INTELLECTUAL RAG-SYSTEM FOR UNIVERSITIES: DEPLOYMENT PLAN

Students : Alekseeva Polina,
Zemskova Sofia

PROJECT PURPOSE AND OBJECTIVES

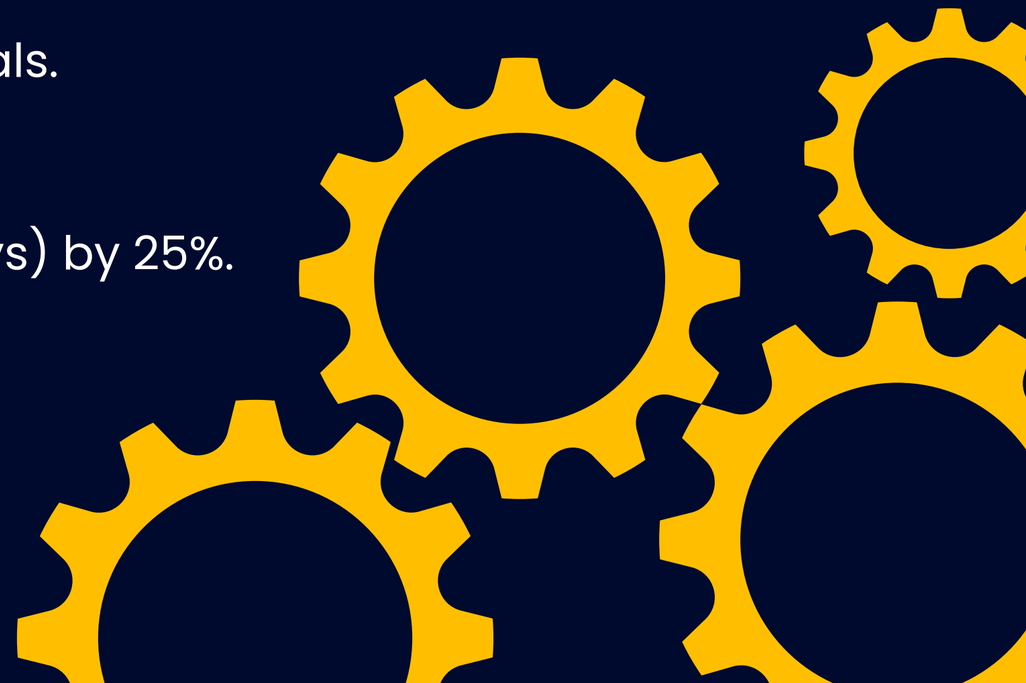
The Intellectual RAG-System for Universities aims to enhance student learning by providing 24/7 access to accurate, course-aligned information via a smart assistant.

Problem Addressed

Students often struggle to clarify lecture content in real time, while teachers are overwhelmed with repeated questions.

Key Objectives

- 90% answer accuracy using only uploaded materials.
- Reduce teacher query workload by 40%.
- Increase student satisfaction (measured by surveys) by 25%.



STAKEHOLDERS AND MINIMUM VIABLE PRODUCT

Primary Users

University students seeking immediate support.

Stakeholders

University administration, teachers, and IT departments.

Minimum Requirements

- Simple browser-based chatbot interface.
- Ability to upload and process PDF lecture files.
- Accurate answer generation using course materials only.

COMPLIANCE AND LEGAL CONSIDERATIONS

01

System must comply with **GDPR** and relevant university **data policies**.

Key measures:

02

Restrict access to sensitive materials.

Ensure anonymization where necessary.

Document data handling decisions.

03

Plan: Collaborate with university's legal/IT department to validate compliance.

HARDWARE AND SOFTWARE REQUIREMENTS

Deployment Infrastructure

Containerized Python 3.12.9 running a single Streamlit process (no separate web server).

Docker image built from python:3.12-slim, with Uvicorn under the hood for async request handling.

Orchestration-ready: can drop into Kubernetes (helm chart) with HPA (CPU target ~60%, memory ~70%) and liveness/readiness probes.

CI/CD via GitHub Actions: on push → build image → run pytest + flake8 → push to registry → helm upgrade

HARDWARE AND SOFTWARE REQUIREMENTS

Storage & Indexing

FAISS-CPU in-memory index (vector dim = 1536) for sub-millisecond nearest neighbor lookups.

Chunking pipeline: ~500-token window with 50-token overlap, implemented in pdf_loader.py, cached via @st.cache_data.

Memory-mapped vectors: vectors stored in a NumPy memmap on disk between restarts for fast reload (optional).

Ephemeral by default: index lives in st.session_state; drop-in persistence via faiss.write_index()/read_index().

HARDWARE AND SOFTWARE REQUIREMENTS

Model Hosting & Inference

OpenAI Embeddings. Endpoint: text-embedding-ada-002

Batching: up to 100 chunks/request for throughput (~30 ms per batch)

Rate-limit handling: tenacity-backed exponential backoff on 429/5xx

OpenAI Chat Completions. Model: gpt-4o-mini-2025-04-16 (or gpt-4)

Streaming: stream=True to render tokens as they arrive, cutting UI latency in half

Context window: up to 8 K tokens, truncating oldest chunks if needed

Local fallback. Transformers pipeline loading quantized gpt2-xl on CPU with 8-bit weights for “best-effort” replies under quota.

HARDWARE AND SOFTWARE REQUIREMENTS

Tech Stack & Observability

Languages & Frameworks Python 3.12.9, Asyncio event loop driving Streamlit; Streamlit 1.44.1 for UI, session-state chat API; PyPDF 5.4.0 for PDF→text extraction; NumPy & FAISS-CPU 1.10.0 for vector math

APIs & SDKs openai 1.71.0 (v1.0+ async interface) for both embeddings & chat; Hugging Face transformers 4.x for fallback generation

Logging & Metrics; structlog for structured JSON logs; Prometheus client for request latency histograms (embed vs chat); OpenTelemetry traces around network calls (OpenAI, FAISS search)

Secrets & Config st.secrets["OPENAI_API_KEY"] injected at runtime—no creds in code; config.toml for Streamlit theme and server settings

PERSONNEL AND CHANGE MANAGEMENT

01

Team Roles

Project Managers & Developers:

Polina Alekseeva, Sofia Zemskova

02

Change Management Plan

1. Create urgency (highlight student needs)
2. Form pilot group (early adopters)
3. Build vision (AI-enhanced education)
4. Remove barriers (training, support)
5. Celebrate early wins
6. Sustain momentum (feedback-driven updates)

TESTING AND INTEGRATION

Integration Target

Learning Management Systems (LMS) like Moodle.

Testing Types

Unit testing for backend functionality.

Integration testing for chatbot and PDF pipeline.

Validation testing to ensure model outputs align with course materials.

Performance testing under concurrent usage.

DATA SECURITY AND READINESS

Data Validation

Ensure uploaded PDFs are complete and well-structured.

Security Measures

Data encryption in storage and transit.

Role-based access control.

Regular security audits.

Backup Plan

Daily backups of both course materials and embeddings

POST-DEPLOYMENT MONITORING

Monitoring Tools

Prometheus (system metrics), Grafana (dashboards)

Performance KPIs

Response accuracy

Query volume

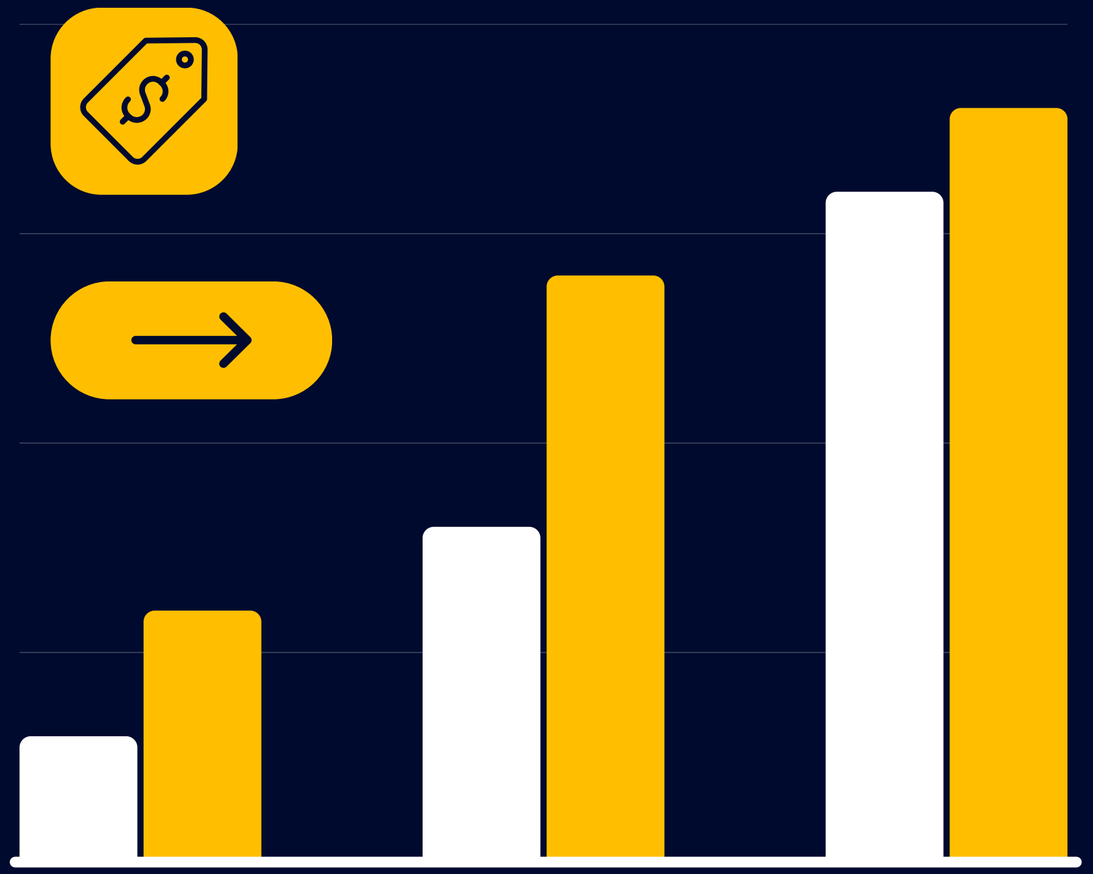
Average response time

Data Validation

Weekly model update based on feedback.

Continuous error analysis and retraining.

Plan to expand to other faculties and universities.



DEPLOYMENT TIMELINE

Phase	Start	End
Development & Measurement	Mar 10, 2025	Apr 14, 2025
Testing & Analysis	Apr 14, 2025	Apr 28, 2025
Improvement & Control	Apr 28, 2025	May 12, 2025
Final Report & Close-out	May 12, 2025	May 20, 2025

- Weekly syncs to track progress
- User testing during final two weeks

RISK ASSESSMENT

Risks

Inaccurate responses due to incomplete PDFs.

Resistance from traditional faculty.

Technical issues with file parsing or embedding.

Constraints

Limited student-project budget.

Academic calendar deadlines.

Assumptions

Students will actively use the system.

Teachers will provide consistent materials.

University infrastructure is sufficient for deployment.

SUMMARY AND NEXT STEPS

System ready for pilot deployment.

Legal and infrastructure requirements considered.

Integration and testing planned with clear timeline.

Post-launch monitoring in place.

Next Steps

Finalize system deployment

Conduct live pilot

Gather and act on feedback

Scale up to broader university use

THANK YOU