Project: Resilient LLM Gateway API Platform

Subject: CS691

Computer Science

Team 2

Agenda

- Objective: Evaluate our progress against deadlines, sync on development milestones, and clear any existing roadblocks.
- Timeline Management: Establish a firm schedule for submissions and sync all critical dates to the shared team calendar.
- Presentation Development: Select core presentation topics and define a workflow for continuous iterative updates.
- Client & Persona Alignment: Review the project through the lens of user personas to ensure all client requirements and expectations are met.
- Risk Mitigation: Identify potential threats to the project’s success and discuss preventative measures.
- Team Collaboration: Address the necessity of active engagement, consistent attendance, and unified teamwork for a successful delivery.

Team Members Roles & Responsibilities

Diya Farakte
Business Analyst & Developer

Nisarga Vishwamanjuswamy
Project Manager & Developer

Prachi Budhrani
Data & Observability Engineer & Developer

Team Members Roles & Responsibilities

Rohan Brahmbhatt
Performance Engineer &
Developer

Pramod Kumar Reddy
Parvath Reddy
LLM Integration Engineer
& Developer

PROBLEM STATEMENT

Problem Statement

Many organizations and developers rely on Large Language Model providers such as OpenAI and Gemini to build AI-powered applications. However, most applications are directly integrated with a single provider, which creates several challenges, including vendor lock-in, service downtime, inconsistent APIs, limited fault tolerance, and lack of centralized monitoring and cost control. When a provider experiences failures, rate limits, or performance issues, dependent applications are immediately affected, leading to poor user experience and increased maintenance effort. Additionally, managing multiple API keys, tracking token usage, and enforcing security policies across teams becomes difficult. These issues make it hard to deploy LLM-based systems reliably and efficiently at scale, highlighting the need for a unified, resilient, and centrally managed gateway platform.

PROJECT DESCRIPTION

Project Description

- The Resilient LLM Gateway is a production-grade API platform that provides a unified interface for accessing multiple Large Language Model (LLM) providers such as OpenAI and Gemini. Instead of connecting applications directly to individual providers, the gateway exposes a single REST API for chat, summarization, classification, and embeddings.
- The system improves reliability by implementing automatic retries and fallback mechanisms when a provider fails. It also enables centralized governance through API keys, rate limiting, usage monitoring, and cost tracking. Additionally, the platform provides observability features such as structured logging and performance metrics.
- This project focuses on building scalable and secure AI infrastructure that allows organizations to safely deploy LLM-powered applications in production environments.

Personas: John

Software Developer, Builds AI-powered apps, Needs simple API, Wants reliability

Personas: Anthony

Data Scientist, Experiments with models, Compares OpenAI vs Gemini, Needs cost tracking

Personas: Kate

IT Administrator, Manages access, Controls budgets, Monitors usage

TECHNOLOGIES

Tool	Purpose
Python	Main programming language
FastAPI	REST API framework
PostgreSQL	Store usage, cost, and audit logs
Redis	Rate limiting and caching
Docker	Containerized deployment
GitHub	Version control and collaboration
OpenTelemetry	Monitoring and tracing

ALGORITHMS

1) Provider Routing Algorithm

- Chooses which provider to use: **OpenAI or Gemini**
- Strategy: default provider OR rules (latency/cost/availability)

2) Retry Algorithm

- If request fails (timeout/429/5xx), retry **N times**
- Uses delay between retries (basic backoff)

3) Fallback Algorithm

- If primary provider fails after retries → switch to **secondary provider**
- Example: OpenAI fails → automatically send to Gemini

ALGORITHMS

4) Rate Limiting Algorithm

- Limits requests per API key (e.g., 60 req/min)
- Implemented using Redis counter per time window

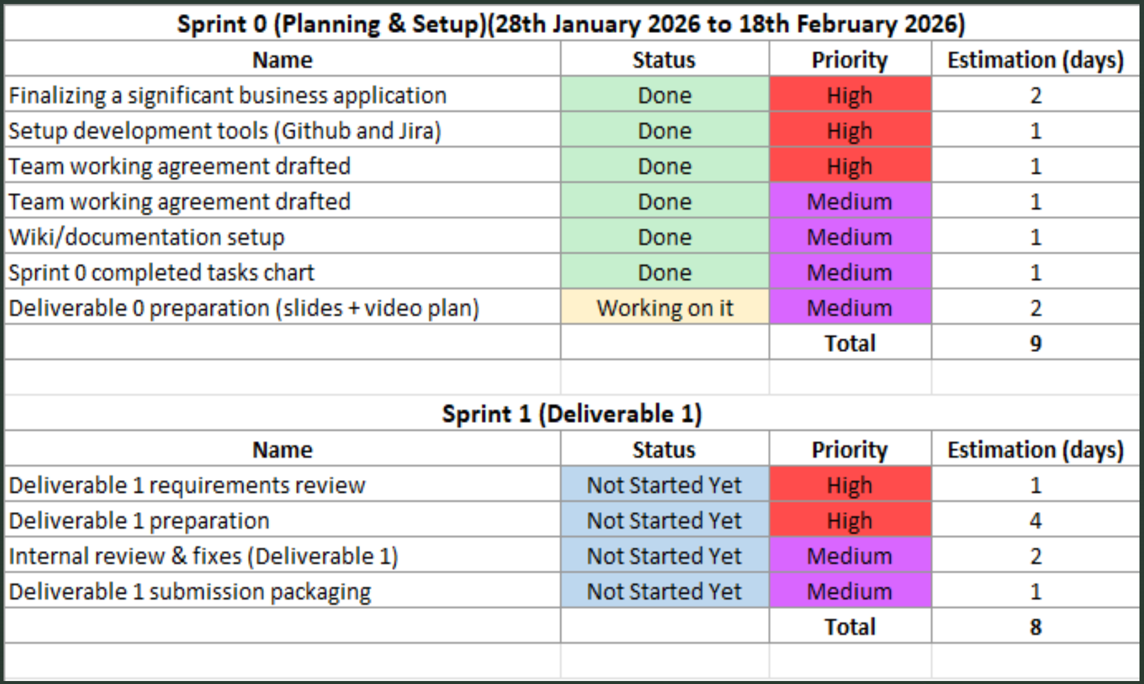
5) Caching Algorithm

- Store repeated results (especially embeddings) in Redis
- If same input appears again → return cached output to save cost/latency

6) Usage & Cost Tracking Algorithm

- For each request: record tokens in/out, model, provider, cost estimate
- Store in PostgreSQL for analytics and quotas

Project Schedule



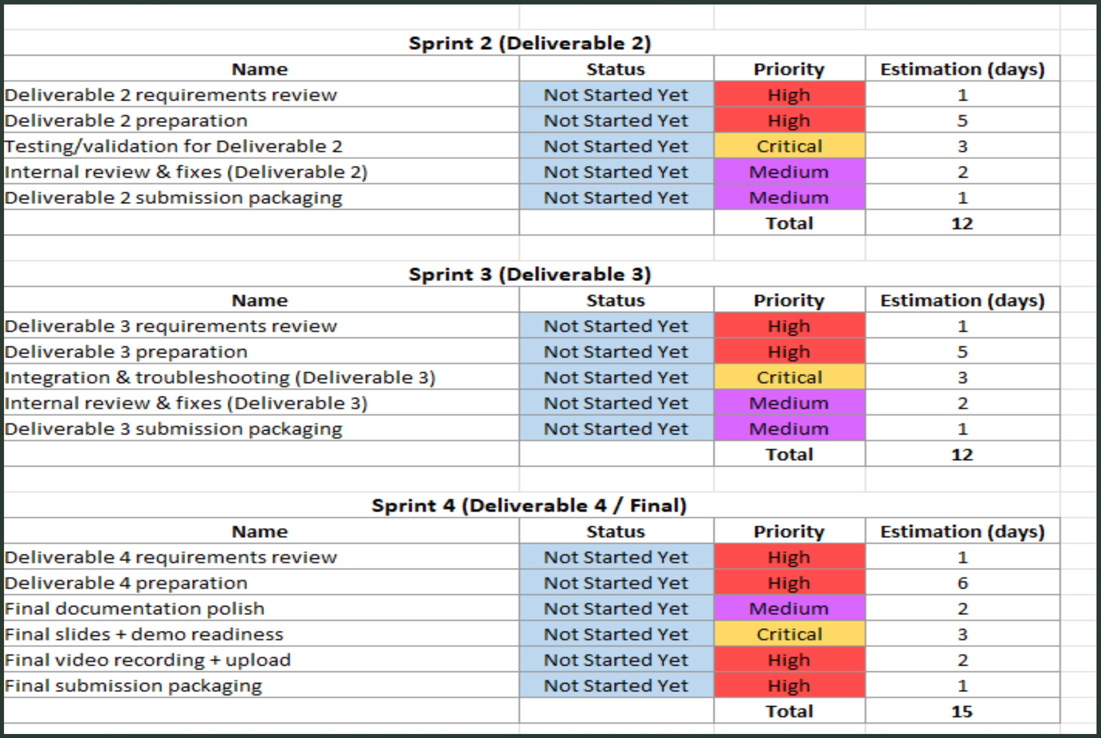
Sprint 0 (Planning & Setup)(28th January 2026 to 18th February 2026)

Name	Status	Priority	Estimation (days)
Finalizing a significant business application	Done	High	2
Setup development tools (Github and Jira)	Done	High	1
Team working agreement drafted	Done	High	1
Team working agreement drafted	Done	Medium	1
Wiki/documentation setup	Done	Medium	1
Sprint 0 completed tasks chart	Done	Medium	1
Deliverable 0 preparation (slides + video plan)	Working on it	Medium	2
		Total	**9**

Sprint 1 (Deliverable 1)

Name	Status	Priority	Estimation (days)
Deliverable 1 requirements review	Not Started Yet	High	1
Deliverable 1 preparation	Not Started Yet	High	4
Internal review & fixes (Deliverable 1)	Not Started Yet	Medium	2
Deliverable 1 submission packaging	Not Started Yet	Medium	1
		Total	**8**

Project Schedule



Sprint 2 (Deliverable 2)

Name	Status	Priority	Estimation (days)
Deliverable 2 requirements review	Not Started Yet	High	1
Deliverable 2 preparation	Not Started Yet	High	5
Testing/validation for Deliverable 2	Not Started Yet	Critical	3
Internal review & fixes (Deliverable 2)	Not Started Yet	Medium	2
Deliverable 2 submission packaging	Not Started Yet	Medium	1
		Total	**12**

Sprint 3 (Deliverable 3)

Name	Status	Priority	Estimation (days)
Deliverable 3 requirements review	Not Started Yet	High	1
Deliverable 3 preparation	Not Started Yet	High	5
Integration & troubleshooting (Deliverable 3)	Not Started Yet	Critical	3
Internal review & fixes (Deliverable 3)	Not Started Yet	Medium	2
Deliverable 3 submission packaging	Not Started Yet	Medium	1
		Total	**12**

Sprint 4 (Deliverable 4 / Final)

Name	Status	Priority	Estimation (days)
Deliverable 4 requirements review	Not Started Yet	High	1
Deliverable 4 preparation	Not Started Yet	High	6
Final documentation polish	Not Started Yet	Medium	2
Final slides + demo readiness	Not Started Yet	Critical	3
Final video recording + upload	Not Started Yet	High	2
Final submission packaging	Not Started Yet	High	1
		Total	**15**

TEAM WORKING AGREEMENT

Link:

Agreement - Google Docs

RETROSPECTIVE

What Went Well

- Clear project scope and objectives were defined early.
- Team roles were assigned effectively.
- Initial documentation and planning were completed on time.

What Could Be Improved

- Earlier coordination on task ownership would improve efficiency.
- Documentation formatting could be reviewed mid-sprint rather than at the end.
- Better version control discipline is needed.

Action Items for Next Sprint

- Define clearer milestone checkpoints.
- Implement stricter GitHub workflow practices.
- Schedule mid-sprint internal reviews.
- Improve time management and task tracking.

Lessons Learned

- Strong planning improves overall execution quality.
- Consistent communication prevents last-minute pressure.
- Equal participation strengthens project outcomes.

Wiki page Link

https://github.com/Resilient-LLM-Gateway-Team-2/llm-gateway/wiki

Thank you!