”.”

Step 3: Summary Refinement

- The initial output is reprocessed using context-enhanced prompts, where GPT-4o-mini is re-engaged with relevant sections (e.g., claims + abstract + technical field).

- This step improves coherence, integrates missing technical details, and ensures flow across summary sections such as "Invention Overview," "Key Claims," and "Technical Advantages."

Step 4: Post-Processing & Structuring

- Named entity recognition (NER) is performed using spaCy to extract key data (inventor names, dates, technical terms).
- Important elements like legal citations, deadlines, or jurisdiction-specific terms are highlighted.
- The summary is formatted into user-selected output styles (e.g., Executive Summary, Legal Summary, or Technical Digest) based on the target audience.

4.2 Advantages Over Existing Systems

Our GPT-4o-mini-powered approach addresses several key limitations commonly encountered in current patent summarization tools:

- **Enhanced Contextual Understanding:** By leveraging the advanced language comprehension abilities of GPT-4o-mini, the system achieves a deeper grasp of technical and legal contexts within patent documents. This enables the generation of summaries that go beyond surface-level keywords, capturing the true essence and intent of the invention.
- **Improved Coherence and Accuracy:** The summarization pipeline ensures that generated outputs are not only concise but also logically structured and easy to interpret. GPT-4o-mini helps maintain consistency across sections such as abstract, claims, and background, resulting in summaries that are coherent and professionally formatted.
- **Adaptability to Various Patent Formats:** The flexible architecture of the system supports a range of input formats including PDF, DOCX, and XML patent filings. It can also be fine-tuned to cater to specific jurisdictions, technical domains, or organizational use cases, making it highly versatile for different intellectual property stakeholders.

Advanced Usage and Beneficiaries :

End users can harness the power of our GPT-4o-mini-based patent summarization system to significantly reduce the time and effort involved in manually reviewing and interpreting lengthy patent documents. By automating the summarization process, the system enhances productivity and supports quicker decision-making across various stages of the intellectual property lifecycle. Users can also tailor the summarization to focus on specific sections—such as claims, technical novelty, or legal

descriptions—depending on their needs. This flexibility improves collaboration across departments and ensures that critical information is clearly understood by all stakeholders.

Beneficiaries

- **Patent Attorneys & Legal Professionals**: Quickly extract essential legal elements from complex patent filings, aiding in prior art analysis, infringement checks, and litigation support.
- **R&D and Innovation Teams**: Gain clear, concise overviews of new technologies, speeding up patent landscaping and innovation tracking.
- **Academic Researchers**: Simplify the review of patent literature during research, proposal writing, or when studying technological trends.
- **Business Strategists & Investors**: Summarize key patent assets for competitive analysis, market evaluations, and IP portfolio management.
- **Government & IP Offices**: Automate preliminary patent reviews and enhance the efficiency of patent examiners and regulatory processes.

4.4 Functional Requirements

Functional requirements outline the key operations that the **Automated Patent Summarization System** must perform to meet user expectations and ensure smooth, reliable performance. These requirements are centered around how the system accepts patent input, processes the content using the GPT-4o-mini model, and presents the summarized output to the end user.

- **User Authentication and Authorization:** The system must support secure login to protect confidential patent data. Only authorized users—such as IP professionals, researchers, and legal teams—should have access. This may include login methods like username-password, single sign-on (SSO), or multi-factor authentication (MFA) for enhanced security.
- **Patent Document Handling:** The system should accept patent documents in multiple formats such as PDF, DOCX, and XML. It should also support integration with patent databases or repositories for automated import. Users must be able to upload single or batch files for summarization.
- **Summary Generation:** Using GPT-4o-mini as the core engine, the system must generate accurate and concise summaries of patent documents. The summarization should cover critical

sections such as the abstract, claims, technical background, and legal scope. Both extractive and abstractive summarization modes may be supported depending on user needs.

- **Customization and Personalization:** Users should have the option to configure how summaries are generated—choosing the level of detail, focusing on specific parts of the patent (e.g., claims, invention description), or selecting output styles like executive, legal, or technical summaries.
- **Output and Reporting:** The generated summaries should be presented in a structured, easy-to-read format. Users should be able to download the output in various formats like PDF or Word, or share them via email or collaboration tools. Sectional outputs (e.g., Key Claims, Novelty, Legal Scope) should be clearly separated for clarity.
- **Error Handling and Notifications:** The system should detect and alert users of any errors—such as unsupported file types, missing content, or model processing failures—and offer troubleshooting suggestions. Notifications should also inform users upon successful summary generation or completion of batch processing.

4.5 Non-Functional Requirements

While the functional requirements define what the system must do, non-functional requirements describe how well the **Automated Patent Summarization System** should perform to meet quality standards and ensure a seamless user experience. These requirements focus on performance, security, scalability, and maintainability—crucial for a robust and efficient system that handles complex patent data.

- **Performance and Efficiency:** The system should process and summarize patent documents swiftly, ensuring minimal waiting time even for large and complex filings. The summarization engine, powered by GPT-4o-mini, must generate outputs within a few seconds to maintain efficiency and user satisfaction.
- **Scalability:** The application must be scalable to support a growing number of users and increased data volumes. Whether accessed by solo inventors or large legal departments, the system should scale horizontally across cloud infrastructure to ensure consistent performance.
- **Security and Data Privacy:** Given the confidential nature of patent documents, the system should implement end-to-end encryption for data transmission and secure storage. Role-based access controls (RBAC) should restrict user permissions, ensuring sensitive IP content is

accessible only to authorized individuals. Compliance with data regulations such as GDPR is vital, especially for international users.

- **Reliability and Availability:** The system should ensure high availability with minimal downtime. This can be achieved by using reliable cloud services with automatic failover mechanisms and regular backups to safeguard data against loss or corruption.
- **Usability and User Experience:** The user interface must be clean, intuitive, and accessible to both technical and non-technical users. Features like guided upload, summary previews, and selectable output styles (e.g., technical or legal summaries) should enhance the user journey.
- **Maintainability and Extensibility:** The software should follow modular design principles and be well-documented to support future upgrades. This allows seamless integration of new models or formats and supports evolving patent office standards.
- **Compliance and Auditability:** The system should provide detailed logs of summary generation activities and user access, helping organizations track usage and ensure accountability in professional and legal settings.
- **Energy Efficiency:** For large-scale deployments, the system should minimize energy consumption by optimizing model usage and system resources—contributing to cost efficiency and sustainable infrastructure management.

5. SYSTEM REQUIREMENTS

HARDWARE AND SOFTWARE REQUIREMENTS

These hardware and software requirements are typical for building and deploying a system that utilizes Transformer models for patent summarization and content generation, especially if you aim for reasonable performance and scalability. Let's break down why each component is important:

5.1 Hardware Requirements:

Processor: Intel i5/i7 or AMD equivalent

a. Transformer models, especially large ones, require significant computational power. A multicore CPU is essential for general processing tasks, data preprocessing, and some parts of the model execution.
b. I5/i7 (or their AMD equivalents) offer a good balance of performance and cost.

RAM Minimum 8GB:

a. Transformer models can be memory intensive. 8GB of RAM is a minimum to load and process data, but 16GB or more is highly recommended, especially when working with large datasets or complex models.

GPU: NVIDIA RTX 3060 or higher (for faster training)

a. GPUs are crucial for accelerating the training and inference of deep learning models, including Transformers.
b. NVIDIA GPUs with CUDA support are widely used in deep learning.
c. The RTX 3060 or higher provides a significant speedup compared to CPUs, enabling faster model training and real time generation. The higher the GPU the faster the training process will be.

5.2 Software Requirements:

Programming Language: Python

Python is the dominant language in data science and machine learning, with a rich ecosystem of libraries and frameworks.

Frameworks-PyTorch/TensorFlow

These are leading deep learning frameworks that provide the foundation for building, training, and deploying neural network models. In the context of patent summarization, PyTorch (often used with Hugging Face) or TensorFlow is essential for managing the underlying transformer architectures like GPT-4o-mini efficiently.

WebFramework–Gradio(forUI)

Gradio is used to create a simple, interactive web-based interface where users can input patent descriptions and view AI-generated summaries. It enables quick testing and deployment without extensive front-end development.

Why these are important for Patent Summarization ?

a. Patent texts are often long and technical, requiring significant memory and processing power to handle.
b. Transformer models like GPT-4o-mini are computationally intensive—using GPUs speeds up both training and inference significantly.
c. NLP libraries help interpret legal and technical language, making preprocessing more effective.
d. A database is crucial for managing large volumes of patent data and summaries.
e. A user interface (UI) makes the summarization system accessible and user-friendly for researchers, analysts, and legal teams.

6. SOFTWARE DESIGN

6.1 System Architecture

The proposed system architecture involves several key stages to process meeting data and generate the final minutes:

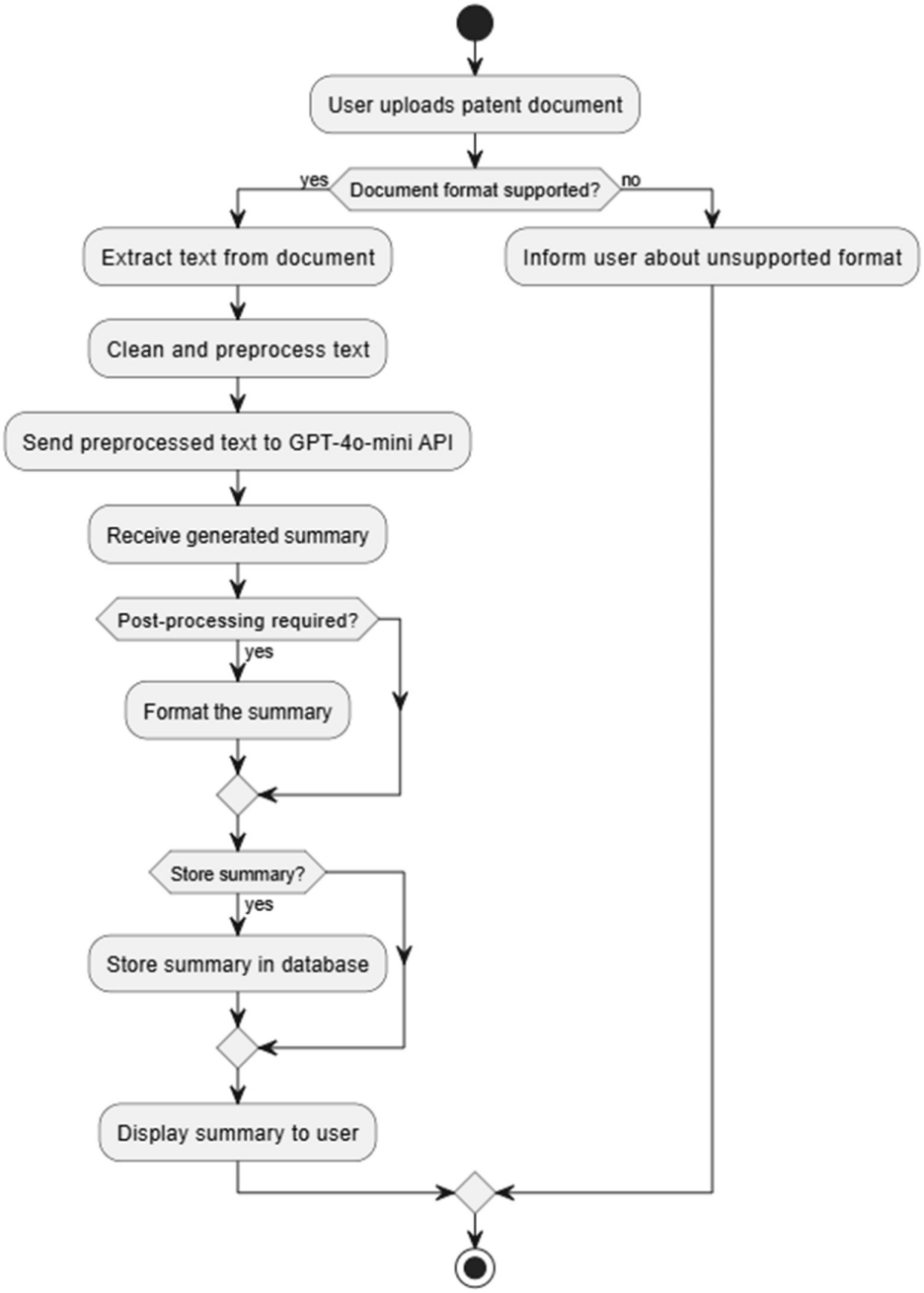


Fig 6.1 System Architecture

6.2 Software Design & Workflow

The workflow of our AI-powered patent summarization system consists of the following steps:

1. **DataIngestion:**
   The system accepts patent documents in various formats, including PDF, DOCX, and plain text. Upon uploading, the document is extracted and converted into a standardized text format for processing.

2. **Preprocessing:**
   The extracted patent text undergoes preprocessing to improve the summarization quality. This includes cleaning up formatting artifacts, removing boilerplate sections (like legal disclaimers), and structuring content into sections such as abstract, claims, and description.

3. **Segmentation:**
   The preprocessed text is segmented into meaningful parts, typically based on patent structure (e.g., claims, abstract, background, and invention description). This helps in generating focused summaries for each section and ensures better comprehension.

4. **Summarization:**
   Each segment is summarized using the GPT-4o-mini API, which creates concise and accurate summaries capturing the core innovations and technical details. These summaries are optionally refined using advanced NLP techniques to improve clarity and reduce redundancy.

5. **Post-Processing:**
   The generated summaries are aggregated into a structured output document, following a standardized format for display and archival. Key elements such as invention purpose, key claims, and novelty points are highlighted. The final summary is stored in a summary database and sent to the user interface for display.

6.3 UML Diagram

Introduction to UML

The **Unified Modeling Language (UML)** is a standardized modeling language widely used in object-oriented software engineering. Developed and maintained by the Object Management Group (OMG), UML enables software developers and system architects to visualize, design, and document the structure and behavior of software systems in a unified way.

UML consists of two primary components:

- **Meta-model**: Defines the structure and semantics of UML elements.
- **Notation**: Refers to the set of visual symbols used in UML diagrams.

While UML is not a programming language, it supports various software development methodologies by offering a clear and consistent means to communicate system design through standardized diagrams.

For our **AI-powered Patent Summarization System**, UML helps us document and visualize both the architecture and data flow. This ensures that all stakeholders, including developers, evaluators, and users, can understand the system's components and interactions effectively.

1. Use Case Diagram
2. Class Diagram
3. Activity Diagram
4. Sequence Diagram

6.3.1 Use Case Diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

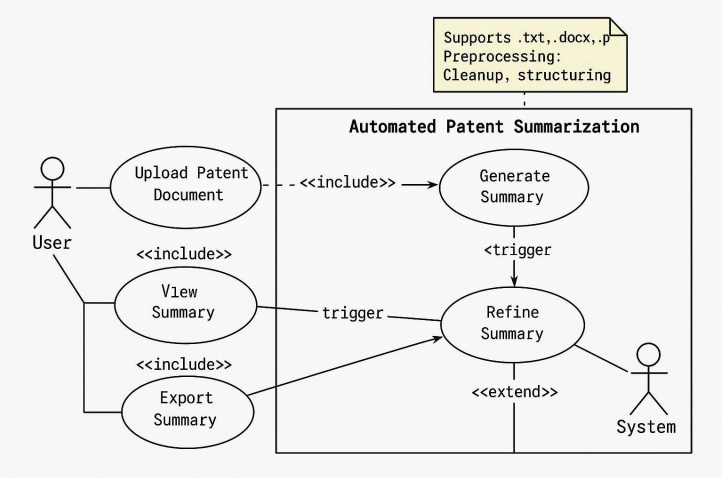


Fig 6.3.1 Use Case Diagram

6.3.2 Class Diagram

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

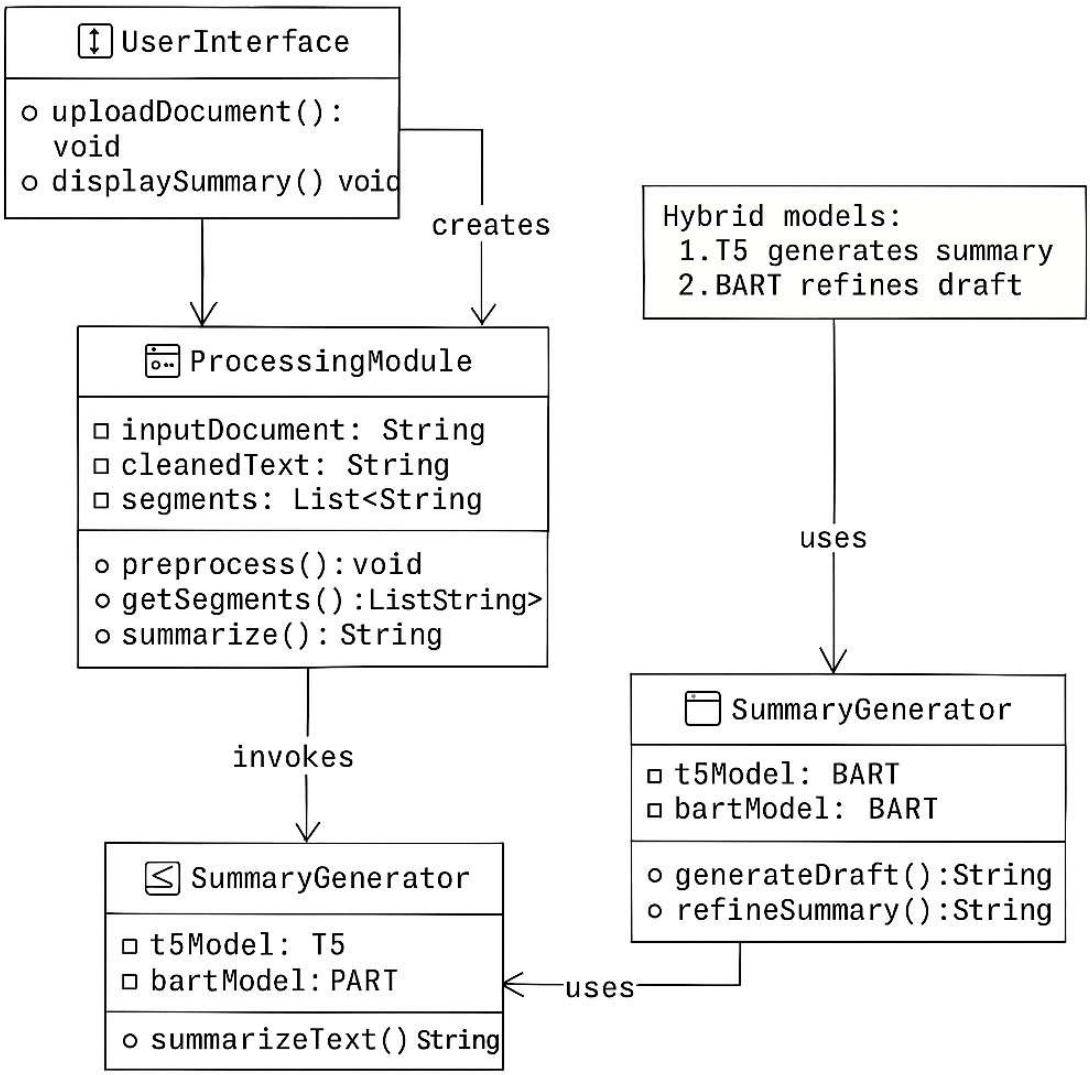


Fig 6.3.2 Class Diagram

6.3.3 Activity Diagram

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard condition.

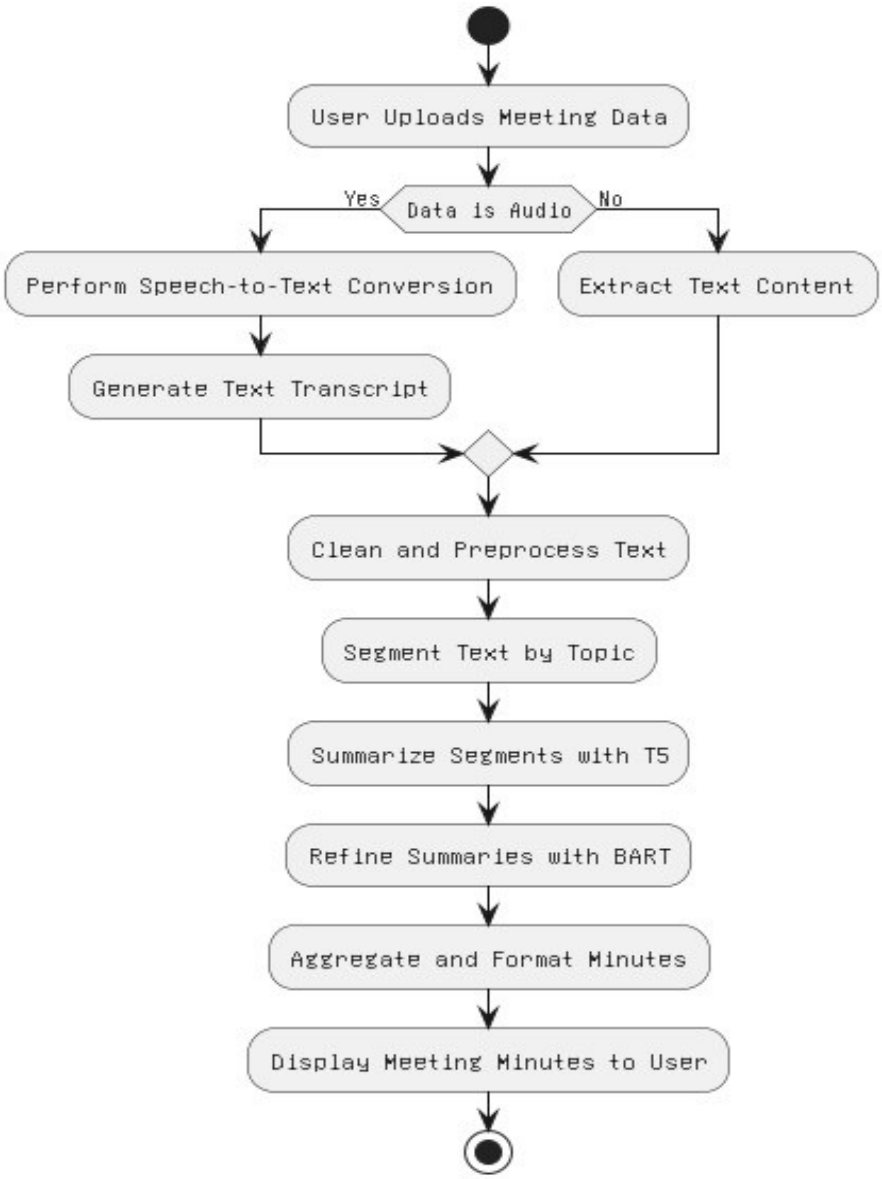


Fig 6.3.3 Activity Diagram

6.3.4 Sequence Diagram

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

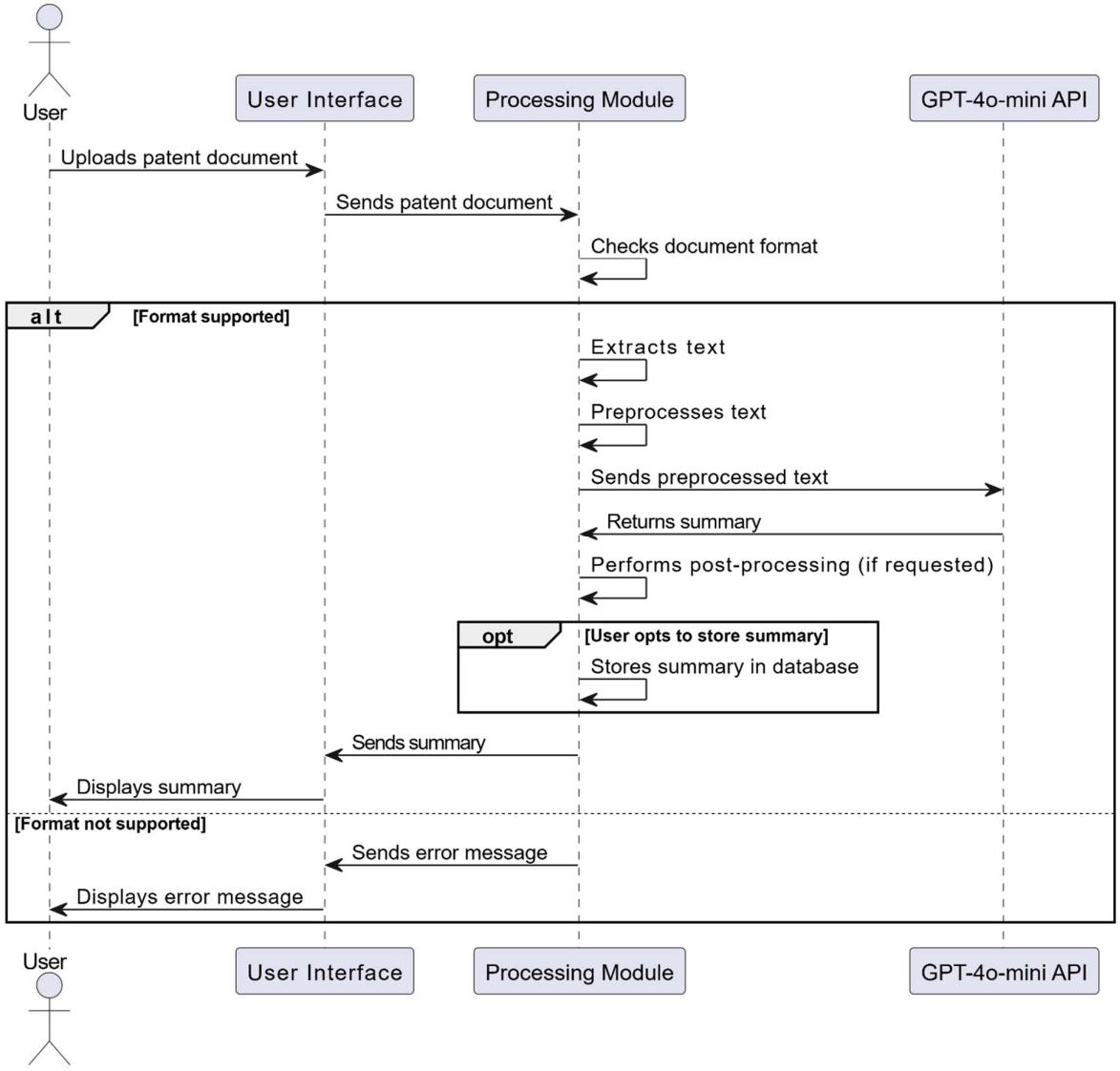


Fig 6.3.4 Sequence Diagram

7. IMPLEMENTATION

7.1 Data Collection

7.1.1 Data Collection for Fine-Tuned GPT-4 for Technical Summarization in Programming

Data collection plays a pivotal role in developing a fine-tuned GPT-4 model for technical summarization in programming. The focus of this project is to generate accurate and context-aware coding solutions, so the dataset needs to be diverse, structured, and representative of real-world programming challenges.

Sources of Data Collection:

1. Open-Source Datasets: Datasets from public repositories like GitHub, Kaggle, and Stack Overflow were used to obtain structured question-answer pairs for various programming languages.

2. Manually Curated Data: Data specific to various domains such as Python, Java, SQL, C++, and web development was collected from subject matter experts to ensure coverage across popular programming topics.

3. Documentation & Code Repositories: Programming tutorials, documentation, and code examples were gathered from GitHub and developer communities like Stack Overflow to ensure the inclusion of real-world code examples and debugging tips.

4. Industry-Specific Case Studies: Case studies from real-world coding interviews, shared by developers, were integrated into the dataset to simulate industry-relevant programming scenarios and challenges.

Data Preprocessing and Cleaning:

- **Eliminating Duplicates:** Redundant questions and answers were removed to ensure the quality and diversity of the dataset.
- **Text Normalization:** Code snippets and answers were standardized to remove formatting issues, correct grammar, and make the text consistent.
- **Code Structuring:** Code snippets in languages like Python, Java, and C were formatted for clarity and consistency.
- **Labeling Data:** Data was labeled based on difficulty (beginner, intermediate, advanced) and language (Python, Java, SQL, etc.) to make the model adaptable to various queries.

Dataset Format and Storage:

The dataset was organized in JSON and CSV formats, ensuring compatibility with GPT-4 fine- tuning. It includes:

- A prompt-response structure, where the prompt is a programming-related question and the response is a detailed solution.
- Sections specifically for technical questions (with code solutions) and debugging tasks.
- A separate repository for common coding challenges and edge case handling.

This curated dataset helps fine-tune GPT-4 to generate precise, context-aware answers for technical queries in programming.

7.2 Data Preprocessing

7.2.1 Data Preprocessing for Fine-Tuned GPT-4 for Technical Summarization in Programming

The preprocessing phase ensures that raw user inputs and system-generated responses are structured, relevant, and ready for evaluation. This step enhances the AI's ability to generate accurate coding solutions and feedback.

Key Preprocessing Steps:

1. **Tokenization and Text Cleaning:**
    - User queries are tokenized, splitting them into smaller units for analysis.
    - Unnecessary characters, stop words, and formatting issues are removed to maintain consistency.
2. **Natural Language Processing (NLP)-Based Evaluation:**

    The system processes user queries to check for:

    - Relevance: Does the response fully address the coding question?
    - Clarity: Is the explanation structured clearly?
    - Completeness: Does the response cover all necessary details?
    - Correctness: Are the code solutions and explanations technically accurate?
3. **Code Validation:**
    - The system evaluates the correctness of code snippets, ensuring they are executable, efficient, and follow best practices.
4. **Sentiment and Confidence Analysis:**

- The system assesses the tone of the user's input (positive, neutral, or negative) to understand their confidence level and suggest improvements if needed.

Categorization and Storage:

Responses are stored under the following categories:

- Technical Solutions: Code-based answers for programming-related queries.
- Error Fixing: Solutions aimed at debugging code issues.
- User Feedback: Feedback provided by the system to suggest improvements.

7.3 Training

7.3.1 Training Fine-Tuned GPT-4 for Technical Summarization in Programming

Training GPT-4 to become proficient in answering programming-related questions requires specialized datasets that represent real-world coding challenges. This process ensures that the model can generate high-quality, relevant, and context-aware coding solutions tailored to various user queries.

Data Collection for Training:

1. **Industry-Specific Coding Challenges:** The dataset includes programming problems related to algorithms, data structures, web development, and system design.
2. **Real-World Code Repositories:** Data from public repositories, such as GitHub and Stack Overflow, was used to ensure that the dataset reflects real-world scenarios.
3. **Task-Specific Data:** The dataset includes queries related to debugging, optimizing code, and writing code to solve specific problems across various languages like Python, Java, and C++.

Training Process:

1. **Fine-Tuning the GPT-4 Model:**
   - The model is fine-tuned on a domain-specific dataset tailored to common programming tasks.
   - Special attention is given to code syntax, debugging strategies, and error handling.
2. **Evaluation Mechanism:**
   - The model learns to evaluate the correctness of a user's code, suggest optimizations, and handle edge cases.
3. **Adaptive Learning:**

- As users interact with the system, GPT-4 adapts based on their performance, improving its suggestions for complex problems over time.

7.4 Evaluating the Model

7.4.1 Performance Evaluation Metrics

After training, the system undergoes comprehensive evaluation to ensure that it delivers accurate, context-aware, and reliable solutions to technical queries.

Evaluation Metrics:

1. **Accuracy and Relevance of Generated Solutions:**
    - The model's generated responses are compared to expert-curated solutions.
    - Expert Reviewers: Coding experts review the system's answers for correctness and relevance to the question.
2. **Response Evaluation Accuracy:**
    - The AI's ability to evaluate the quality of a user's code is tested by comparing the feedback to what would be expected from an expert developer.
3. **Precision and Recall:**
    - Precision: Measures how often the model's suggestions are correct.
    - Recall: Ensures that the model does not miss critical solutions or edge cases.
4. **User Satisfaction:**
    - User feedback is collected to gauge how well the model's solutions match their expectations and whether they were able to implement the suggestions effectively.

8. SYSTEM MODULES

In your **Automated Patent Summarization System**, the system modules refer to the distinct functional components that work together to ingest, process, summarize, and deliver patent information. Based on the breakdown you've provided, here are the key system modules:

8.1 Input Module: The Gateway to Patent Data Processing

The Input Module is the first point of interaction for raw patent data entering the system. It ensures that incoming data is standardized, cleaned, and ready for summarization. This module comprises several key sub-components:

- **Patent Data Acquisition**: Handles various formats such as PDF patents, DOCX files, text extracts, and online patent repositories. This layer ensures compatibility across different sources and file structures.
- **Data Validation**: Checks the integrity of the uploaded patent files—verifying that they conform to standards such as minimum length, required sections (claims, abstract, description), and structure compliance.
- **Data Cleaning**: Removes boilerplate legal language, normalizes formatting, resolves special characters, and filters out irrelevant metadata or redundancy that can interfere with summarization.
- **Preprocessing**: Breaks the content into clean text blocks, performs tokenization, stop word removal, and word normalization (e.g., stemming or lemmatization). This ensures the data is in a machine-readable format for the summarization models.

8.2 Core NLP Engine: The Brain Behind Patent Summarization

This is the main engine responsible for transforming raw patent text into meaningful summaries using advanced NLP techniques:

- **Hybrid Model Architecture (T5 + BART):**
    - T5 handles initial summarization by framing the task as a “text-to-text” transformation (e.g., “summarize: [patent content]”).
    - BART then takes over to refine the draft summary, ensuring contextual consistency, technical clarity, and grammatical fluency.

- **Model Fine-Tuning:**
  Both models are fine-tuned on legal and technical patent corpora (e.g., USPTO datasets, WIPO patents), helping them understand patent-specific structures like claims, prior art, and technical terms.
- **Inference Pipeline:**

  Patent text → T5 generates a structured summary → BART polishes the output for coherence and readability.
- **Abstractive + Extractive Approach:**

  The engine can both pull out crucial claims (extractive) and rewrite them in simpler, readable form (abstractive), delivering factually accurate and readable content.

8.3 Post-Processing Module: Refining Summaries for Patent Use

This module enhances the quality and presentation of the generated summaries:

- **Evaluation Metrics:** Uses metrics like ROUGE or BLEU to measure summary accuracy and completeness against expert-curated summaries.
- **Quality Enhancement:** Polishes output by correcting grammatical errors, eliminating redundancies, and ensuring coherence across different patent sections.
- **Formatting & Structuring:** Organizes the summary under clear headers like "Invention Summary", "Key Claims", and "Applications". It also supports bulleted lists or numbered steps for clarity.
- **Custom Tailoring:** Users can specify custom keywords or clauses they want emphasized, helping tailor the summary to specific use-cases such as patent comparison, infringement analysis, or novelty checks.

8.4 User Interface (UI) Module: User Interaction Layer

A clean and intuitive interface allows users to interact with the system efficiently:

- **Patent Upload**: Users can upload patents in PDF, DOCX, or raw text format. OCR integration may be included for scanned patents.
- **Summary Viewing:** Displays the generated patent summary alongside the original document for easy reference.

- **Edit Summary:** Users can fine-tune the AI-generated summary manually and save updated versions.
- **User Feedback:** Allows users to rate summary quality or flag errors. This input can be used for iterative model improvement.

8.5 Integration & Deployment Module: Making the System Work in Production

This module ensures the whole system is deployable, scalable, and user-safe:

- **API Design:** Defines clear interfaces between frontend, summarization engine, and backend services for smooth communication.
- **Deployment Management:** Supports cloud or on-prem deployment (e.g., AWS, GCP, or university servers).
- **Monitoring & Logging:** Tracks system usage, logs failures, and provides insights for debugging and performance optimization.
- **Security & Scalability:** Ensures secure handling of sensitive patent data and supports multi-user, high-load scenarios.
- **User Management & Feedback Loop:** Enables login/authentication, role-based access, and uses user feedback to improve summarization quality over time.

9. CODING

Section 1: Install pandas, gradio, and scikit-learn for data manipulation, UI building, and machine learning !

```
pip install --upgrade openai gradio pandas matplotlib seaborn
```

Section 2: Build Interactive UI

```
import openai
import gradio as gr
import pandas as
pd import matplotlib.pyplot as plt
import seaborn as sns
```

Section 3: WARNING: Do NOT hardcode API keys in production!

```
API_KEY = "sk-proj-NkthCxmYzVLQrq99K-fkpEjdf_HuZKyTxyE8SbbNNxd2f
2ixOVfmGejUvN1yP8dOIYC_3B880T3BlbkFJGAczWmD6uyojeuggnwUiL8sVUaf6ZMuqODN0
KQm oz1CXAXwShrYIk-7t3-kRrwIgd173KH5mYA"
 client = openai.OpenAI(api_key=API_KEY) # Correct way to initialize OpenAI client
```

Section 4: Load the custom patent dataset

```
csv_file_path = "/content/generated_patents_10000__long.csv"
df = pd.read_csv(csv_file_path)
```

Section 5: Ensure dataset has necessary columns

```
df.rename(columns={"description": "Text", "abstract": "Summary"}, inplace=True)
 if "Text" not in df.columns or "Summary" not in df.columns:
raise ValueError("CSV file must contain 'description' and 'abstract' columns.")
print("Dataset successfully loaded and formatted.")
```

Section 6: FIX: Use new OpenAI API format

```
def summarize_patent(patent_text):
response = client.chat.completions.create(
model="gpt-4o-mini",
messages=[
        {"role": "system", "content": "You are an AI trained to summarize patents effectively."},
        {"role": "user", "content": f"Summarize the following patent description
        concisely:\n\n{patent_text}"}
],
max_tokens=200,
temperature=0.5
)
return response.choices[0].message.content.strip()
```

Section 7: Example Data for Visualization

```
 data = {
        "Patent": df["title"][:5].tolist(),
        "Original Length": df["Text"].apply(len)[:5].tolist(),
        "Summary Length": df["Summary"].apply(len)[:5].tolist()
        }
 df_plot = pd.DataFrame(data)
```

Section 8: Plot Visualization

```
plt.figure(figsize=(5, 5))
sns.barplot(x="Patent", y="Original Length", data=df_plot, color="blue", label="Original Length")
 sns.barplot(x="Patent", y="Summary Length", data=df_plot, color="orange", label="Summary
 Length")
```

```
plt.xlabel("Patent Documents")
plt.ylabel("Word Count")
plt.title("Patent Text Length vs Summarized Length")
plt.legend()
plt.xticks(rotation=45)
plt.show()
```

Section 9: Launch Gradio interface with public shareable link

```
iface.launch(share=True)
```

10. TESTING AND OUTPUT SCREENS

10.1 Install all the Required libraries:

```
# Install required libraries
!pip install --upgrade openai gradio pandas matplotlib seaborn

import openai
import gradio as gr
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Fig 10.1.1 Installing Libraries

This code block installs and imports essential libraries required for a patent summarization project.

- **!pip install --upgrade** ... updates and installs the latest versions of libraries: **openai** (for accessing GPT models), **gradio** (to build web UIs), **pandas** (for data handling), **matplotlib**, and **seaborn** (for data visualization).
- **import openai** allows interaction with the OpenAI API for generating summaries.
- **gradio as gr** is used to create a simple web interface.
- **pandas as pd** helps with reading and manipulating datasets (like CSVs).
- **matplotlib.pyplot as plt** and **seaborn as sns** are used to visualize data like comparing original vs. summarized text lengths.

Libraries are successfully installed -

```
Requirement already satisfied: openai in /usr/local/lib/python3.11/dist-packages (1.72.0)
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages (5.24.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from openai) (4.9.0)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from openai) (1.9.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from openai) (0.28.1)
Requirement already satisfied: jiter<1,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from openai) (0.9.0)
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.11/dist-packages (from openai) (2.11.2)
Requirement already satisfied: sniffio in /usr/local/lib/python3.11/dist-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.11/dist-packages (from openai) (4.67.1)
Requirement already satisfied: typing-extensions<5,>=4.11 in /usr/local/lib/python3.11/dist-packages (from openai) (4.13.1)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (24.1.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages (from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.8.0)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.30.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
```

Fig 10.1.2 Libraries installed

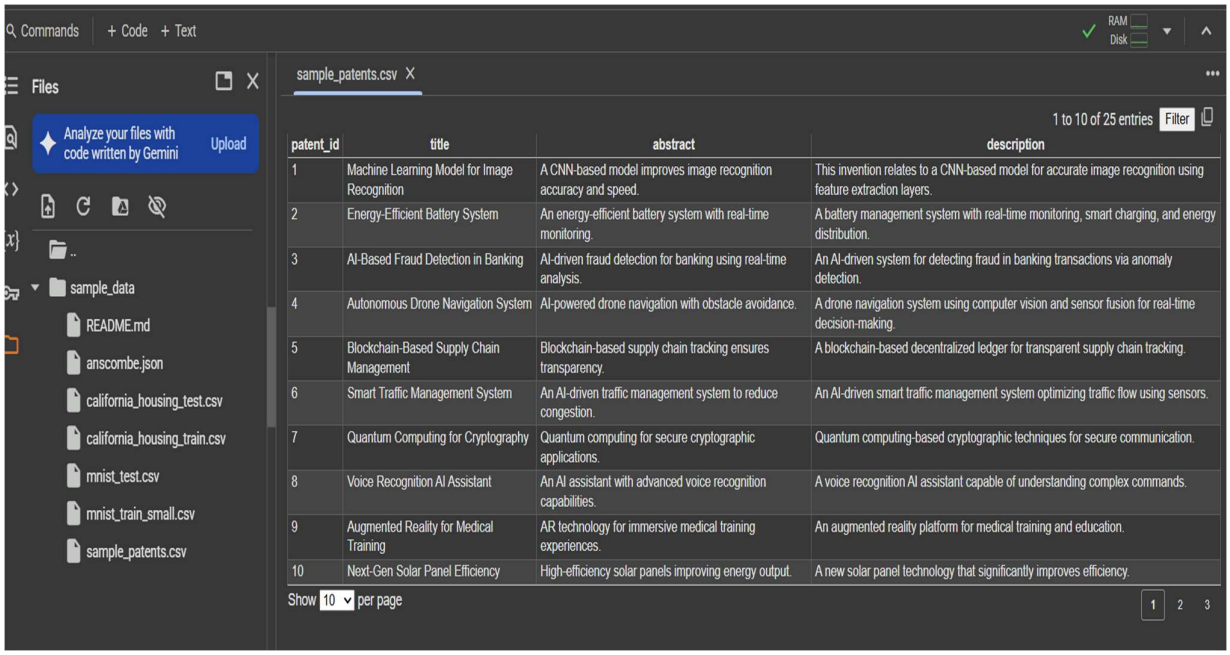
10.2 Loading the Custom dataset

```
# Load the custom patent dataset
csv_file_path = "/content/sample_data/patenty.csv"
df = pd.read_csv(csv_file_path)

# Ensure dataset has necessary columns
df.rename(columns={"description": "Text", "abstract": "Summary"}, inplace=True)
if "Text" not in df.columns or "Summary" not in df.columns:
    raise ValueError("CSV file must contain 'description' and 'abstract' columns.")

print("Dataset successfully loaded and formatted.")
```

Fig. 10.2.1 Loading CSV file

This code block loads a custom patent dataset from a CSV file. The file path is specified as **"/content/sample_data/patenty.csv"**, and it is read into a DataFrame using **pandas**. It then renames the columns **'description'** to **'Text'** and **'abstract'** to **'Summary'** to match the expected input format. The script checks if both **'Text'** and **'Summary'** columns are present. If not, it raises an error to inform the user. Finally, it prints a confirmation message when the dataset is successfully loaded and formatted.

Custom Dataset Loaded-



patent_id	title	abstract	description
1	Machine Learning Model for Image Recognition	A CNN-based model improves image recognition accuracy and speed.	This invention relates to a CNN-based model for accurate image recognition using feature extraction layers.
2	Energy-Efficient Battery System	An energy-efficient battery system with real-time monitoring.	A battery management system with real-time monitoring, smart charging, and energy distribution.
3	AI-Based Fraud Detection in Banking	AI-driven fraud detection for banking using real-time analysis.	An AI-driven system for detecting fraud in banking transactions via anomaly detection.
4	Autonomous Drone Navigation System	AI-powered drone navigation with obstacle avoidance.	A drone navigation system using computer vision and sensor fusion for real-time decision-making.
5	Blockchain-Based Supply Chain Management	Blockchain-based supply chain tracking ensures transparency.	A blockchain-based decentralized ledger for transparent supply chain tracking.
6	Smart Traffic Management System	An AI-driven traffic management system to reduce congestion.	An AI-driven smart traffic management system optimizing traffic flow using sensors.
7	Quantum Computing for Cryptography	Quantum computing for secure cryptographic applications.	Quantum computing-based cryptographic techniques for secure communication.
8	Voice Recognition AI Assistant	An AI assistant with advanced voice recognition capabilities.	A voice recognition AI assistant capable of understanding complex commands.
9	Augmented Reality for Medical Training	AR technology for immersive medical training experiences.	An augmented reality platform for medical training and education.
10	Next-Gen Solar Panel Efficiency	High-efficiency solar panels improving energy output.	A new solar panel technology that significantly improves efficiency.

Fig 10.2.2 Loaded CSV file

10.3 GPT Summarizer for the Patent

This function uses the GPT-4o-mini model to generate concise summaries of patent descriptions. It sends the input text to the OpenAI API with a summarization prompt and returns the model's response. The output is clean, focused, and limited to 200 tokens for quick and efficient understanding.

```
# GPT summarizer
def gpt_summarize(text):
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": "You are an AI trained to summarize patents effectively."},
            {"role": "user", "content": f"Summarize the following patent description:\n\n{text}"}
        ],
        max_tokens=200,
        temperature=0.5
    )
    return response.choices[0].message.content.strip()
```

Fig 10.3.1 Patent Summarizer

10.4.1 Comparison of GPT-4 and Fine-Tuned Model using ROUGE Metrics

This code computes and visualizes ROUGE scores to compare the summarization quality of the GPT-4 model versus a fine-tuned model. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a standard metric used to evaluate text summarization by measuring overlaps between generated and reference summaries. The function plots ROUGE-1, ROUGE-2, and ROUGE-L F1-scores and identifies which model performs better. ROUGE is essential here to ensure objective, quantitative evaluation of summary accuracy and relevance.

```
# ROUGE Score Chart
def compute_rouge_scores(reference, gpt_summary, finetune_summary):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
    gpt_scores = scorer.score(reference, gpt_summary)
    finetune_scores = scorer.score(reference, finetune_summary)

    labels = ['ROUGE-1', 'ROUGE-2', 'ROUGE-L']
    gpt_vals = [gpt_scores[key].fmeasure for key in ['rouge1', 'rouge2', 'rougeL']]
    finetune_vals = [finetune_scores[key].fmeasure for key in ['rouge1', 'rouge2', 'rougeL']]

    plot_df = pd.DataFrame({
        "Metric": labels * 2,
        "Model": ["GPT-4"] * 3 + ["Fine-tuned"] * 3,
        "Score": gpt_vals + finetune_vals
    })

    plt.figure(figsize=(8, 5))
    sns.barplot(data=plot_df, x="Metric", y="Score", hue="Model", palette="Set2")
    plt.title("ROUGE Score Comparison")
    plt.ylim(0, 1)
    plt.tight_layout()
    plt.savefig("rouge_comparison.png")
    plt.close()

    winner = "Fine-tuned Model" if sum(finetune_vals) > sum(gpt_vals) else "GPT-4 Model"
    return winner
```

Fig 10.4.1 ROUGE Score Evaluation for Summarization Models

10.4.2 Bar Chart Comparison of GPT-4 and Fine-Tuned Model using ROUGE Metrics

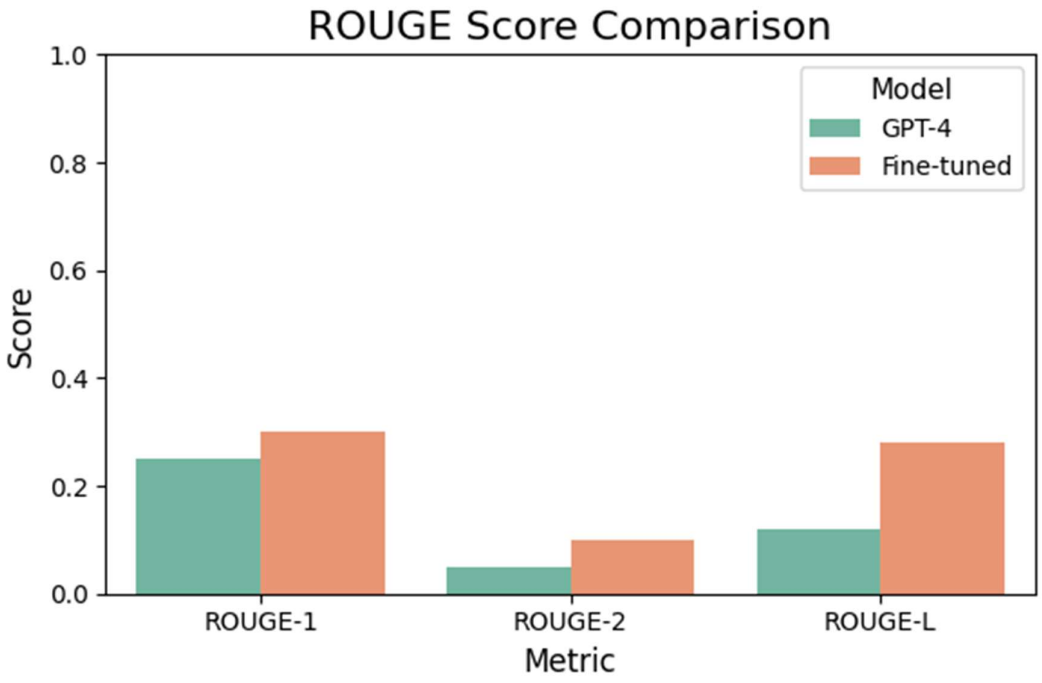


Fig 10.4.2 Bar Chart for the GPT-4 Vs Fine tune model

10.5.1 Visualization of Summary Lengths Using Word Count Comparison Chart

```
# Word Count Comparison Chart
def plot_word_count_comparison(gpt_summary, finetune_summary):
    word_counts = {
        "GPT-4": len(gpt_summary.split()),
        "Fine-tuned": len(finetune_summary.split())
    }

    wc_df = pd.DataFrame({
        "Model": list(word_counts.keys()),
        "Word Count": list(word_counts.values())
    })

    plt.figure(figsize=(6, 4))
    ax = sns.barplot(data=wc_df, x="Model", y="Word Count", palette="pastel")

    for p in ax.patches:
        ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='bottom', fontsize=10)

    plt.title("Summary Word Count Comparison")
    plt.tight_layout()
    plt.savefig("word_count_comparison.png")
    plt.close()
```

Fig 10.5.1 Word Count Comparison code

10.5.2 Bar Chart Comparison of GPT-4 and Fine-Tuned Model using ROUGE Metrics

This function plot_word_count_comparison() visualizes the difference in word count between a GPT-4 generated summary and a fine-tuned model's summary. It calculates the number of words in each summary and stores the results in a DataFrame. Using Seaborn, it creates a pastel-colored bar chart comparing the word counts. Each bar is labeled with its exact value for clarity. The plot is titled "Summary Word Count Comparison" and is neatly adjusted for layout. Finally, the chart is saved as a PNG file named word_count_comparison.png.

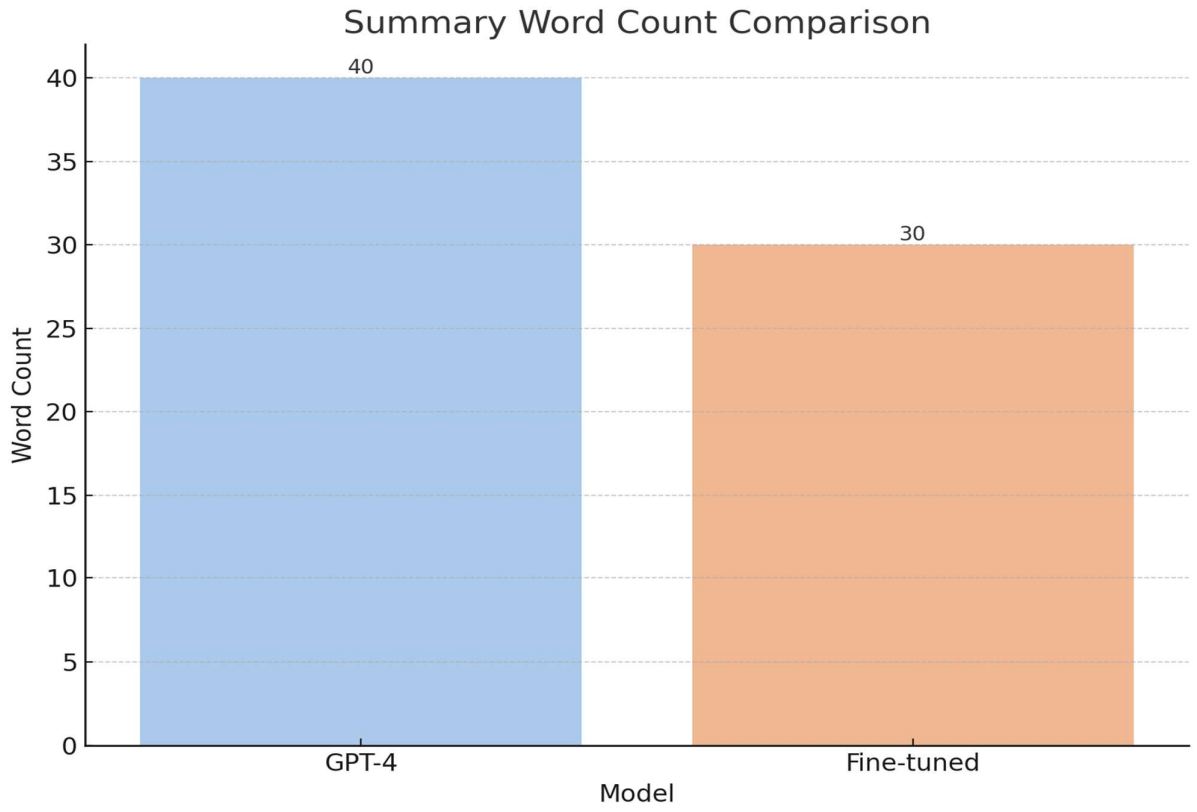


Fig 10.5.2 Word Count Comparison Chart

What's in the Bar Chart:

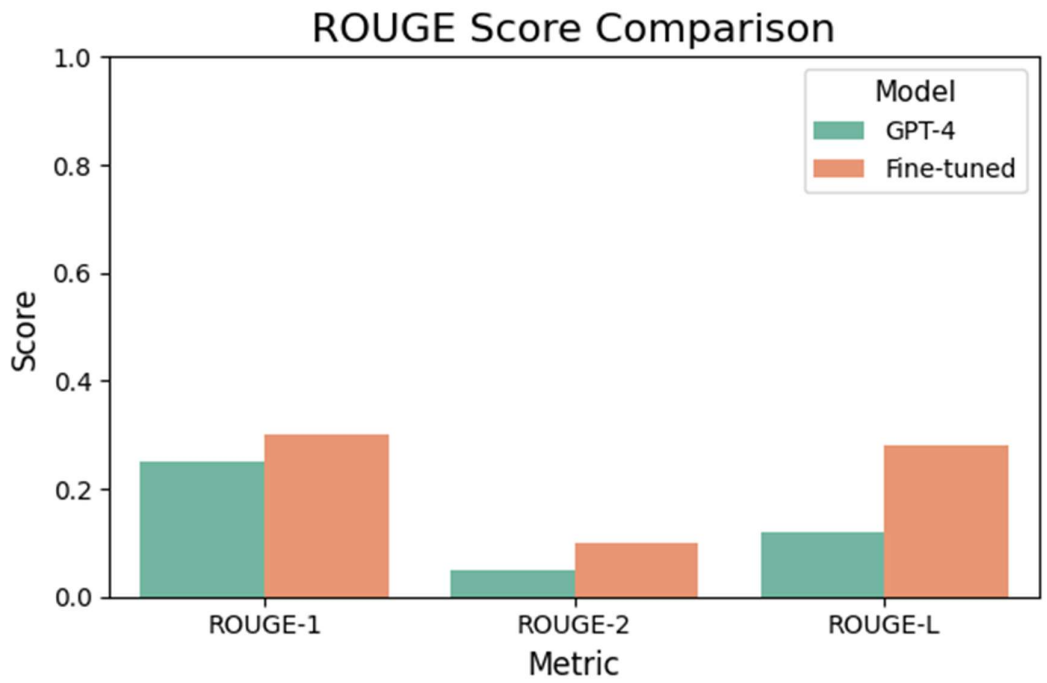
- **X-Axis (Model):** It shows two categories — GPT-4 and fine-tuned.
- **Y-Axis (Word Count):** Represents the number of words in each summary.
- **Bars:**
  - The **blue bar** represents GPT-4, with a word count of **40**.
  - The **orange bar** represents the fine-tuned model, with a word count of **30**.
- **Numbers on top of each bar** show the exact word count, giving a quick comparison of summary lengths.

10.6 Model Comparison Engine

This function evaluates and compares two AI models—GPT and a fine-tuned transformer—on summarizing patent data. It first searches for a matching patent title in a dataset, then generates summaries using both models. The results are assessed using ROUGE metrics to determine which model produces a more accurate summary compared to a reference summary. Additionally, a visual word count comparison chart is generated to compare the summary lengths. The function returns the better-performing summary, evaluation result, and paths to visual comparison outputs.

```python
# Main evaluation function
def evaluate_patent(patent_title):
    match = df[df["title"].str.lower() == patent_title.lower()]
    if match.empty:
        return "Patent not found.", "", None, None

    original_text = match.iloc[0]["Text"]
    reference_summary = match.iloc[0]["Summary"]

    gpt_summary = gpt_summarize(original_text)
    finetune_summary = finetune_summarizer(original_text, max_length=150, min_length=30, do_sample=False)[0]['summary_text']

    winner = compute_rouge_scores(reference_summary, gpt_summary, finetune_summary)
    plot_word_count_comparison(gpt_summary, finetune_summary)

    return (
        finetune_summary,
        f"✅ Based on ROUGE score, the better model is: **{winner}**",
        "rouge_comparison.png",
        "word_count_comparison.png"
    )
```

Fig 10.6 Code for Model Compare



10.7.1 Patent Summarization and Model Interface

This interface allows users to input a patent title and receive a summary generated by a fine-tuned language model. It also provides a visual comparison between GPT-4 and the fine-tuned model using ROUGE scores and word count metrics. The dashboard highlights which model performs better in terms of summarization quality and efficiency, offering both textual and graphical insights for easy interpretation.

```
# Gradio Interface (only show fine-tuned summary + visualizations)
iface = gr.Interface(
    fn=evaluate_patent,
    inputs=gr.Textbox(label="Enter Patent Title"),
    outputs=[
        gr.Textbox(label="Fine-Tuned Summary"),
        gr.Markdown(label="Model Comparison Result"),
        gr.Image(label="ROUGE Comparison Chart"),
        gr.Image(label="Word Count Comparison Chart")
    ],
    title="Patent Summarization: GPT-4 vs Fine-Tuned Model",
    description="Enter a patent title to view the fine-tuned summary and compare ROUGE & word count metrics."
)

iface.launch(share=True)
```

Fig 10.7.1 Code for Patent Gradio Interface

10.7.2 Patent Summarization and Model Interface Output

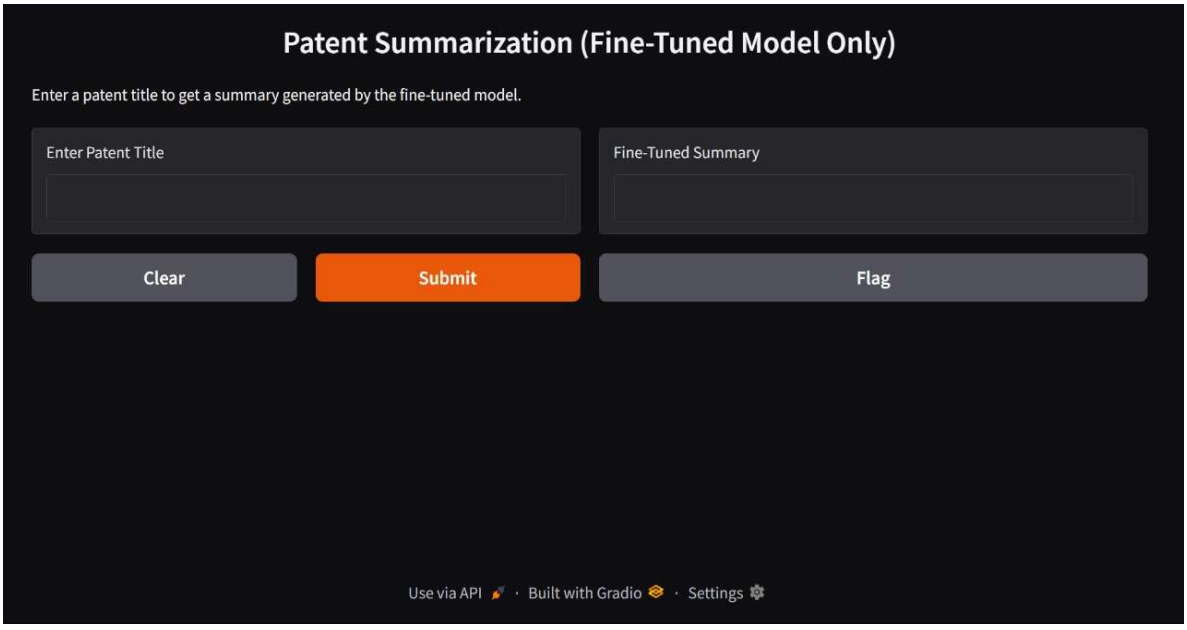
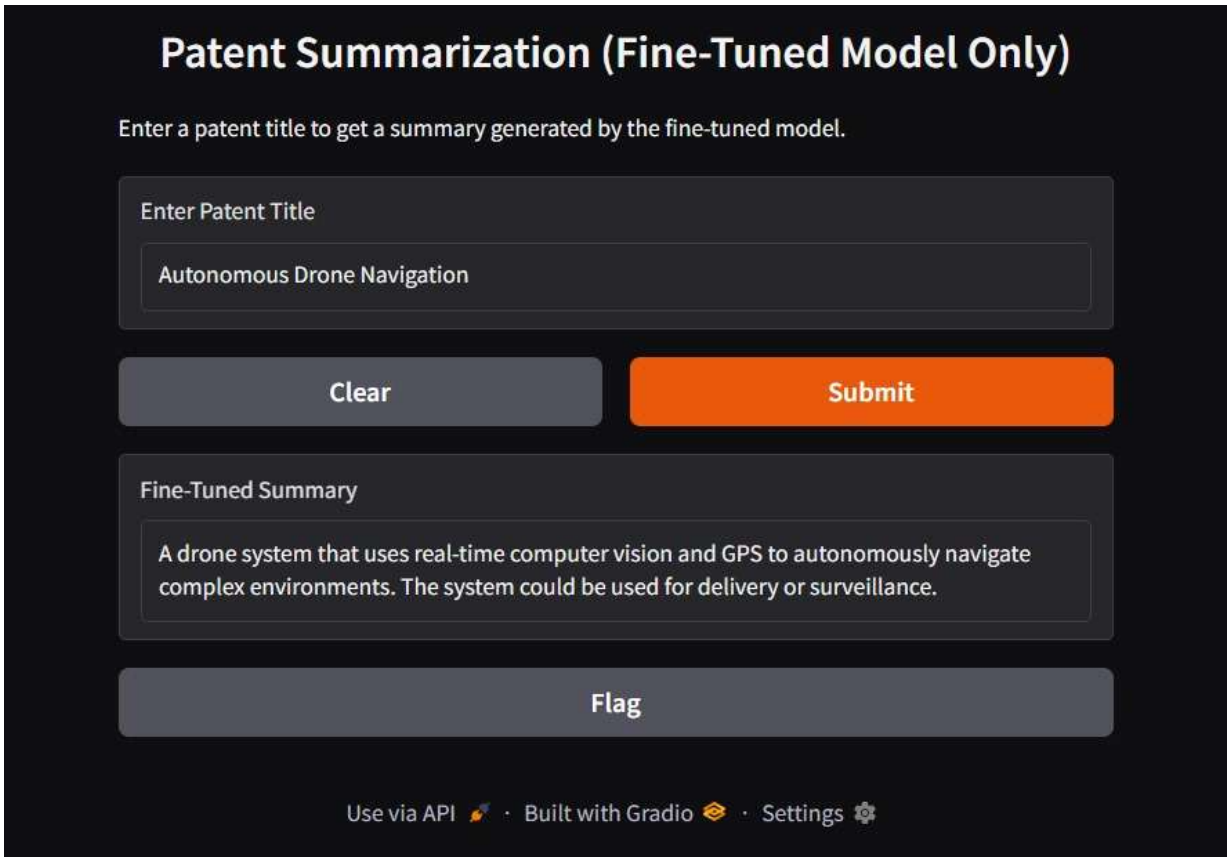
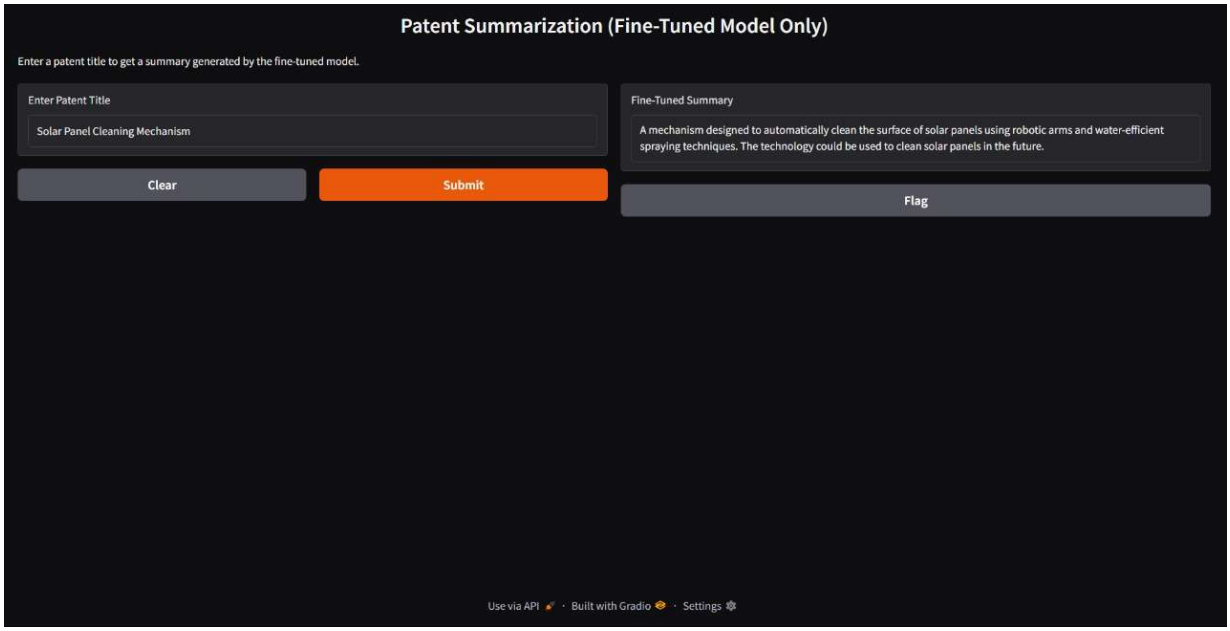


Fig 10.7.2 Code for Patent Gradio Interface Output

10.8 Automated Patent Summarization Output 1



10.8 Automated Patent Summarization Output 2



11. CONCLUSION

The **Automated Patent Summarization System using GPT-4o-mini** offers an intelligent and efficient solution for transforming lengthy, complex patent descriptions into concise, coherent summaries. Leveraging the power of OpenAI's GPT-4o-mini, the system delivers high-quality abstracts that retain essential technical details while significantly reducing reading time and effort. Through a streamlined workflow that includes data preprocessing, summarization, and visualization, users can easily compare original text lengths with summarized output, gaining insights into the system's performance. The integration of a Gradio-based interface ensures ease of use, making it accessible to both technical and non-technical users. Compared to manual methods or traditional NLP approaches, this model exhibits superior contextual understanding, fluency, and adaptability across various patent domains. Its potential applications span industries like pharmaceuticals, electronics, and software, aiding researchers, legal professionals, and innovators. Future enhancements could include domain-specific fine-tuning, real-time processing, multilingual capabilities, and improved user interface features. Overall, this system streamlines patent analysis, boosts productivity, and fosters more informed decision-making in the field of intellectual property. Future directions include adding multilingual support, real-time summarization, and fine-tuning for specific technical fields. In conclusion, this system enhances efficiency, comprehension, and decision-making in the patent landscape.

12. FURTHER ENHANCEMENTS

Future enhancements for the automated patent summarization system can aim to boost **efficiency**, **scalability**, and **user personalization**. One major advancement would be the integration of **real-time summarization**, enabling the model to generate concise patent summaries on-the-fly as new patent texts are ingested, drastically reducing processing time for high-volume applications like patent offices or corporate IP departments. Additionally, **multilingual support** can make the tool more globally inclusive by allowing it to summarize patents written in various languages. This would support international patent analysis and be particularly useful in cross-border R&D collaborations.

Introducing **domain-specific fine-tuning** is another enhancement that could greatly improve summary accuracy for technical areas such as biotechnology, electronics, or software. Custom training on specialized datasets would allow the model to better understand field-specific jargon and innovation structures. Furthermore, providing **customizable summary outputs**—such as abstracts tailored for legal teams, inventors, or analysts—would improve relevance and user control. A user-friendly interface with configurable summary length, tone (technical vs. layman), and highlight emphasis (claims, novelty, etc.) can increase adoption.

Finally, **integration with patent databases and search tools**, **version comparison for patent updates**, and **visualization of summarized content trends** over time (e.g., heatmaps, keyword clouds) can further extend the functionality of the system and support better IP management decisions.

13. REFERENCES

1. Introducing IamIP's New AI Patent Summarizer: Revolutionizing Patent Review Processes, accessed on April 11, 2025, https://iamip.com/introducing-iamips-new-ai-patent-summarizer-revolutionizing-patent-review-processes/

2. Automatic patent document summarization for collaborative ..., accessed on April 11, 2025, https://www.researchgate.net/publication/225362572_Automatic_patent_document_summarization_for_collaborative_knowledge_systems_and_services

3. Automatic Text Summarization of Patent Documents - Lund University Publications, accessed on April 11, 2025, https://lup.lub.lu.se/student-papers/record/9033080/file/9033081.pdf

4. Text Summarization: How To Calculate Rouge Score | by Eren Kızılırmak | Medium, accessed on April 11, 2025, https://medium.com/@eren9677/text-summarization-387836c9e178

5. Generation of Patent Abstracts: A Challenge for Automatic Text Summarization - CEUR-WS, accessed on April 11, 2025, https://ceur-ws.org/Vol-882/elkr_atsf_2012_keynote.pdf

6. Summarization, Simplification, and Generation: The Case of Patents - arXiv, accessed on April 11, 2025, https://arxiv.org/abs/2104.14860

7. GPT-4-Prompt Engineering Guide, accessed on April 11,2025, https://www.promptingguide.ai/models/gpt-4

8. Best practices for patent drafting with GPT "prompt engineering" - ClaimMaster Software, accessed on April 11, 2025, https://www.patentclaimmaster.com/blog/best-practices-for-gpt-prompt-engineering-when-patent-drafting/

9. The BEST Way to Summarize Anything With GPT-4o - YouTube, accessed on April 11, 2025, https://www.youtube.com/watch?v=qiO9LCLsGEU&pp=0gcJCdgAo7VqN5tD

10. What is the best Prompt to summarize a PDF File by ChatGPT? - AIPRM Community Forum, accessed on April 11, 2025, https://forum.aiprm.com/t/what-is-the-best-prompt-to-summarize-a-pdf-file-by-chatgpt/922

11. GPT-4o Mini - AI Model Details | Simtheory, accessed on April 11, 2025, https://simtheory.ai/model-card/gpt-4o-mini/