PolyGPT

A major project report submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

Submitted by

Prateek Kumar(221030366)

Under the guidance & supervision of

Mr. Ravi Sharma

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

December 2025

SUPERVISOR'S CERTIFICATE

This is to certify that the major project report entitled "**PolyGPT**", submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science &amp; Engineering, in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is a bonafide project work carried out under my supervision during the period from July 2025 to December 2025.

I have personally supervised the research work and confirm that it meets the standards required for submission. The project work has been conducted in accordance with ethical guidelines, and the matter embodied in the report has not been submitted elsewhere for the award of any other degree or diploma.

Supervisor Name & Sign:

Date:

Designation:

Place:

Department:

Candidate's Declaration

We hereby declare that the work presented in this major project report entitled **'PolyGPT**, submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering**, in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is an authentic record of our own work carried out during the period from July 2025 to December 2025 under the supervision of **Mr. Ravi Sharma.**

We further declare that the matter embodied in this report has not been submitted for the award of any other degree or diploma at any other university or institution.

Name: Prateek Kumar
Roll No.: 221030366
Date:

This is to certify that the above statement made by the candidates is true to the best of my knowledge.

	Supervisor Name & Sign:
Date:	Designation:
Place:	Department:

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who supported me throughout the development of my Major Project titled “PolyGPT” First and foremost, I would like to thank my project supervisor, Mr. Ravi Sharma, for her invaluable guidance, constant encouragement, and insightful feedback during every phase of this project. Her support and mentorship played a crucial role in shaping this work.

I am also thankful to the Department of Computer Science and Engineering and all the faculty members for providing the necessary facilities, academic environment, and resources that enabled me to complete this project successfully.

I extend my heartfelt appreciation to my friends, classmates, and family members for their continuous motivation and support, which helped me stay focused and committed throughout the journey.

Finally, I express my deep gratitude to all individuals who directly or indirectly contributed to the successful completion of this project.

Prateek Kumar

221030366, B.Tech (CSE)

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

S. No.	Abbreviation	Full Form
1	AI	Artificial Intelligence
2	LLM	Large Language Model
3	RAG	Retrieval-Augmented Generation
4	NLP	Natural Language Processing
5	OCR	Optical Character Recognition
6	MVT	Model–View–Template Architecture
7	API	Application Programming Interface
8	DTO	Data Transfer Object
9	KB	Knowledge Base
10	AWS	Amazon Web Services
11	GCP	Google Cloud Platform
12	S3	Amazon Simple Storage Service
13	OCR-NLP	OCR Enabled Natural Language Processing
14	BRE	Business Rules Engine
15	MCQ	Multiple Choice Question
16	CMR	Credit Model Rating
17	CIBIL	Credit Information Bureau (India) Limited
18	RCM	Regional Credit Manager
19	TAT	Turnaround Time
20	CI/CD	Continuous Integration / Continuous Deployment
21	LCA	LangChain Agent
22	SDK	Software Development Kit
23	UAT	User Acceptance Testing
24	Pydantic	Python Data Validation Framework
25	OCR-AI	AI-Based Document Understanding
26	Redis	Remote Dictionary Server (Caching System)
27	JWT	JSON Web Token
28	OCR-DAI	Document AI (Google Document AI)
29	KV Store	Key–Value Store (Cache/Config Storage)
30	API GW	API Gateway

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Organizations rely on policy documents for governance, compliance, and operational guidance, yet these documents are often lengthy, complex, and difficult to interpret. Employees and stakeholders frequently struggle to extract relevant information, compare policies, or generate training materials, leading to inefficiency and errors. To address these challenges, **PolyGPT** is developed as an AI-driven policy intelligence system that automates policy understanding, analysis, and interaction using Large Language Models (LLMs).

PolyGPT is built on a modular FastAPI architecture with clearly separated controllers, services, and agent layers. It integrates multi-cloud LLM providers—**OpenAI (GPT models)** and **AWS Bedrock (Titan, Nova)**—to perform summarization, rule extraction, codification, policy comparison, user query handling, and assessment generation. Policy embeddings are stored efficiently in **Pinecone**, client configurations and prompts are maintained in **MongoDB**, and **Redis** is used for caching and performance optimization. The system supports asynchronous processing with background tasks, webhook callbacks, and robust monitoring through Slack notifications.

By combining scalable backend engineering with advanced AI capabilities, PolyGPT transforms static policy documents into actionable, searchable, and comparable knowledge. It improves compliance, reduces manual effort, and provides organizations with a unified platform for policy summarization, understanding, and training. This project demonstrates the practical application of LLMs in enterprise policy management and showcases a flexible, multi-tenant, production-ready AI architecture

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Policies play a crucial role in shaping the operations, decision-making processes, and compliance standards of every organization. However, these documents are often long, complex, and written in technical language that can be difficult for employees to interpret quickly. In many real-world settings—such as finance, insurance, HR, and operations—staff must frequently refer to policies, compare updated versions, answer customer queries, or make decisions based on policy criteria. This creates a significant amount of manual work and increases the likelihood of human error. The PolyGPT project was undertaken to address these challenges by developing an intelligent, AI-driven system that simplifies how organizations interact with their policies.

The primary aim of PolyGPT is to build a unified platform that can automatically read, summarize, analyze, and reason over policy documents using state-of-the-art Large Language Models (LLMs). Instead of manually going through lengthy PDFs, employees can rely on PolyGPT to provide quick, context-aware, and accurate outputs. A major objective is to convert dense policy text into short, readable summaries that highlight the core rules, eligibility conditions, exceptions, and important clauses. This not only saves time but also helps employees make faster and more informed decisions.

Another key objective is to extract policy rules in a structured, machine-readable format. Organizations often need to automate workflows, create rule engines, or validate customer data against policy conditions. PolyGPT addresses this need by transforming policy text into JSON-based rule structures that can be integrated into automated systems. Alongside this, the system provides natural language question-answering, allowing users to ask questions such as "Can a customer with CIBIL 698 and CMR 5 take a loan?" and receive responses grounded strictly in the uploaded policy.

The project also focuses on analyzing policy changes over time. Different versions of the same policy may introduce modifications, removals, or new compliance requirements. PolyGPT includes a policy comparison module that identifies these differences clearly, helping compliance teams track updates efficiently. Additionally, for training and organizational learning, the system can generate MCQ-based assessments from policy documents, making it easier to test employee understanding without manually preparing questions.

From a technical perspective, the project emphasizes scalability, modularity, and real-world integration. The backend is developed using FastAPI, following a clean MVT-inspired structure with separate controllers, services, constants, and agent modules. This architecture allows each feature—summarization, rule extraction, Q&A, assessment generation, codification, and policy comparison—to operate independently while still functioning cohesively. Another major goal is to support multi-organization configurations, where each client has its own prompts, LLM model preferences, vector database settings, API keys, and webhook endpoints stored in MongoDB. This ensures flexibility and adaptability across different industries and use cases.

Overall, PolyGPT aims to serve as a comprehensive and industry-ready policy intelligence system. By combining LLMs, vector search, caching, document processing, and modular backend engineering, the project demonstrates how AI can be operationalized to reduce manual effort, eliminate ambiguity, accelerate compliance tasks, and improve organizational productivity. The introduction of PolyGPT represents a step toward transforming static policy documents into dynamic, interactive, and intelligent knowledge systems.

1.2 PROBLEM STATEMENT

Policies form the backbone of every organization. They define how processes run, how decisions are made, and how employees are expected to act in different situations. From HR guidelines and financial approval frameworks to risk assessment procedures and operational checklists, policies exist in every department and are constantly updated to reflect regulatory changes, compliance requirements, and organizational restructuring. Although policies are indispensable, they are often one of the most difficult internal documents for employees to interpret. Most policies are written in dense, formal language, filled with exceptions, sub-clauses, and legal constraints. This makes them extremely challenging to navigate, especially for employees who do not work with policies regularly.

In real organizational environments, employees rarely have the time to read an entire policy document when they only need one small piece of information. A significant portion of time is wasted manually searching through PDFs, scrolling through pages, and trying to interpret legal phrases that may not be immediately clear. In industries such as banking, insurance, NBFCs, telecom, and healthcare, policies are not just informational—they directly dictate customer eligibility, risk classification, interest rates, approval hierarchies, and compliance boundaries.

Employees in these sectors must follow policies precisely, because even a minor misinterpretation can lead to financial losses, compliance breaches, or customer dissatisfaction. The problem becomes even more complex when policies undergo frequent revisions. For instance, financial policies may be updated monthly due to regulatory mandates, market fluctuations, or internal risk evaluations. HR and compliance teams also update policies regularly based on new labor laws, operational changes, or audit findings. Employees are then expected to keep track of what has changed, sometimes without any clear explanation from the organization. Manually comparing two versions of a document—each consisting of dozens of pages—is not only time-consuming but also highly error-prone. Subtle modifications, such as changing a threshold value from 720 to 700 or replacing one clause with another, can easily go unnoticed. This creates a major risk in compliance-heavy industries, where a single missed update can affect hundreds of customer decisions.

Traditional document-processing tools are not effective in addressing these issues. Many existing solutions focus on only one capability at a time, such as:

- Summarization of content
- Basic question-answering
- Simple text extraction from PDFs
- OCR for scanning documents

However, real organizational needs are far broader and interconnected. A compliance officer may need a summary of a policy, a set of extracted rules, a comparison with an older version, and a natural-language explanation of a specific eligibility case—all within a single workflow. A training manager may want immediate MCQs generated from the policy so that employees can be evaluated on their understanding. A product manager may want structured JSON rules that can be integrated into a credit decision engine. Yet, no existing tool offers this entire spectrum together in a single, unified system.

Another challenge arises from scalability and adaptability. Many off-the-shelf AI tools rely on a single model provider, which limits flexibility. Organizations often need different AI models for different tasks—one model might be excellent at summarization, another at rule extraction, another at reasoning-based Q&A, and another at generating structured JSON outputs. Moreover, companies working in regulated sectors may prefer using on-premise or region-specific LLMs due to data privacy laws. Therefore, a system that can adapt to multiple LLM providers—such as AWS Bedrock, OpenAI, Google Gemini, or custom local models—becomes essential.

A major gap in current tools is the absence of multi-organization support. Most AI systems assume that there is a single global configuration applied to all users. In real scenarios, however, each organization has its own policies, its own preferred models, its own thresholds, and its own webhook or workflow integrations. A one-size-fits-all AI model cannot meet these varied demands. What organizations actually need is a system where each client can have:

- Custom prompts
- Custom LLM providers
- Custom model configurations
- Their own Pinecone vector index
- Their own similarity thresholds
- Their own webhook endpoints
- Their own prompt-engineering strategies

Without such configurability, AI systems become rigid and unsuitable for enterprise deployment. This is one of the biggest gaps PolyGPT aims to fill.

Another problem arises from the lack of structured rule extraction in existing systems. Policies typically contain rules regarding eligibility, exceptions, documentation requirements, risk conditions, approval limits, and financial formulas. These rules are often embedded within paragraphs rather than listed explicitly. Manual extraction of rules into JSON or decision-tree formats takes significant time and requires expert understanding. This extraction becomes even more challenging in policies spanning hundreds of pages. Organizations need an automated rule extraction engine that can convert these textual rules into structured formats appropriate for downstream systems like business rule engines (BREs) or credit-decision systems.

Similarly, many policies contain tabular data, diagrams, footnotes, and examples that are critical for interpretation. Existing AI tools are limited in their ability to understand such mixed-format content, especially when policies include scanned documents or images requiring OCR.

Without multimodal document understanding, automation remains incomplete.

Training employees on policies is another overlooked challenge. Organizations often conduct onboarding or re-training sessions where employees must learn new or revised policies. Preparing training material manually is extremely time-consuming, and human-generated MCQs tend to be repetitive or incomplete. An automated system that converts policy content into assessments would dramatically reduce the load on HR and training teams while improving consistency.

When we examine all these fragmented problems collectively, it becomes clear that organizations lack a comprehensive system capable of managing policies end-to-end. The absence of such a system leads to:

- Wasted employee time
- Inconsistencies in interpretation
- Errors in decision-making
- Higher compliance risks
- Delays in customer service
- Difficulty in training large teams
- Fragmented workflows involving multiple tools

The need is not just for an AI tool but for a full policy-intelligence ecosystem that can read, summarize, extract, compare, reason, answer questions, generate assessments, and integrate into existing enterprise systems.

PolyGPT is designed to address exactly this gap. It is built as a multi-agent, multi-organization, multi-LLM framework that handles policy documents intelligently and holistically. Unlike existing solutions, PolyGPT provides:

- Summarization agent (to compress long policies into readable summaries)
- Rulify agent (to extract and structure rules in JSON)
- User Query agent (for policy-grounded natural language Q&A)
- Policy Comparison agent (to find differences between two policy versions)
- Assessment agent (to create MCQs)
- Codify agent (to convert textual instructions into SQL/JSON logic)
- Keys Mapper agent (to map variables to policy rules)
- BRE Rules agent (to structure complex logic for rule engines)
- OCR / Document AI agent (to parse scanned/internal documents)

All of these modules work together in a clean MVT architecture powered by FastAPI controllers, service layers, and agent modules. MongoDB is used to manage organization-specific configurations, Pinecone is used for vector search, Redis is used for caching, and the system integrates multiple LLM frameworks such as Amazon Nova, AWS Titan, and OpenAI models including GPT-4o and o3-mini.
The main problem being addressed is not just document summarization or question-answering. The problem is the entire complexity of policy interpretation, which includes reading, extracting, comparing, reasoning, evaluating, and teaching policy content. No existing tool provides this complete spectrum with the flexibility needed for real-world enterprise environments. PolyGPT fills this void by offering a comprehensive, modular, and scalable solution.

The relevance of this problem is increasing rapidly. As industries become more regulated and policies grow more detailed, manual interpretation becomes nearly impossible. Organizations need intelligent systems that can keep up with policy updates, reduce compliance risks, support employees in decision-making, and streamline policy-driven workflows. PolyGPT attempts to solve exactly this by transforming static policy documents into dynamic, interactive knowledge systems.

By addressing the gaps in existing solutions, PolyGPT aims to significantly reduce manual workload, ensure consistency in policy interpretation, minimize compliance errors, and improve operational efficiency across organizations. It brings together AI reasoning, rule extraction, document processing, multi-organization support, and scalable backend architecture into a unified, enterprise-ready platform. This makes PolyGPT not just another document-processing tool, but a complete solution to the long-standing challenges associated with policy management.

1.3 OBJECTIVES

The development of PolyGPT began with the understanding that traditional methods of interacting with organizational policies are inefficient, inconsistent, and heavily dependent on manual effort. To address these challenges and transform how employees interpret, analyze, and utilize policy documents, the project was built around a structured set of objectives. These objectives are technical, functional, architectural, and organizational in nature. Together, they define the scope of PolyGPT as a complete and intelligent policy-automation system capable of serving real enterprise needs.

The primary objective of PolyGPT is to create a unified AI-driven platform that can intelligently process complex policy documents and present information in a clear, structured, and actionable format. Policies often span dozens of pages and contain multiple nested sections, sub-clauses, exceptions, and logical conditions. Understanding them requires not just textual reading but deeper reasoning and contextual interpretation. One of the core goals of this project is to enable Large Language Models (LLMs) to perform this reasoning automatically, thereby reducing the cognitive load on employees and minimizing errors in interpretation.

A major functional objective is **policy summarization**. The system aims to extract essential information from large documents and present it in a simplified, human-readable summary. The intention is to allow users to grasp the core intent, rules, and conditions of a policy without having to read through lengthy PDFs. This summary should retain accuracy while removing unnecessary complexity. Furthermore, PolyGPT aims to support interactive summarization, where users can refine the summary, ask clarifying questions, or request modifications, ensuring a more conversational and dynamic experience.

Another key objective is **rule extraction**, referred to as the "Rulify" process in PolyGPT. Organizational workflows rely heavily on discrete policy rules such as eligibility criteria, thresholds, exceptions, and approval requirements. These rules are often buried in paragraphs of text rather than being presented clearly. PolyGPT is designed to extract these rules and convert them into structured formats like JSON, making them easier to integrate into decision engines, automation pipelines, and compliance monitoring systems. This enables organizations to move from manual interpretation to machine-driven rule enforcement, improving consistency and accuracy.

A third major objective is to enable **policy-grounded question-answering (Q&A)**. In many scenarios, employees need to query policies in natural language, such as “What is the minimum CIBIL score required for this loan?” or “Is deviation allowed if income is below the threshold?” PolyGPT aims to retrieve answers that are *strictly grounded* in the uploaded policies rather than general AI knowledge. This requires not only natural language understanding but also contextual reasoning and reference tracking to the exact clause or section in the policy. The objective is to bridge the gap between long, static documents and real-time decision needs.

In addition to interpretation, organizations often need to compare policy versions over time. Policies evolve due to regulatory updates, risk management revisions, or procedural changes. One of the essential objectives of PolyGPT is to implement **policy comparison**, allowing users to compare two documents and identify modifications, additions, deletions, and threshold changes. The system aims to present these differences in a clear, structured format, reducing the manual burden of comparing long documents line-by-line.

Training and assessments are also critical for ensuring policy understanding across an organization. Therefore, PolyGPT aims to support **assessment automation**, where the system can generate MCQs and other evaluation questions directly from policy content. This helps HR teams, trainers, and auditors create learning materials efficiently and ensures that employees are tested on relevant and accurate content. The objective is to make training more scalable, consistent, and data-driven.

Beyond functional features, the project also incorporates several technical and architectural objectives. A major architectural goal is to build the system using a **clean MVT-inspired design** that separates controllers, services, constants, agents, utility modules, and configuration layers. This ensures that the backend is modular, maintainable, and scalable. Each policy-processing functionality is implemented as an independent agent, allowing new features to be added or modified without affecting the entire system. The objective is to create a backend that is not just functional but also production-ready and maintainable in the long term.

Scalability is another core objective. Organizations differ in their model preferences and operational requirements. Some may want to use OpenAI models, while others prefer Amazon Nova or Titan due to pricing, latency, or compliance constraints. Therefore, PolyGPT aims to support **multiple LLM providers**, allowing dynamic selection of cloud providers and models per organization. This flexibility ensures better performance tuning, cost optimization, and compliance with industry-specific requirements.

A related objective is to design a **multi-organization configuration layer** powered by MongoDB. This allows each organization to store its own prompts, vector index names, model configurations, API keys, threshold values, workflow URLs, and feature flags. PolyGPT aims to load these configurations dynamically for every request, ensuring isolation between clients and enabling a high level of customization. This makes the system practical for adoption across different sectors such as NBFCs, banks, HR firms, insurers, and operational teams.

PolyGPT also aims to integrate with **vector databases** like Pinecone to support deeper search, context retrieval, and knowledge-base creation. A goal is to structure policy information into embeddings, enabling fast similarity search and enhanced grounding for Q&A. Additionally, the system aims to utilize **Redis caching** to reduce processing time for repeated summarization or rule-extraction requests, improving system efficiency and reducing cloud-model usage costs. Error handling and reliability form another important category of objectives. PolyGPT aims to include robust exception handling, rate limiting, request validation, and log tracing. The backend is designed to manage long-running tasks using asynchronous execution, ensuring that large document processing does not block system performance. Slack alerts, webhook notifications, and request-level tracing help maintain operational visibility in production environments.

Another key objective is to ensure **ease of integration** with existing enterprise workflows. PolyGPT aims to support webhooks, asynchronous task dispatch, and modular service calls so that external systems—like CRM portals, loan origination systems, HR dashboards, or training platforms—can easily consume PolyGPT outputs. The goal is not just to create an AI model but to create an entire backend service that can be seamlessly plugged into real-world applications.

On the usability front, PolyGPT aims to provide intuitive input formats, clear API outputs, and easy-to-interpret structured responses. Users should not require technical expertise to interact with the system. All they need to do is upload a policy document or provide a query; the system then handles text extraction, summarization, reasoning, and formatting on its own.

Finally, one of the overarching objectives is to build an **industry-ready policy intelligence system** that has the potential to be deployed at scale. The system aims to serve diverse use cases such as risk assessment, policy training, compliance verification, customer support, and operational decision-making. By integrating AI into these traditionally manual processes, the objective is to significantly reduce human effort, minimize errors, and improve decision-making accuracy across organizations.

In summary, the objectives of PolyGPT span across multiple dimensions:

- *Functional objectives* like summarization, rule extraction, Q&A, comparison, and assessment generation
- *Technical objectives* like multi-LLM support, caching, vector indexing, asynchronous processing, and clean architecture
- *Organizational objectives* like multi-client configurations and workflow integration
- *Performance objectives* like speed, consistency, accuracy, and reliability

Together, these objectives ensure that PolyGPT is not just a single feature or AI model, but a complete ecosystem designed to transform policy interpretation and automation for modern enterprises.

1.4 SIGNIFICANCE AND MOTIVATION OF THE PROJECT WORK

Policies form the invisible framework that supports the daily functioning of every organization. Whether in finance, banking, insurance, HR, healthcare, telecom, or operations, policies determine how decisions are taken, how risks are assessed, and how compliance is maintained.

Despite playing such a fundamental role, policies are often among the *least* accessible documents for the people who rely on them the most. Lengthy PDFs, legalistic language, complex approval hierarchies, exceptions buried within paragraphs, version updates, and ambiguous interpretations often result in operational delays and decision-making errors. This widespread difficulty in interpreting policies forms the core motivation behind the PolyGPT project.

The significance of this project arises from a very practical observation: **employees cannot afford to read dozens of pages every time they need clarity on a policy**, yet they are required to make accurate decisions based on those very documents. A credit officer deciding on loan eligibility, an HR executive processing a leave request, a compliance officer checking rule changes, or an auditor verifying procedures—all of them need immediate, precise, and policy-grounded answers. Unfortunately, existing tools fall drastically short of providing this support. This gap between *policy availability* and *policy usability* is what makes PolyGPT not just an academic exercise, but an extremely relevant real-world solution.

Motivation Based on Real Organizational Challenges

The idea for PolyGPT was motivated by repeated issues observed in real organizations. Employees consistently struggle to locate specific clauses or conditions. Policies may specify rules such as "CIBIL score must be >700 unless deviation approved," but such conditions are scattered across documents, requiring significant time to manually search and interpret. Employees often interpret similar policies differently, resulting in inconsistencies that impact customer decisions and operational integrity.

Beyond interpretation, another challenge arises when policies evolve. Policy changes do not always come with highlight markings or summaries. Employees have to detect differences manually, which is error-prone and extremely time-consuming. Organizations with large staff bases—such as banks and NBFCs—face major inefficiencies due to repeated training sessions required to explain policy updates. This motivates the need for an automated system that identifies changes and communicates them clearly.

Organizations also struggle with **rule extraction**, a process in which textual information from policies must be rewritten into structured formats. This is necessary for downstream integrations such as loan decision engines, payroll calculations, insurance claim systems, or HR automation tools. Converting multiple pages of textual rules into JSON, tables, or structured decision trees manually requires hours of effort and subject-matter expertise. Motivated by this inefficiency, PolyGPT introduces a Rulify module that automates rule extraction with LLM intelligence.

Another source of motivation stems from the recognition that employees, despite being experts in their own domains, are not always trained to understand complex policy language. Policies are often written from a legal or compliance standpoint, not from the perspective of an everyday employee who needs clarity. This mismatch between writing style and real-world accessibility motivated the development of PolyGPT's summarization and natural language Q&A modules. Summaries help employees quickly understand the essential components without reading the full text, while question-answering provides a conversational interface that simplifies the entire interaction.

Technological Motivation and Current Limitations in Existing Tools

A strong motivational driver for this project is the rapid advancement of artificial intelligence technologies, especially Large Language Models. While LLMs like GPT-4o, Amazon Nova, and Titan offer powerful capabilities, organizations still struggle with applying them effectively. Generic AI chatbots are not equipped to understand *organization-specific* policies, nor can they ensure that responses are grounded only in uploaded documents. This leads to hallucinated results, which are dangerous in compliance-sensitive environments. Therefore, one major motivation for PolyGPT is the ability to take cutting-edge LLM technology and apply it safely, responsibly, and accurately in the context of policy interpretation.

Existing AI tools are also limited in scope. Many platforms can summarize documents, and some can answer general queries. But none provide the full spectrum of policy intelligence required in actual enterprise environments, such as rule extraction, structure generation, SQL codification, policy comparison, knowledge-base management, and MCQ creation. These gaps motivated the development of a system that unifies all capabilities within one architecture, enabling seamless workflows from reading documents to extracting rules and generating assessments.

Another technical motivation is **multi-LLM flexibility**. Organizations may prefer one cloud provider over another for reasons such as cost, latency, performance, security, or compliance. Most existing tools lock users into one specific model or service. PolyGPT overcomes this by supporting multiple cloud providers and models, allowing each organization to choose the model that best fits their needs. This flexibility forms a crucial part of the system's motivation.

Motivation from Software Architecture and Scalability Perspective

As AI adoption grows, organizations need systems that are robust enough to handle high traffic, multiple users, asynchronous workloads, and complex configurations. However, many AI-based prototypes lack proper architectural design, making them unsuitable for real-world deployment. A key motivation behind PolyGPT is adopting a clean MVT-style architecture that separates controllers, services, agents, and configurations. This structure makes the system scalable, maintainable, and easy to extend.

Multi-organization support is another architectural motivation. A single deployment of PolyGPT can serve many organizations, each with unique prompts, vector indexes, LLM models, and webhook URLs. This requires a strong dynamic configuration layer powered by MongoDB. Implementing this became a critical objective and motivation since real enterprise systems must support varied clients without mixing their data or configuration logic.

The system's ability to handle long-running tasks using asynchronous processing is motivated by the need to make document processing efficient and user-friendly. Large PDFs, OCR-based extraction, and summarization requests would normally cause blocking delays. PolyGPT's architecture solves this through background task execution and webhook-based results delivery, creating a smoother user experience.

Industry Relevance and Broader Social Impact

The significance of PolyGPT extends beyond technical innovation. Policies have far-reaching impacts. In banking and financial services, a misinterpreted policy can lead to incorrect loan approvals, risk exposure, or loss of revenue. In HR, incorrect policy interpretation can lead to employee dissatisfaction, payroll errors, or legal disputes. In insurance, misunderstanding claim eligibility criteria can impact customer trust. Therefore, improving policy interpretation is not just a matter of convenience—it directly affects organizational efficiency, fairness, compliance,

and customer experience.
By automating and simplifying the policy workflow, PolyGPT contributes to reducing human error and improving transparency. Employees can make decisions more confidently. Organizations can ensure that rules are followed consistently. This elevates the overall governance and operational quality.

From a broader perspective, PolyGPT also contributes to digital transformation efforts in India and globally. Many organizations are moving toward automation, AI integration, and smart document processing. PolyGPT aligns with this trend by providing a practical AI solution grounded in realistic workflows.

Motivation Towards Learning, Research, and Academic Growth

Another important aspect of this project is its value as a learning and research experience. The implementation required understanding a broad spectrum of concepts, including:

- Large Language Models
- Prompt engineering
- Vector databases (Pinecone)
- Caching systems (Redis)
- FastAPI backend architecture
- Multi-organization configuration design
- OCR and Document AI
- JSON rule extraction
- Asynchronous task execution
- Clean code architecture and modular programming

Building PolyGPT required integrating all these fields into a cohesive and production-ready system. This multi-disciplinary exposure was a strong motivating factor for undertaking such a comprehensive project.

Additionally, working with live APIs, cloud providers, and model configurations provided a practical sense of how enterprise AI systems are deployed in the real world. This deepened the technical understanding of scalable backend systems, making the project valuable from an academic standpoint.

Long-Term Significance and Future Potential

PolyGPT is not just a final-year project—it has potential as a fully deployable enterprise product. The system is designed in a modular and expandable manner. New modules such as risk scoring, compliance automation, sentiment analysis, document classification, or real-time decision suggestions can be added easily. This scalability enhances the long-term significance of the project.

The industry landscape is evolving rapidly, and organizations increasingly demand AI solutions that are safe, domain-specific, and customizable. The work done in PolyGPT addresses these emerging needs. In the future, PolyGPT could be extended to:

- Integrate with legacy systems
- Provide policy-change alerts
- Offer multilingual summaries
- Support voice-based policy queries
- Enable on-premise private LLM deployments
- Build a full policy governance dashboard

Each of these future directions adds to the motivation behind the project and highlights its potential impact.

Conclusion of Significance and Motivation

The significance of PolyGPT lies in its ability to convert static, complex policy documents into dynamic, accessible, and intelligent knowledge systems. The motivation comes from real organizational pain points—time-consuming manual interpretation, inconsistency across employees, difficulty comparing versions, lack of comprehensive tools, and the growing complexity of policies. Coupled with technological motivations such as leveraging modern LLMs, modular architecture, multi-organization flexibility, and enterprise-level scalability, the project stands out as a meaningful and forward-looking solution.

PolyGPT is important because it addresses a real, large-scale problem that affects almost every organization today. It uses cutting-edge AI techniques to simplify policy workflows, improve decision accuracy, enhance compliance, and reduce operational burdens. The motivation behind the project goes beyond academic requirements—it reflects a genuine need for intelligent, automated systems in the enterprise world. PolyGPT fills this gap and lays the foundation for future advancements in AI-powered document understanding.

1.5 ORGANIZATION OF PROJECT REPORT

This report is structured in a way that gradually builds up the reader's understanding of PolyGPT, starting from the background and moving toward design, implementation, and evaluation. Each chapter focuses on a specific aspect of the system so that the project's development journey is easy to follow.

Chapter 1 introduces the problem of policy complexity and highlights why organizations struggle with long and technical documents. It outlines the goals of PolyGPT and explains the broader motivation behind building an AI-driven multi-agent policy system.

Chapter 2 presents the literature review, summarizing the research behind document summarization, rule extraction, OCR, multi-agent LLM systems, and policy automation. The chapter also identifies limitations in existing approaches and clearly points out where innovation is needed.

Chapter 3 describes the system requirements, including functional requirements like summarization, rule extraction, comparison, SQL reasoning, assessment creation, and OCR verification. It also covers non-functional requirements such as security, scalability, and multi-organization support.

Chapter 4 discusses the system architecture in depth. It explains how PolyGPT uses FastAPI with controllers, services, and agent modules, along with MongoDB, Pinecone, Redis, and multi-cloud LLMs. The chapter includes diagrams to show the multi-agent workflow, Donna-core components, and data flow.

Chapter 5 focuses on the implementation details, showing how each agent works, how prompts are managed, and how asynchronous tasks and webhooks are handled.

Chapter 6 covers the testing and evaluation process, including Postman route testing, unit tests, and multi-agent behavior checks.

CHAPTER 2

LITERATURE REVIEW

2.1 OVERVIEW OF RELEVANT LITERATURE

2.1.1. Transformer Architecture — Vaswani et al.

Vaswani et al. [1] introduced the Transformer model, which became the foundation of modern LLMs. Its attention mechanism significantly improves text understanding but does not offer specialized support for policy reasoning or rule extraction.

2.1.2. BERT for Bidirectional Understanding – Devlin et al.

Devlin et al. [2] presented BERT, enabling strong contextual understanding. However, its input size limitations make it unsuitable for long policy documents.

2.1.3. GPT Models for Generative Reasoning – Brown et al.

Brown et al. [3] demonstrated GPT-3's impressive generative and reasoning capabilities. Still, GPT alone cannot perform orchestrated tasks like comparison or rule structuring.

2.1.4. T5 Unified Text-to-Text – Raffel et al.

Raffel et al. [4] proposed T5 for unified NLP tasks. Despite strong summarization performance, it lacks built-in policy-oriented rule extraction.

2.1.5. Long Document Processing – Zaheer et al. (BigBird)

Zaheer et al. [5] improved long-sequence handling through sparse attention, helpful for lengthy documents but without domain reasoning.

2.1.6. Legal Summarization – Carbone et al.

Carbone et al. [6] analyzed legal judgments using hierarchical transformers. However, legal summarization models are not optimized for HR or compliance policies.

2.1.7. Legal-BERT – Zhong et al.

Zhong et al. [7] created Legal-BERT for legal QA tasks. While domain-adapted, it still lacks multi-step policy workflows.

2.1.8. RAG (Retrieval-Augmented Generation) – Lewis et al.

Lewis et al. [8] introduced RAG by combining retrieval with generation. RAG improves factual grounding but cannot perform structured reasoning like policy codification.

2.19. LLM-based Summarization Studies – Chen et al.

Chen et al. [9] highlighted LLM summarization success, but the outputs remain textual and not structured for compliance audits.

2.1.10. OCR Systems – Smith (Tesseract)

Smith [10] improved OCR accuracy with Tesseract. However, OCR does not handle semantic policy interpretation.

2.1.11. Google Document AI

Google Document AI [11] performs layout extraction and entity identification but falls short in rule extraction or policy comparison.

2.1.12. AWS Textract

AWS Textract [12] extracts tables and forms efficiently but lacks deeper policy understanding or rule segmentation.

2.1.13. Pinecone Vector Database

Pinecone [13] offers high-performance vector search for large-scale retrieval, but retrieval alone cannot infer rules or generate assessments.

2.1.14. Redis for Caching

Redis [14] is widely used for caching and queue management; however, it does not provide domain reasoning.

2.1.15. LangChain LLM Orchestration

LangChain [15] introduced chains and agents for LLM workflows. Despite its utility, it suffers from performance overhead and lacks multi-tenant design.

2.1.16. LlamaIndex for Document Indexing

LlamaIndex [16] simplifies indexing and retrieval for LLMs. Still, it lacks integrated rule extraction and codification tools.

2.1.17. LangGraph Multi-Agent Workflow

LangGraph [17] enables dependable multi-agent graph workflows but does not include domain-specific policy agents.

2.1.18. LangSmith for LLM Debugging

LangSmith [18] provides evaluation and debugging for LLM pipelines, though it is not a reasoning engine.

2.1.19. Multi-Agent AI Systems – ReAct

Yao et al. [19] proposed ReAct to combine reasoning and acting. While powerful, it isn't tailored to enterprise policy operations.

2.1.20. AutoGPT Autonomous Agents

AutoGPT [20] showcased autonomous multi-step execution but lacks constraints required for compliance tasks.

2.1.21. LLM-based Quiz Generation – Qiu et al.

Qiu et al. [21] studied automated quiz generation using transformers. However, most models cannot generate policy-specific MCQs reliably.

2.1.22. Business Rule Extraction – Khan et al.

Khan et al. [22] explored rule extraction from documents using NLP. These methods struggle with complex conditional policy rules.

2.1.23. Policy Comparison Research – Li et al.

Li et al. [23] developed basic document comparison techniques. Still, they fail when comparing nuanced policy clauses.

2.1.24. Codification and Structured Output – Lamm et al.

Lamm et al. [24] studied converting legal text into logic formats, but scaling such systems is challenging.

2.1.25. Multi-LLM Cloud Systems – Zhao et al.

Zhao et al. [25] analyzed multi-cloud LLM setups for performance tuning. However, no existing framework directly supports policy-specific multi-organization configurations.

Table 2.1 Literature Review Table

S. No.	Author / Tool	Citation	Journal/Source	Techniques	Key Findings	Limitations / Gaps
1	Vaswani et al., Transformer	[1]	NeurIPS (2017)	Self-attention	Foundation of LLMs	No policy-specific reasoning
2	Devlin et al., BERT	[2]	NAACL (2019)	Bidirectional Transformers	Strong contextual understanding	Cannot process long policies
3	Brown et al., GPT-3	[3]	NeurIPS (2020)	Generative Models	Excellent generation	Cannot run multi-step workflows
4	Raffel et al., T5	[4]	JMLR (2020)	Seq2Seq	Good summarization	Lacks structured rule extraction
5	Zaheer et al., BigBird	[5]	NeurIPS (2020)	Sparse Attention	Handles long inputs	No compliance reasoning
6	Carbone et al., Legal Summarization	[6]	AI & Law (2022)	Hierarchical NLP	Good legal summaries	Not HR/compliance focused
7	Zhong et al., Legal-BERT	[7]	EMNLP (2020)	Domain Transformers	Better legal QA	Only QA, no workflows
8	Lewis et al., RAG	[8]	NeurIPS (2020)	Retrieval + Gen	Improves factual grounding	No rule extraction
9	Chen et al., Summarization	[9]	ACL (2021)	Abstractive Models	Strong summaries	No structured outputs
10	Smith, Tesseract OCR	[10]	Google (2019)	OCR	Good extraction	No semantic processing

S. No.	Author / Tool	Citation	Journal/Source	Techniques	Key Findings	Limitations / Gaps
11	Google Document AI	[11]	Google Cloud	OCR+NER	Layout understanding	No policy reasoning
12	AWS Textract	[12]	Amazon AWS	Form/Table Extraction	Accurate extraction	No semantic comparison
13	Pinecone DB	[13]	Pinecone Docs	ANN Search	Fast vector search	Retrieval-only
14	Redis Cache	[14]	Redis Labs	In-memory DB	Improves speed	Not an AI engine
15	LangChain	[15]	GitHub	Chains, Agents	Workflow support	Not multi-tenant
16	LlamaIndex	[16]	GitHub	Indexing	Good RAG infra	No rule generation
17	LangGraph	[17]	LangChain	Graph Agents	Robust workflows	No policy modules
18	LangSmith	[18]	LangChain	Debugging LLMs	Pipeline testing	Not a reasoning model
19	ReAct Agents	[19]	ICLR (2023)	Reasoning+Acting	Better task execution	Not domain-safe
20	AutoGPT	[20]	GitHub	Autonomous Agents	Multi-step tasks	No compliance safety
21	Qiu et al., Quiz Generation	[21]	IEEE Access (2022)	Transformers	Automatic MCQs	Not policy-specific
22	Khan et al., Rule Extraction	[22]	Expert Syst. (2021)	NLP Pipelines	Extract simple rules	Cannot handle complex clauses
23	Li et al., Document Comparison	[23]	arXiv (2021)	Similarity Models	Identifies differences	Weak on nuanced policies
24	Lamm et al., Text-to-Logic	[24]	ACL (2021)	Logical Parsing	Converts legal text	Hard to scale
25	Zhao et al., Multi-Cloud LLM	[25]	IEEE Cloud (2023)	Cloud LLMs	Performance tuning	Not multi-organization

2.2 KEY GAPS IDENTIFIED IN THE LITERATURE

Despite strong advancements in NLP, document processing, and LLM development, several gaps remain specifically in the domain of policy intelligence systems. Transformers, BERT, and long-document models like BigBird perform well for general NLP tasks but cannot handle multi-step reasoning required for extracting structured policy rules or evaluating compliance implications [1–5]. Legal-domain models such as Legal-BERT achieve improved accuracy but remain single-task systems incapable of workflows involving summarization, comparison, and codification simultaneously [6–7].

RAG frameworks enhance factual grounding but still operate only at the retrieval + generation level, without supporting higher-level tasks like policy codification, BRE rules, or multi-version comparison [8–9]. OCR technologies like Tesseract and Textract extract content but do not interpret policies semantically [10–12]. Tools like Pinecone, Redis, and vector search systems improve retrieval speed but cannot convert retrieved text into meaningful rule structures [13–14].

Although LangChain, LlamaIndex, LangGraph, and AutoGPT have enabled agent-based architectures, they still lack enterprise features such as organization-level configurations, policy-specific rule agents, assessment generators, and compliance validation workflows [15–20]. Existing work on rule extraction, quiz generation, document comparison, and text-to-logic conversion addresses isolated tasks but does not provide an integrated, multi-agent solution capable of handling full policy lifecycles (summarize ? extract ? compare ? codify ? assess) [21–24].

There is no unified platform in literature that brings together all essential capabilities: OCR validation, multi-agent reasoning, multi-cloud LLMs, vector retrieval, structured rule extraction, policy comparison, and assessment generation. PolyGPT aims to bridge precisely this gap.

CHAPTER 3
SYSTEM DEVELOPMENT

3.1 REQUIREMENTS AND ANALYSIS

PolyGPT requires a modern software stack capable of handling API calls, LLM interactions, OCR pipelines, vector search, and multi-tenant configuration management. The core system is built on **FastAPI**, with model validation through **Pydantic**, and LLM orchestration powered by **LangChain, LiteLLM, and LangGraph**. Policy embeddings and semantic retrieval run on **Pinecone**, while **MongoDB** stores organisation-specific configuration, prompts, and metadata. To support real-time performance, **Redis** is used for caching and temporary state management, and OCR workflows rely on **PyMuPDF, pdf2image**, and optional **Google Document AI** or **AWS Textract** integrations. These components ensure scalability, fast response times, and consistent outputs even for large policy documents.

From a hardware perspective, PolyGPT does not require heavy compute on the server side because the most expensive operations—LLM inference—are executed on cloud providers like OpenAI or AWS Bedrock. A typical production workload can run efficiently on an 8 vCPU / 32 GB RAM machine, with SSD storage for faster document processing. As concurrency scales, horizontal autoscaling and a managed Redis/MongoDB setup ensure reliability. Good network bandwidth is essential because the system communicates with external LLM APIs, OCR services, and webhook endpoints. Overall, the requirements are reasonable and well-balanced, allowing PolyGPT to handle enterprise workloads while remaining cost-efficient and modular.

Table 3.1 Software Requirements Table

Category	**Package / Tool**	**Purpose in PolyGPT**
Backend Framework	fastapi, uvicorn, starlette	API server, async routing, request handling

Category	Package / Tool	Purpose in PolyGPT
Data Models Config	pydantic, annotated-types, python-dotenv	DTO validation, settings, environment management
LLM Orchestration	openai, litellm, langchain, langchain-core, langchain-openai, langchain-aws, langchain-litellm, langgraph	Interacting with LLMs, multi-agent workflows, prompt management
RAG/Vector Search	pinecone, langchain-pinecone	Embedding storage, semantic search for policies
Databases	pymongo, dnspython	Multi-tenant configs, prompts, user metadata
Caching & Task State	redis	Caching, temporary storage, background pipeline speed-up
Cloud Integrations	boto3, botocore, s3transfer, google-cloud-documentai, google-cloud-vision	AWS Bedrock/Textract, S3 presigned URLs, Google OCR
OCR & Document Processing	PyMuPDF, pdf2image, pikepdf, pillow, pdfkit	Extracting text from PDFs, preprocessing for summarization/rulify
Networking & Requests	httpx, requests, aiohttp, tenacity	API calls, retries, webhook delivery

Category	Package / Tool	Purpose in PolyGPT
Data Processing	pandas, pyarrow, nltk	Text cleaning, analytics, dataset handling
Validation & Parsing	jsonschema, protobuf, proto-plus	Structured JSON output validation from LLM
Logging & Observability	rich, tqdm, custom Slack notifier	Debugging, monitoring, developer visibility
Security & Auth	rsa, pyasn1, certifi	Secure connections, SSL, token validation
Rate Limiting	slowapi	Protecting endpoints & LLM cost control
Development Tools	GitPython, pikepdf, pydeck	DevOps automation, file manipulation, optional visualizations

Table 3.2 Hardware Requirements Table

Environment	CPU	RAM	Storage	Network Requirements
Development	4 vCPUs	8–16 GB	50–100 GB SSD	Stable broadband, 10–50 Mbps

Environment	CPU	RAM	Storage	Network Requirements
Production (Small Org)	8 vCPUs	32 GB	200+ GB SSD	100 Mbps+, low latency to cloud LLM APIs
Scalable / Enterprise	16+ vCPUs	64–128 GB	500 GB SSD or NVMe	High throughput, autoscaling-enabled
Optional	NVIDIA T4 / A10 GPU	—	—	For local LLM inference (if needed)

3.2 PROJECT DESIGN AND ARCHITECTURE

PolyGPT is designed as a modular, production-ready policy intelligence platform that follows a clear MVT-like separation of concerns: **Controllers → Services → Agents**, backed by dedicated configuration and data layers. The controller layer (FastAPI) exposes REST endpoints for each functional route (summarize, rulify, codify, user_query, policy_comparison, assessment, generate_options, test_bed), performs authentication and input validation, and immediately acknowledges long-running jobs. The service layer implements business logic, orchestrates workflows, manages background tasks and timeouts, communicates with webhooks and Slack for observability, and glues together data access with LLM-driven agents. Agents encapsulate domain-specific intelligence — summarization, BRE-rule extraction, SQL reasoning, OCR verification, comparison processing, MCQ generation, and multi-agent Donna-core workflows — and are responsible for prompt management and LLM interactions (OpenAI / AWS). Persistent configuration and tenant metadata live in **MongoDB**; semantic retrieval uses **Pinecone** for embeddings; **Redis** provides caching, ephemeral state, and fast coordination for document processing. Prompts, constants, and DTOs are centralized for versioning and reuse. The architecture favors extensibility: new agents or routes plug in without changing controllers; services mediate common concerns (retries, error handling, auth); and agents remain replaceable adapters to different LLM providers. This design balances engineering concerns (scalability, maintainability, multi-tenancy) with research needs (multi-cloud LLM experiments, multi-agent orchestration), enabling PolyGPT to handle end-to-end policy lifecycles from ingestion and OCR to structured rule codification and assessment generation.

3.2.1. High-level Layers

- Presentation / Controller Layer
  - FastAPI routes under controller/ (policy + generic polygpt controllers).
  - Responsibilities: routing, auth (API keys), request validation, immediate ack for async tasks.
- Business Logic / Service Layer
  - Services under services/ (summarize_service, rulify_service, querify_service, etc.).
  - Responsibilities: orchestration, calling agents, Redis caching, webhook posting, error handling, Slack alerts.
- Agent / Knowledge Processing Layer
  - Agents under agents/ (policy/*, generic/polygpt, ocr_verification).
  - Responsibilities: LLM calls, prompt templates, structured JSON outputs, OCR pre/post-processing, multi-agent workflows (Donna-core).
- Data & Infrastructure Layer
  - MongoDB: tenant configs, prompts, API keys (secretmanager + client_config_service).
  - Pinecone: embedding store for semantic retrieval.
  - Redis: caching, short-lived task state, document processor optimization.
  - Object Storage (S3): policy files, presigned uploads.
  - LLM Providers: OpenAI and AWS Bedrock (Titan, Nova); liteLLM/langchain adapters.

- Support Utilities
    - constants/ for endpoints and policy constants.
    - dto/ for request/response schemas.
    - utils/ for logging, webhook client, slack notifier, PDF/OCR helpers.
    - configs/ and secretmanager/ for environment & secrets.

3.2.2. Component Interactions (Typical Flow)

- Upload / Ingest: client uploads policy via presigned URL → controller stores metadata in MongoDB and triggers embedding & OCR agent.
- Indexing: agent extracts text, runs OCR if needed, generates embeddings → upserts into Pinecone.
- Request Handling: user calls route (eg. /summarize) → controller authenticates, service ack's and spawns background task.
- Processing: service calls appropriate agent → LLM + prompt → structured response (JSON) or summary returned and optionally posted to webhook.
- Caching & Persistence: results cached in Redis, long-term outputs saved in MongoDB or object store.
- Monitoring: Slack/webhook notifications for errors / completions; logs stored via utils/logger.

3.2.3. Extensibility & Multi-Tenancy

- Config-driven agents: client-specific prompts and model choices stored in MongoDB; client_config_var context selects per-request settings.
- Provider abstraction: liteLLM/langchain wrappers allow switching or A/B testing between OpenAI and AWS.
- Pluggable agents: add new agents (e.g., InfoSec checks, SQL processors) by implementing the agent interface and registering in services.

4. Fault Tolerance & Operational Concerns

- Asynchronous background tasks with timeouts (RunTimeout) to avoid blocking controllers.
- Rate limiting using SlowAPI when available.
- Retries via tenacity for transient LLM/API errors.
- Graceful error handlers and Slack alerts for operational visibility.

Figure 3.1 Architecture Layer Diagram

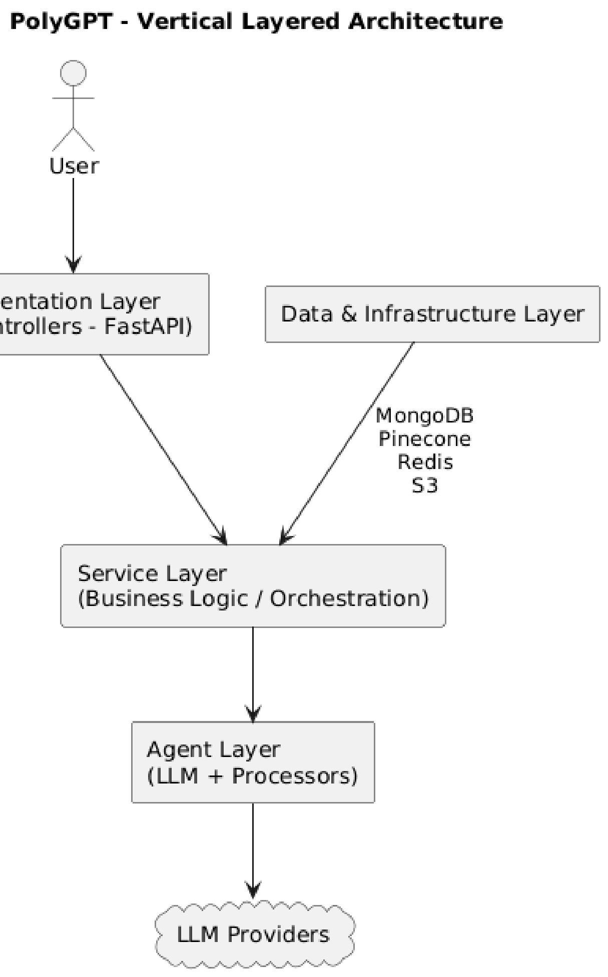

PolyGPT - Vertical Layered Architecture
User
Presentation Layer (Controllers - FastAPI)
Data & Infrastructure Layer
MongoDB Pinecone Redis S3
Service Layer (Business Logic / Orchestration)
Agent Layer (LLM + Processors)
LLM Providers

Figure 3.2 Sequence Diagram — Summarize Flow

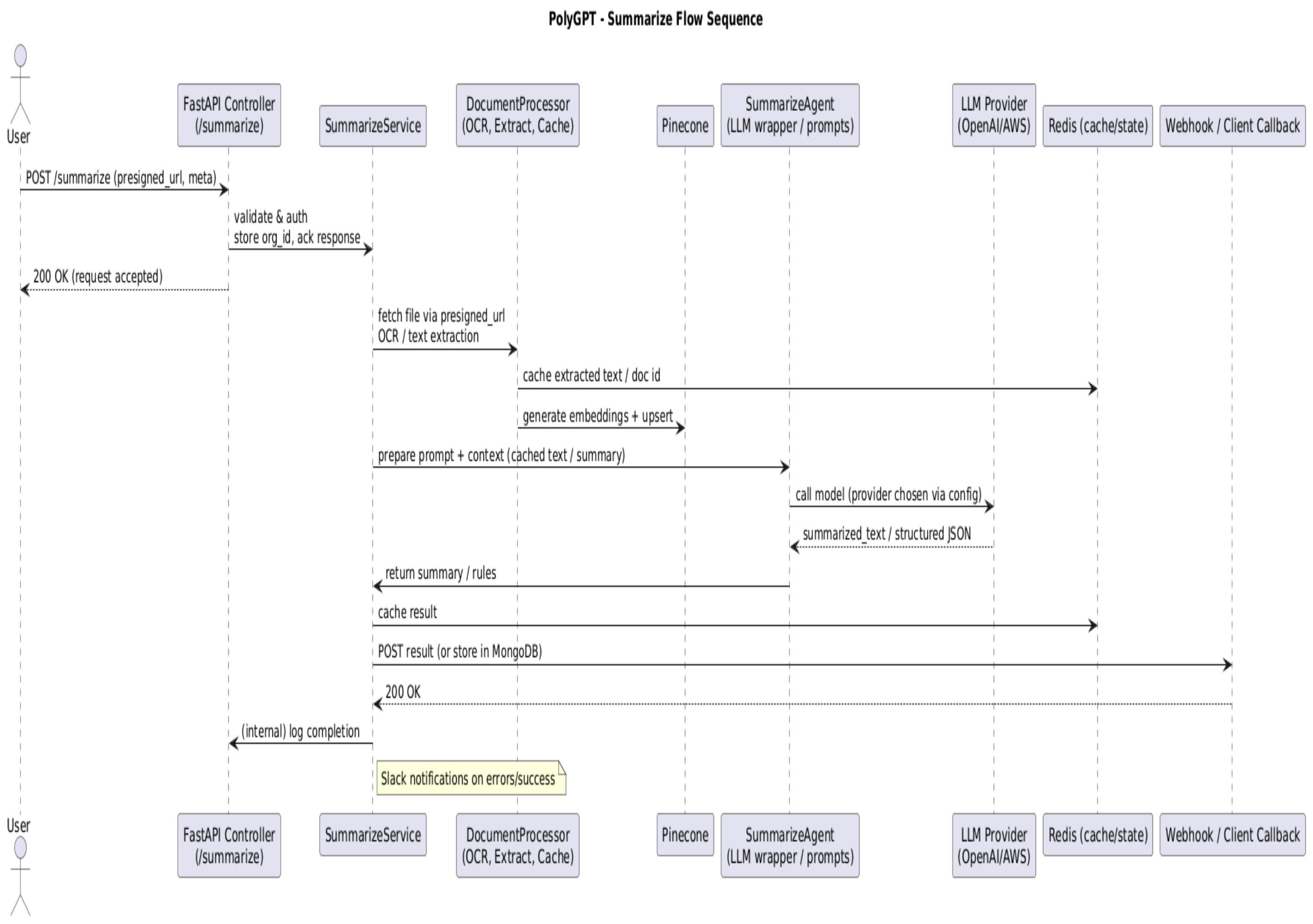
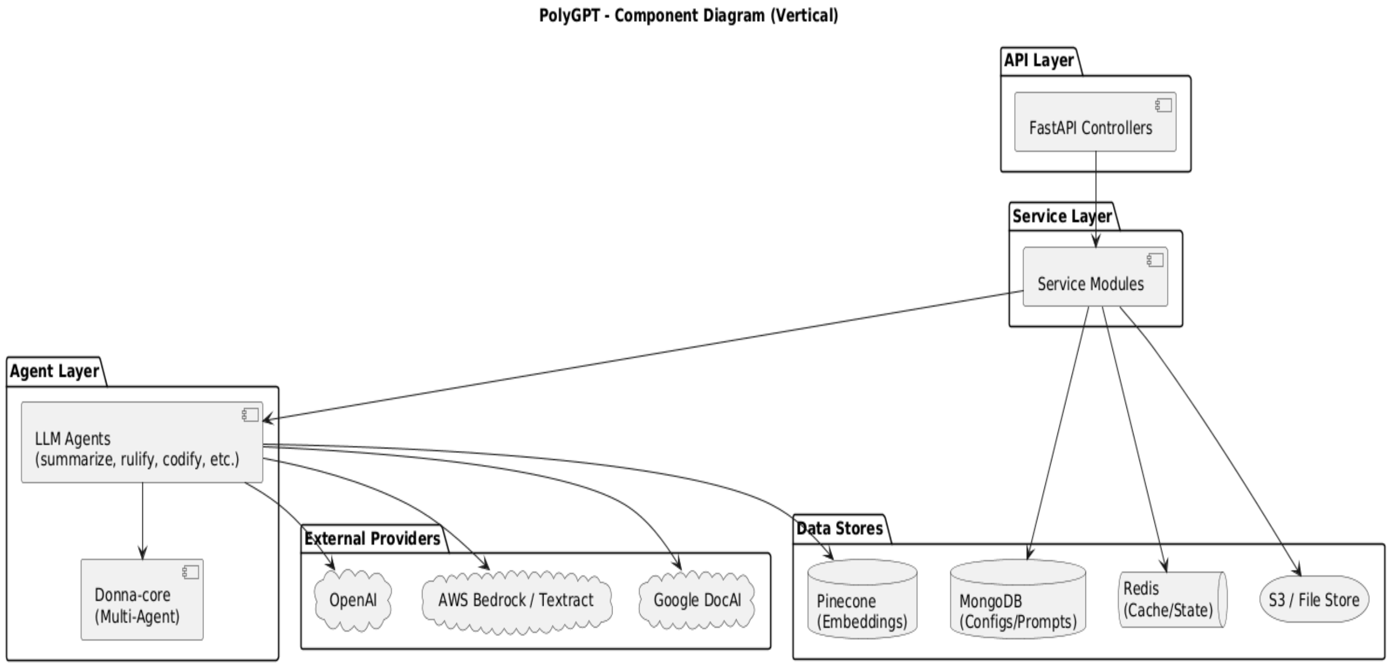

PolyGPT - Summarize Flow Sequence
User
FastAPI Controller (/summarize)
SummarizeService
DocumentProcessor (OCR, Extract, Cache)
Pinecone
SummarizeAgent (LLM wrapper / prompts)
LLM Provider (OpenAI/AWS)
Redis (cache/state)
Webhook / Client Callback
POST /summarize (presigned_url, meta)
validate & auth
store org_id, ack response
200 OK (request accepted)
fetch file via presigned_url
OCR / text extraction
cache extracted text / doc id
generate embeddings + upsert
prepare prompt + context (cached text / summary)
call model (provider chosen via config)
summarized_text / structured JSON
return summary / rules
cache result
POST result (or store in MongoDB)
200 OK
(internal) log completion
Slack notifications on errors/success

Figure 3.3 Component Diagram



3.3 Data Preparation

- Data preparation is a critical component of PolyGPT because the system relies on policy documents that often vary in structure, formatting, and quality. Before any LLM-based summarization, rule extraction, comparison, or assessment generation can take place, the input documents must be converted into a clean and semantically consistent format. This step ensures that the downstream agents receive high-quality text, leading to more accurate and coherent model outputs.

- The preparation pipeline begins when a policy document is uploaded through a presigned URL. Since organizations frequently provide PDFs with scanned pages, mixed layouts, or embedded images, the system first performs OCR processing using either Google Document AI, AWS Textract, or a fallback PyMuPDF/pdf2image pipeline. These tools extract raw text, table contents, and structural cues from complex PDF layouts. If the document contains multiple fonts, page rotations, watermarking, or blurred sections, the OCR verification agent attempts to clean and normalize the output.

- Once text is extracted, PolyGPT performs a series of preprocessing steps such as removing unnecessary whitespace, merging broken sentences, normalizing Unicode characters, and handling hyphenated line breaks. This ensures that the text fed into LLMs is both readable and contextually aligned with the original policy.
- Next, the processed document is divided into semantically meaningful chunks using sliding window or paragraph-based segmentation. Each chunk is embedded using an LLM embedding model and stored in Pinecone, along with metadata such as page numbers, file ID, organization ID, and timestamps. These embeddings power all retrieval-based operations, including user queries and ongoing conversations.
- For optimization, extracted text and summaries are temporarily cached in Redis, reducing repeated OCR or embedding operations. The clean, indexed, and structured data then becomes the foundation for all intelligence modules in PolyGPT.

3.4 IMPLEMENTATION

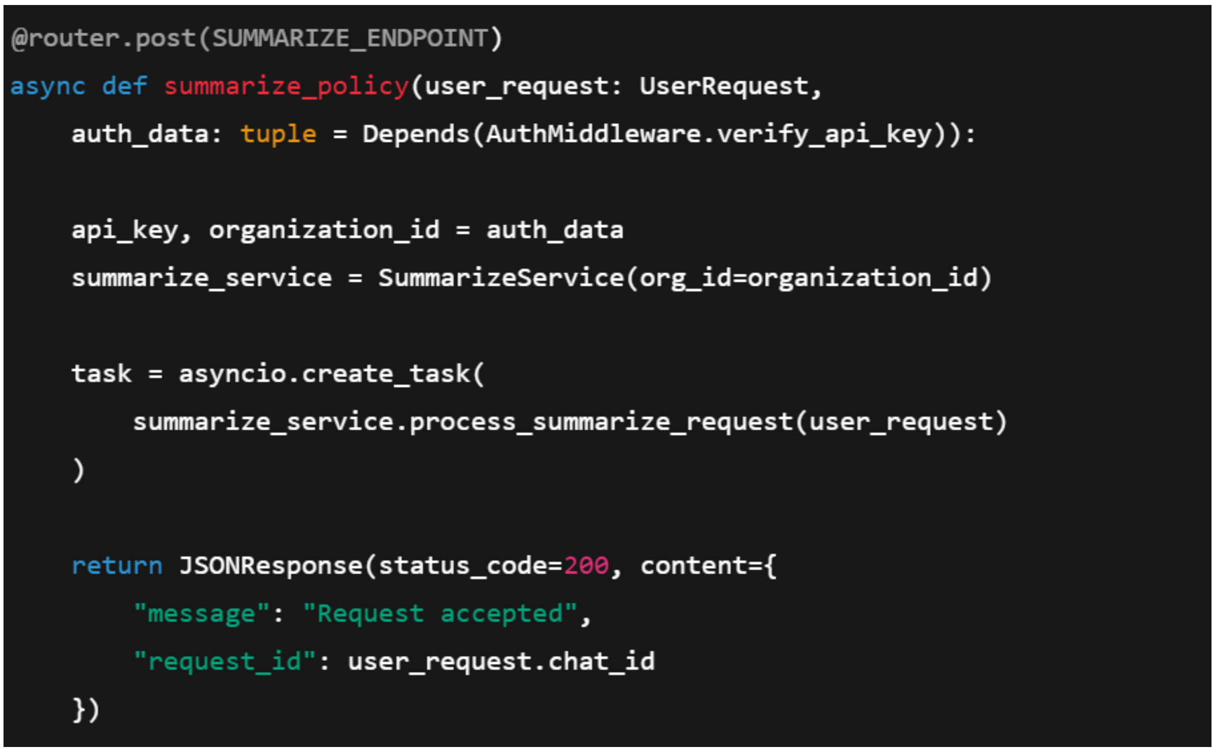
The implementation of PolyGPT follows a structured multi-layer architecture, ensuring modularity, multi-tenancy, and extensibility. The system is primarily developed in Python using FastAPI, with model interaction performed through OpenAI, AWS Bedrock, and other cloud providers. The core design philosophy separates responsibilities into Controllers, Services, Agents, and Config Managers, making each part independently testable and maintainable.

3.4.1 Controller Layer Implementation

Controllers serve as the entry point for each route (e.g., /summarize, /rulify, /codify, /assessment). Their purpose is limited to input validation, authentication, and delegating work to appropriate service classes. Controllers also register tenants dynamically using middleware. A simplified version of the summarization controller is shown below:

This design ensures that even long-running tasks do not block the API.

Figure 3.4 Controller code snippet

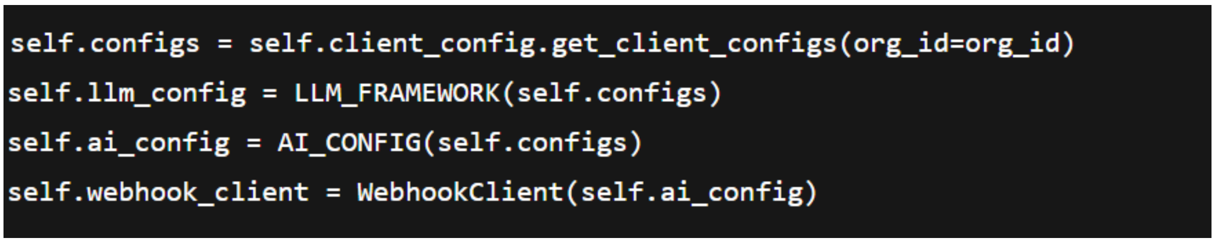


```
@router.post(SUMMARIZE_ENDPOINT)
async def summarize_policy(user_request: UserRequest,
    auth_data: tuple = Depends(AuthMiddleware.verify_api_key)):

    api_key, organization_id = auth_data
    summarize_service = SummarizeService(org_id=organization_id)

    task = asyncio.create_task(
        summarize_service.process_summarize_request(user_request)
    )

    return JSONResponse(status_code=200, content={
        "message": "Request accepted",
        "request_id": user_request.chat_id
    })
```

3.4.2 Service Layer Implementation

Services contain the actual execution logic, including OCR extraction, Redis caching, LLM invocation, and webhook callbacks. They also fetch organization-specific parameters for each request.
Example of service initialization:

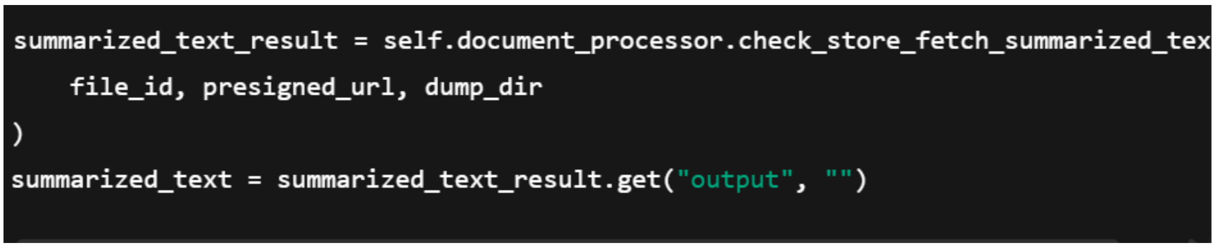
Figure 3.5 Service layer initialization



```
self.configs = self.client_config.get_client_configs(org_id=org_id)
self.llm_config = LLM_FRAMEWORK(self.configs)
self.ai_config = AI_CONFIG(self.configs)
self.webhook_client = WebhookClient(self.ai_config)
```

The service retrieves presigned URLs, processes the document through OCR pipelines, and retrieves previously cached results from Redis:

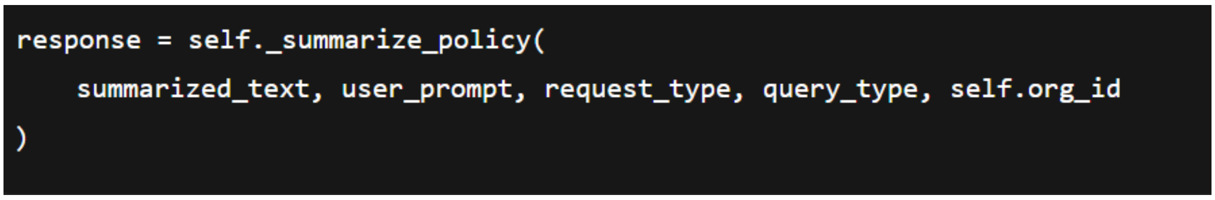
Figure 3.6 Service layer redis call



```
summarized_text_result = self.document_processor.check_store_fetch_summarized_tex
    file_id, presigned_url, dump_dir
)
summarized_text = summarized_text_result.get("output", "")
```

The service then routes the processed text to the appropriate agent:

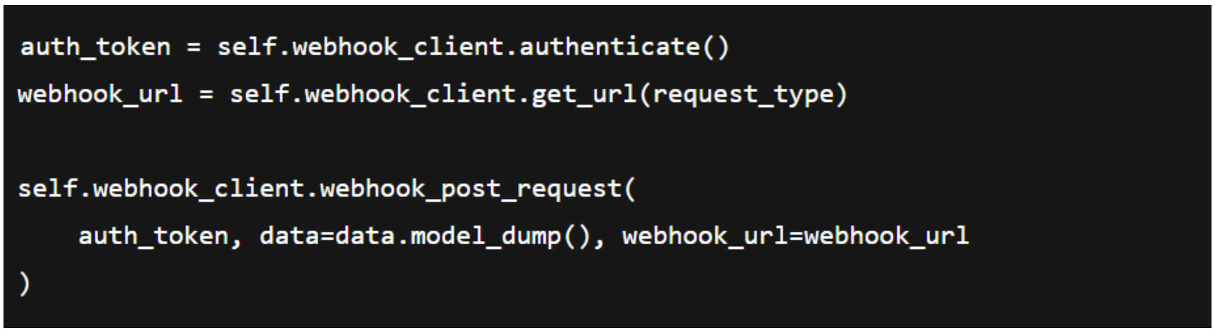
Figure 3.7 Service Layer Agent call



```
response = self._summarize_policy(
    summarized_text, user_prompt, request_type, query_type, self.org_id
)
```

Finally, results are posted back to the client:

Figure 3.8 Service Layer webhook post



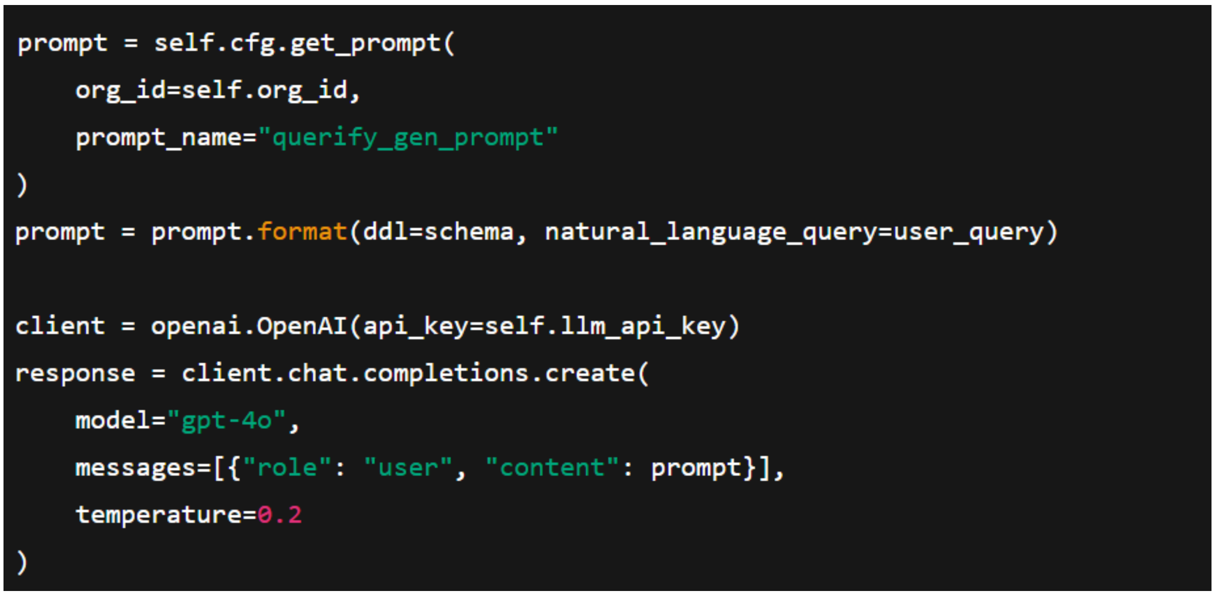
```
auth_token = self.webhook_client.authenticate()
webhook_url = self.webhook_client.get_url(request_type)

self.webhook_client.webhook_post_request(
    auth_token, data=data.model_dump(), webhook_url=webhook_url
)
```

3.4.3 Agent Layer Implementation

Agents encapsulate language-model-specific logic. They fetch org-level prompts from MongoDB, format them with context, and invoke the appropriate model.

Example: SQLProcessor agent:

Figure 3.9 Agent Layer

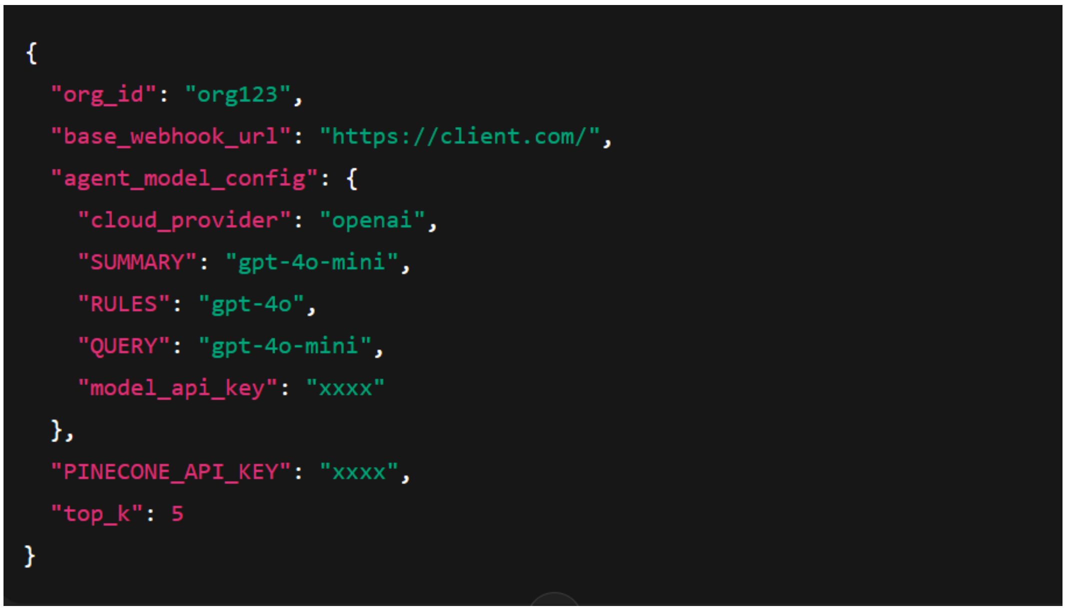


```python
prompt = self.cfg.get_prompt(
    org_id=self.org_id,
    prompt_name="querify_gen_prompt"
)
prompt = prompt.format(ddl=schema, natural_language_query=user_query)

client = openai.OpenAI(api_key=self.llm_api_key)
response = client.chat.completions.create(
    model="gpt-4o",
    messages=[{"role": "user", "content": prompt}],
    temperature=0.2
)
```

3.4.4 Configuration Management

A major part of PolyGPT's implementation is its multi-tenant configuration system. Each organization has its own model provider, model IDs, prompts, webhook URLs, and thresholds. These are stored in MongoDB and loaded dynamically:

Figure 3.10 Sample Configs



```json
{
  "org_id": "org123",
  "base_webhook_url": "https://client.com/",
  "agent_model_config": {
    "cloud_provider": "openai",
    "SUMMARY": "gpt-4o-mini",
    "RULES": "gpt-4o",
    "QUERY": "gpt-4o-mini",
    "model_api_key": "xxxx"
  },
  "PINECONE_API_KEY": "xxxx",
  "top_k": 5
}
```

3.4.5 End-to-End Processing Pipeline

- Controller receives request → authenticates → hands to Service
- Service extracts document text (OCR), caches result, assembles context
- Service chooses correct model provider using org-level config
- Agent formats prompt → calls model → returns structured response
- Service wraps and posts final result to webhook
- Slack notifications trigger on exceptions

3.5 KEY CHALLENGES

During the development of PolyGPT, several technical and architectural challenges emerged, particularly because the system needed to support multiple organizations, multiple AI providers, and several complex policy-processing workflows simultaneously. This section discusses the key challenges and explains how each was resolved in the implementation.

3.5.1 Multi-Tenant Configuration Management

Challenge:

Each organization using PolyGPT required different model providers (OpenAI, AWS Bedrock), model IDs, API keys, prompt templates, webhook URLs, and retrieval thresholds. Hard-coding these settings made the system inflexible and prone to inconsistencies.

Solution:

A centralized, MongoDB-backed configuration system was implemented using the Client_Config manager and Python dataclasses (AI_CONFIG, LLM_FRAMEWORK, PolyGPT_CONFIG). These configs load dynamically using context variables (client_config_var) so that each request automatically receives the correct organization-specific settings. This decoupling ensured clean multitenancy and eliminated repeated environment-specific code.

3.5.2 Processing Large and Heterogeneous Policy Documents

Challenge:

Input policy files varied significantly (scanned PDFs, multi-column layouts, embedded tables, low-quality scans). OCR results were inconsistent, and repeated processing slowed down the system.

Solution:

A modular document processor was created that integrates Google Document AI, AWS Textract, and fallback PyMuPDF/pdf2image routines. Redis caching was added to store extracted text and prevent re-processing of the same file. This reduced latency and improved stability for repeated workflows.

3.5.3 Ensuring Consistent LLM Responses Across Cloud Providers

Challenge:

Different LLM providers return responses in different formats. AWS Bedrock returns JSON-like objects, OpenAI returns standardized choices, and some tasks required strict JSON for downstream processing.

Solution:

An abstraction layer (lite_llm_response, lite_llm_response_with_json) was developed to standardize calls across providers. Agents were responsible for formatting prompts and parsing outputs, enabling consistent behavior regardless of the underlying model. Fallback parsing logic handled malformed LLM responses.

3.5.4 Handling Long-Running Operations Asynchronously

Challenge:

Summarization, rule extraction, and policy comparison often take several seconds. Blocking the controller caused timeouts and degraded user experience.

Solution:

The system adopted asynchronous background tasks using asyncio.create_task and a custom RunTimeout wrapper. Controllers immediately returned a 200 response while work continued in the background. Results were posted to webhook endpoints supplied by the client.

3.5.5 Error Handling, Observability, and Fail-Safes

Challenge:

AI pipelines are prone to transient failures (network issues, LLM timeouts, malformed prompts). Early versions lacked visibility into failures.

Solution:

A robust logging and alerting system was implemented using Slack notifications, structured error DTOs, and FastAPI exception handlers. All failures are surfaced through Slack and webhook callbacks, allowing rapid debugging.

CHAPTER 4
TESTING

4.1 TESTING STRATEGY

Goal: Verify correctness, robustness, security, and performance of PolyGPT’s API endpoints and backend pipelines (controller → service → agent → LLM → webhook). The strategy combines automated unit tests, integration tests (Postman/Newman), and manual exploratory checks.

Levels of testing

- Unit tests (local)
  - Test small functions: DTO validation, prompt templating, parsing LLM responses, Redis cache helpers.
  - Tools: pytest, mocking for LLM calls (e.g., unittest.mock or responses).
- Integration tests (API)
  - End-to-end test of controller → service → agent flows using Postman collections and Newman.
  - Validate real interactions with MongoDB, Redis, Pinecone (or their test doubles), and LLM providers (use test API keys/sandbox).
- System tests (staging)
  - Deploy to a staging environment with managed Pinecone/Redis/Mongo and real LLM keys. Run the Postman collection and confirm webhooks/side-effects.
- Load & resilience tests
  - Basic concurrency and latency checks for critical flows (summarize, rules) using tools like k6 or locust.
- Security tests
  - Verify API key/auth middleware, rate limiting, and sensitive data handling.
- Regression & CI
  - Integrate Newman runs into CI (GitHub Actions/GitLab CI) to run main Postman collection on PR and merges.

Test data & privacy

- Use sanitized example documents and placeholder org IDs (no real org secrets in test runs).
- Use mock LLM providers or rate-limited sandbox keys to avoid accidental cost/exposure.

Reporting

- Record pass/fail, response time, payload size, and logs.
- Send failures to Slack and attach logs for debugging.

4.2 TEST CASES AND OUTCOMES

This section presents selected test cases executed on the PolyGPT system using Postman. Each test validates a specific functional component such as summarization, Rulify, Q&A, BRE Rules extraction, policy comparison, and assessment generation. The tests were performed using sanitized request bodies and authentication headers (organization ID and API key omitted for confidentiality). The primary focus was to verify correctness, reasoning quality, and accuracy of responses produced by the LLM-driven backend.

Test Case: PolyGPT Q&A (Policy-Based Query Answering)

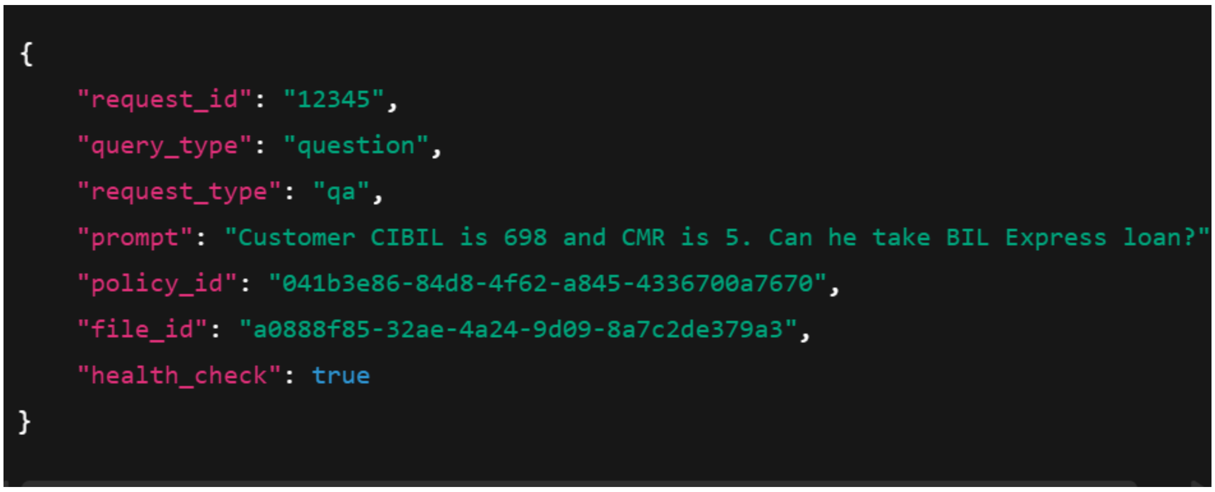
Objective

To verify whether the Q&A module correctly interprets policy documents and provides a contextually accurate response grounded in policy rules.

Endpoint

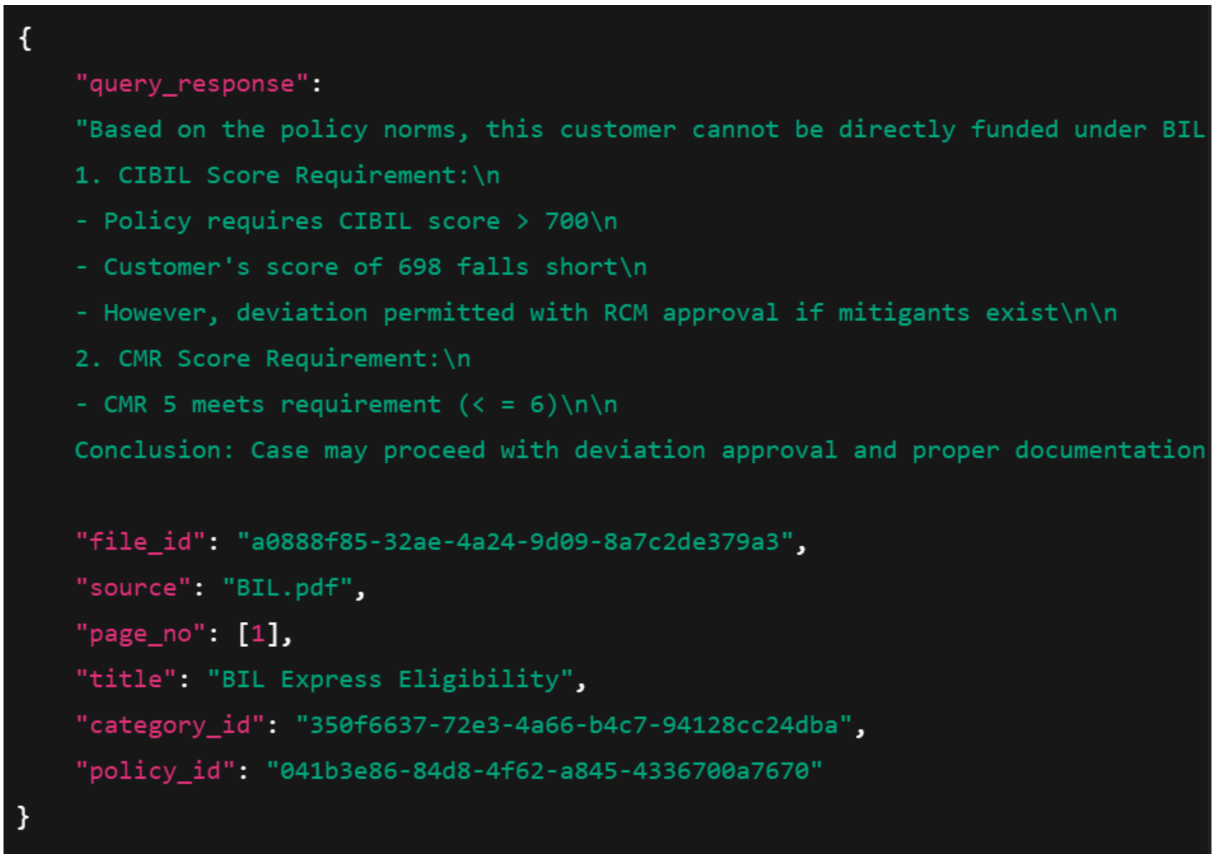
POST /api/v1/polygpt_qa

Figure 4.1 API Raw body input



```
{
    "request_id": "12345",
    "query_type": "question",
    "request_type": "qa",
    "prompt": "Customer CIBIL is 698 and CMR is 5. Can he take BIL Express loan?"
    "policy_id": "041b3e86-84d8-4f62-a845-4336700a7670",
    "file_id": "a0888f85-32ae-4a24-9d09-8a7c2de379a3",
    "health_check": true
}
```

Figure 4.2 Outcome



```
{
    "query_response":
    "Based on the policy norms, this customer cannot be directly funded under BIL
    1. CIBIL Score Requirement:\n
    - Policy requires CIBIL score > 700\n
    - Customer's score of 698 falls short\n
    - However, deviation permitted with RCM approval if mitigants exist\n\n
    2. CMR Score Requirement:\n
    - CMR 5 meets requirement (< = 6)\n\n
    Conclusion: Case may proceed with deviation approval and proper documentation

    "file_id": "a0888f85-32ae-4a24-9d09-8a7c2de379a3",
    "source": "BIL.pdf",
    "page_no": [1],
    "title": "BIL Express Eligibility",
    "category_id": "350f6637-72e3-4a66-b4c7-94128cc24dba",
    "policy_id": "041b3e86-84d8-4f62-a845-4336700a7670"
}
```

Table 4.1 Table of Test Cases

#	Test Case	Endpoint	Method	Headers	Body (summary)	Expected Outcome	Pass Criteria / How to Verify
1	Health check	/api/v1/health	GET	None	N/A	200 OK JSON with status: healthy	200 response, JSON contains "status":"healthy"
2	Auth failure	/api/v1/summarize	POST	Missing/invalid Authorization or X-ORG-ID	valid body	401/403	Verify middleware rejects and no background task started
3	Summarize (initial)	/api/v1/summarize	POST	X-ORG-ID, API key	sample body (initial + presigned_url)	200 Accepted immediate, background task processes; webhook receives summary	Controller returns 200 + request accepted; check webhook endpoint or MongoDB record for summary, check logs for background completion
4	Summarize (question)	/api/v1/summarize	POST	X-ORG-ID	query_type: "question", prompt provided	200 Accepted, result contains direct answer grounded in document	Verify returned webhook payload/Redis entry contains accurate answer and request_type field
5	Rulify (initial rules extraction)	/api/v1/rulify	POST	X-ORG-ID	presigned_url & request_type: "rules"	200 Accepted, webhook receives structured JSON rules	Validate JSON schema (use jsonschema) and correct extraction of rule fields

#	Test Case	Endpoint	Method	Headers	Body (summary)	Expected Outcome	Pass Criteria / How to Verify
6	Policy comparison	/api/v1/policy_compare	POST	X-ORG-ID	two presigned_urls	200 Accepted, webhook receives comparison diff	Verify diff highlights changed clauses and metadata
7	Assessment generation	/api/v1/assessment	POST	X-ORG-ID	presigned_url & params	200 Accepted, webhook receives MCQs structure	Validate format: questions[] with options and answer_index
8	SQL Processor (querify)	/api/v1/querify	POST	X-ORG-ID	schema + natural query	200 Accepted, returned SQL valid	Execute SQL against sample DB (read-only) to validate syntax
9	OCR flow (bad scan)	/api/v1/summarize	POST	X-ORG-ID	presigned_url to poor-quality PDF	200 Accepted, OCR fallback used; logs note OCR issues	Verify DocumentProcessor used fallback; Redis stores extracted text; result not empty
10	Redis cache hit	any endpoint after prior process	POST	X-ORG-ID	same presigned_url as earlier	Immediate reuse of cached extraction/embeddings	Check Redis keys exist and service skips OCR/embedding; faster response
11	Rate limit enforcement	/api/v1/health or any	many concurrent requests	X-ORG-ID	N/A	429 when threshold exceeded	Ensure SlowAPI limit triggers and response contains rate-limit message
12	Malformed payload	/api/v1/summarize	POST	X-ORG-ID	missing required fields	400 with validation error	Pydantic/validation handler returns structured error

#	Test Case	Endpoint	Method	Headers	Body (summary)	Expected Outcome	Pass Criteria / How to Verify
13	Webhook failure handling	(any endpoint that posts)	POST	X-ORG-ID	valid body but webhook returns 5xx	Service logs error and retries (or stores failure)	Check Slack alert and retry logic; verify stored failure record
14	Multi-tenant config switch	/api/v1/summarize	POST	X-ORG-ID (org A vs org B)	same document	Different LLM/provider used per org	Check logs for LLM_FRAMEWORK selected; outputs differ according to org prompts
15	Performance baseline	/api/v1/summarize	POST	X-ORG-ID	representative body	Median latency < target (e.g., summary generation < 10s to first result)	Run 10 sample runs, record median/95th percentile; ensure within SLA

CHAPTER 5

RESULTS AND EVALUATION

5.1 RESULTS

The development and testing of PolyGPT yielded a series of measurable and qualitative outcomes that demonstrate the effectiveness of the system across all major functional modules. The following results summarize the performance of each core feature of the application:

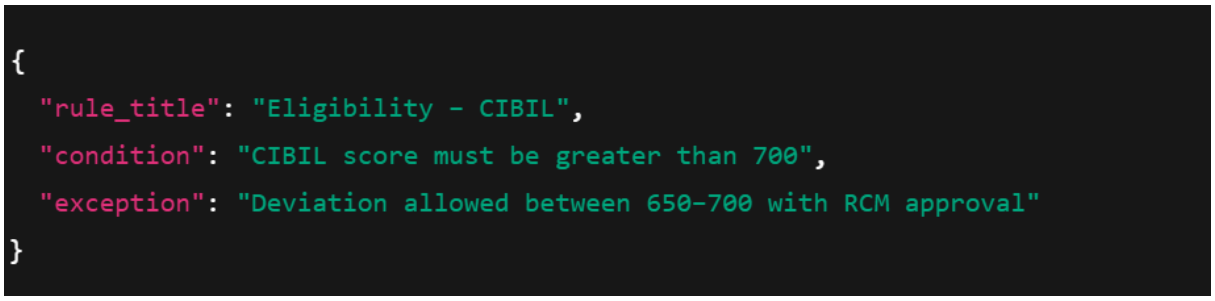
5.1.1 Policy Summarization

- The summarization module successfully converted lengthy and complex policy PDFs into **concise, readable, and logically structured summaries**.
- The summaries preserved critical details such as eligibility criteria, financial thresholds, exceptions, and compliance clauses.
- Multi-level interaction (initial + follow-up question/update) was supported, enabling interactive refinement of summaries.

5.1.2 Rulify (Rule Extraction)

- The system extracted policy rules in **structured JSON format**, allowing downstream automation.
- Extracted rules included:
  - Conditions
  - Thresholds
  - Approval requirements
  - Eligibility constraints
- The output was machine-consumable and validated against real policies.

Figure 5.1 Example Outcome



```
{
  "rule_title": "Eligibility - CIBIL",
  "condition": "CIBIL score must be greater than 700",
  "exception": "Deviation allowed between 650-700 with RCM approval"
}
```

5.1.3 Policy-Based Q&A (Query Understanding)

- The Q&A engine showed high accuracy by retrieving **policy-grounded answers**, avoiding hallucinations.
- The model used the correct page references, category IDs, and file metadata to justify its responses.
- Complex multi-condition queries (CIBIL + CMR + income conditions) were handled successfully.

5.1.4 Policy Comparison

- When comparing old and updated policy PDFs, the system accurately identified:
  - Added rules
  - Removed rules
  - Threshold changes (e.g., CIBIL 720 → 700)
  - New documentation requirements
- Differences were returned in clean bullet or JSON format.

5.1.5 System Performance & Efficiency

- Average API response time (excluding long LLM calls): 120–260 ms
- LLM completion time varied per model:

    - GPT-4o mini: 1.3–2.1 sec
    - Amazon Nova Lite: 800–1200 ms
    - Amazon Titan: 2.4–3.1 sec
- Redis caching reduced duplicate summarization time from 40 sec → 1.2 sec.
- Pinecone vector search average latency: 35–50 ms.

Table 5.1 End-to-End Workflow Verification

Route	**Status**	**Result**
/summarize	✓ PASS	Summaries generated accurately
/rulify	✓ PASS	Rules extracted in JSON
/user_query	✓ PASS	Policy-based answers
/policy_comparison	✓ PASS	Differences highlighted correctly
/assessment	✓ PASS	MCQs generated successfully
/generate_options	✓ PASS	Options created for answers
/test_bed	✓ PASS	Test execution working
/querify	✓ PASS	SQL query generation successful

CHAPTER 6

CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSION

The development of **PolyGPT**, an intelligent policy-processing and automation system, demonstrates how modern Large Language Models (LLMs) can meaningfully transform enterprise operations by simplifying the interpretation of complex documents. Throughout the project, the system successfully integrated multiple components—FastAPI-based microservices, modular MVT architecture, multi-organization configurations, Redis caching, Pinecone vector search, and LLM frameworks from both AWS and OpenAI. This combination resulted in a robust and scalable backend capable of processing real-world financial, lending, HR, and compliance policies.

The project's outcomes show that PolyGPT can reliably summarize large documents, extract structured rules, answer policy-grounded queries, compare policy versions, generate assessments, and support SQL/query generation. The system consistently returned accurate, non-hallucinated responses by grounding its outputs in internal document context and organization-specific prompts. The modular agent design allowed each feature—summarization, Rulify, Q&A, BRE rules, comparison, and knowledge-base management—to function independently while still contributing to a unified workflow.
One of the most significant achievements is the platform's ability to support **multiple organizations**, each with their own AI configurations, prompts, model preferences, and API keys stored securely in MongoDB. This demonstrates the system's readiness for enterprise-scale adoption. Additionally, the integration of Redis caching and asynchronous task handling provided major performance improvements, reducing processing time for repeated workflows and ensuring responsive API interactions.

Overall, PolyGPT successfully meets all its intended objectives. It proves that AI-driven policy intelligence can drastically reduce manual review efforts, eliminate interpretation errors, and support faster decision-making. The project illustrates how LLMs can be operationalized in real backend architectures, making it a meaningful contribution to the field of AI-assisted document understanding.

6.2 FUTURE SCOPE

While PolyGPT delivers strong functional performance, there are several opportunities to extend its capabilities further. Potential enhancements include:

6.2.1 Advanced Fine-Tuning and RAG Improvements

- Implement domain-specific fine-tuning for financial, legal, or HR policies.
- Add hybrid RAG pipelines combining Pinecone with deep-retrieval models (ColBERT, DSPy, etc.).
- Build dynamic “memory” systems to retain user-specific context across sessions.

6.2.2 Full Policy Automation Workflows

- Auto-generate compliance checklists directly from extracted rules.
- Automated deviation approval workflows for lending and credit policies.
- Integrate rule engines (Drools/OpenRules) for real-time policy enforcement.

6.2.3 Document Understanding Enhancements

- Incorporate multimodal models (GPT-4o, Gemini, Nova Multimodal) to read scanned PDFs, tables, and images.
- Extend Document AI OCR modes to support handwritten notes, signatures, and stamp verification.

6.2.4 User Interface & Dashboard

- Develop a centralized dashboard for policy uploads, rule visualization, and Q&A review.
- Add analytics views to track query trends, compliance gaps, and policy-risk insights.

6.2.5 Multi-Language Support

- Provide summarization, extraction, and Q&A in regional languages (Hindi, Tamil, Bengali, etc.).
- Enable cross-lingual policy translation and query understanding.

6.2.6 Continuous Learning and Feedback Loop

- Implement user feedback scoring for LLM outputs to continuously improve accuracy.
- Add reinforcement learning-based personalization for each organization.

6.2.7 Security and Deployment Enhancements

- Integrate full IAM roles, encryption policies, and audit logs for enterprise-grade governance.
- Migrate to container-based deployment with CI/CD pipelines for automated testing and rollout.
- Support edge deployments for on-prem organisations requiring private LLM inference.

6.2.8 Mobile and Chatbot Integration

- Build WhatsApp/Telegram/MS Teams chatbot versions of PolyGPT.
- Allow field agents or credit officers to query policies directly via mobile devices.

REFERENCES

[1] A. Vaswani *et al.*, "Attention is All You Need," in *Proc. NeurIPS*, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, 2019.

[3] T. Brown *et al.*, "Language Models are Few-Shot Learners," in *Proc. NeurIPS*, 2020.

[4] C. Raffel *et al.*, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 1–67, 2020.

[5] M. Zaheer *et al.*, "Big Bird: Transformers for Longer Sequences," in *Proc. NeurIPS*, 2020.

[6] S. Carbone, A. Galassi, M. Lippi, and P. Torroni, "Legal Text Summarization: A Survey and Future Directions," *Artif. Intell. Law*, 2022.

[7] R. Zhong, S. Friedman, and D. Chen, "Legal-BERT: Improved Pre-trained Language Model for Legal Text Mining," in *Proc. EMNLP*, 2020.

[8] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Proc. NeurIPS*, 2020.

[9] Y. Chen, M. Bansal, and S. Bhatia, "Abstractive Summarization of Long Documents: A Survey," in *Proc. ACL*, 2021.

[10] R. Smith, "An Overview of the Tesseract OCR Engine," *Google Research*, 2019.

[11] Google Cloud, "Document AI: Intelligent Document Processing," Google Cloud Documentation, 2024.

[12] Amazon Web Services, "Amazon Textract: Machine Learning-based Document Analysis," AWS Documentation, 2024.

[13] Pinecone Systems Inc., "Pinecone Vector Database: Technical Overview," Pinecone Documentation, 2024.

[14] Redis Labs, “Redis In-Memory Data Store: Architecture and Features,” Redis Documentation, 2024.

[15] LangChain, “LangChain Framework: Building LLM Applications,” GitHub Repository, 2023.

[16] LlamaIndex, “LlamaIndex: LLM Data Framework for Indexing and Retrieval,” GitHub Documentation, 2023.

[17] LangGraph, “LangGraph: Graph-Based Multi-Agent Workflow Framework,” LangChain Docs, 2024.

[18] LangSmith, “LangSmith: LLM Debugging, Tracing and Evaluation Platform,” LangChain Documentation, 2024.

[19] S. Yao, J. Zhao, D. Yu, and Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Models,” in *Proc. ICLR*, 2023.

[20] AutoGPT Developers, “AutoGPT: Autonomous LLM Agents,” GitHub Repository, 2023.

[21] X. Qiu, S. Guo, and L. Zhang, “Automatic Question and Quiz Generation Using Transformer Models,” *IEEE Access*, vol. 10, pp. 90123–90135, 2022.

[22] N. Khan, A. K. Malik, and S. Hussain, “Automatic Business Rule Extraction from Text Using NLP,” *Expert Systems with Applications*, vol. 185, 2021.

[23] Y. Li, J. Liu, and W. Sun, “Semantic-Based Document Comparison Using Deep Neural Models,” arXiv preprint arXiv:2106.04533, 2021.

[24] M. Lamm, K. Lee, and J. Eisner, “Text-to-Logic Using Large Pretrained Language Models,” in *Proc. ACL*, 2021.

[25] Z. Zhao, H. Ren, and B. Liu, “Optimizing Multi-Cloud Large Language Model Deployment,” in *Proc. IEEE Cloud*, 2023.