Generative AI Final Year Project Ideas PDF

A good generative AI final year project wraps an LLM in a real system: retrieval over your own documents, tools or agents, a fine-tuned small model, or a careful evaluation setup. A college helpdesk RAG bot, a code review assistant, a multilingual document Q&A system and an LLM evaluation harness are strong, explainable choices.

Key takeaways

- The examiner wants your system design, not just an API call.
- Add evaluation: test questions, expected answers and a score.
- Open-source models via Ollama or Hugging Face avoid API costs.
- Handle hallucinations, prompt injection and private data explicitly.
- The PDF gives plans and steps, not finished source code.

Who this PDF is for

This PDF is for final-year B.Tech, MCA and M.Tech students building with large language models. Each idea is a complete system: data ingestion, retrieval or tools, the model call, guardrails, evaluation and a user interface. Plans use common open tools (LangChain or LlamaIndex, a vector database, Ollama or a hosted API) so a team of 3 or 4 can split the work.

How to choose your project

- **Own data matters.** RAG over your college or domain documents is more original than a general chatbot.
- **Budget for APIs** or plan to run small open models locally.
- **Build evaluation in** from week one with 50 to 100 test questions.
- **Know your limits:** say what the system should refuse to answer.

Standard level projects

1. College Helpdesk RAG Chatbot

- **Problem:** Answer student questions from official college documents.
- **Dataset:** Your college's public PDFs: prospectus, rules, academic calendar.
- **Tech stack:** Python, LangChain or LlamaIndex, Chroma or FAISS, Ollama or an LLM API, Streamlit.
- **Main modules:** PDF loader, chunking, embeddings, retriever, answer with citations, feedback button.

- **Steps:** Chunk documents, embed, retrieve top passages, prompt the model to answer only from them, show sources.
- **Show in the viva:** Answers with citations and a question it correctly refuses.
- **Extensions:** Add Hindi questions and answers.

2. Resume and Job Description Matcher

- **Problem:** Score resume fit and suggest improvements for a job post.
- **Dataset:** Sample resumes you write and public job posts.
- **Tech stack:** Python, an LLM API or local model, sentence-transformers, FastAPI.
- **Main modules:** Text extraction, skill extraction, similarity score, LLM feedback with structured output.
- **Steps:** Extract skills, compute similarity, ask the LLM for JSON feedback, validate the JSON.
- **Show in the viva:** Structured output and validation.
- **Extensions:** Add a bias check for names and gender words.

3. Meeting Notes Summariser

- **Problem:** Turn recorded meetings into notes and action items.
- **Dataset:** Your own recorded class or club meetings, with consent.
- **Tech stack:** Python, Whisper for speech-to-text, an LLM, Streamlit.
- **Main modules:** Transcription, speaker turns, summary prompt, action-item extraction, export.
- **Steps:** Transcribe, split long text, summarise each part, merge, extract action items with owners.
- **Show in the viva:** How you handled transcripts longer than the context window.
- **Extensions:** Add Hinglish transcription.

4. Text-to-SQL Assistant

- **Problem:** Let users ask database questions in plain English.
- **Dataset:** A sample database such as Chinook or Sakila.
- **Tech stack:** Python, an LLM, SQLite or MySQL, LangChain SQL tools.
- **Main modules:** Schema prompt, SQL generation, read-only execution, result explanation, guardrails.
- **Steps:** Give the model the schema, generate SQL, run it on a read-only connection, explain results.
- **Show in the viva:** Why the connection must be read-only.
- **Extensions:** Add a chart for numeric answers.

Advanced level projects

5. Multi-agent Research Assistant

- **Problem:** Search, read and summarise sources into a short cited report.
- **Dataset:** Web search results or the Kaggle "arXiv Dataset" abstracts.
- **Tech stack:** Python, LangGraph or CrewAI, an LLM, a search API.
- **Main modules:** Planner agent, search tool, reader agent, writer agent, citation checker.
- **Steps:** Define agents and tools, run on a topic, check every claim has a source.

- **Show in the viva:** The agent graph and a trace of one run.
- **Extensions:** Add human approval before the final report.

6. Code Review Assistant

- **Problem:** Review pull requests for bugs and style issues.
- **Dataset:** Your own GitHub repositories.
- **Tech stack:** Python, GitHub API, an LLM, a linter such as Ruff or ESLint.
- **Main modules:** Diff fetcher, linter run, LLM review prompt, comment formatter.
- **Steps:** Fetch the diff, run the linter, ask the LLM to review only changed lines, post comments.
- **Show in the viva:** Cases where the LLM was wrong and how you filter them.
- **Extensions:** Add tests suggested by the model.

7. Fine-tuned Small Model for a Domain

- **Problem:** Fine-tune a small open model to answer in a domain style.
- **Dataset:** A few thousand question-answer pairs you build from public domain material with a clear licence.
- **Tech stack:** Python, Hugging Face Transformers, PEFT (LoRA), Colab GPU.
- **Main modules:** Dataset building, LoRA fine-tuning, evaluation against the base model, demo.
- **Steps:** Prepare pairs, fine-tune with LoRA, compare base and tuned answers on a held-out set.
- **Show in the viva:** What LoRA changes and why it is cheap.
- **Extensions:** Combine the tuned model with RAG.

8. Multilingual Government Scheme Q&A

- **Problem:** Answer questions on public schemes in English and Hindi.
- **Dataset:** Public scheme documents from official government websites.
- **Tech stack:** Python, RAG stack, a multilingual embedding model, an LLM.
- **Main modules:** Bilingual ingestion, cross-lingual retrieval, answer in the user's language, citation.
- **Steps:** Embed documents with a multilingual model, retrieve for Hindi queries, answer with links.
- **Show in the viva:** Cross-lingual retrieval results.
- **Extensions:** Add voice input.

Research-oriented projects

9. LLM Evaluation Harness

- **Problem:** Measure answer quality, faithfulness and safety across models.
- **Dataset:** Your own 100-question test set plus public benchmarks you can download.
- **Tech stack:** Python, an evaluation library such as Ragas or your own scripts.
- **Main modules:** Test set, runners for several models, metrics, LLM-as-judge with human check, report.
- **Steps:** Run models on the same questions, score with metrics and a judge, compare with human ratings.
- **Show in the viva:** Where the judge model disagreed with humans.

- **Extensions:** Track cost and latency.

10. Prompt Injection Defence Study

- **Problem:** Test how RAG apps can be tricked by hidden instructions and defend them.
- **Dataset:** Documents you write with injected instructions.
- **Tech stack:** Python, your RAG app, guardrail libraries.
- **Main modules:** Attack set, baseline app, defences (input filters, instruction hierarchy, output checks), results.
- **Steps:** Build attacks, measure success rate, add defences one by one.
- **Show in the viva:** Attack success rate before and after each defence.
- **Extensions:** Map results to the OWASP Top 10 for LLM Applications.

11. Hallucination Detection for RAG

- **Problem:** Flag answers not supported by retrieved text.
- **Dataset:** Your RAG app's logs and a labelled set of supported and unsupported answers.
- **Tech stack:** Python, NLI models from Hugging Face, an LLM.
- **Main modules:** Claim splitting, entailment check, confidence score, UI warning.
- **Steps:** Split answers into claims, check each against sources with an NLI model, warn on weak support.
- **Show in the viva:** Precision and recall of the detector.
- **Extensions:** Auto-retry retrieval when support is low.

12. Image Generation with Controlled Style

- **Problem:** Generate images in a fixed style for a use case such as posters.
- **Dataset:** A small set of images you own or that are licensed for reuse.
- **Tech stack:** Python, Stable Diffusion via the diffusers library, LoRA or DreamBooth.
- **Main modules:** Prompt library, style fine-tuning, safety filter, evaluation by user survey.
- **Steps:** Fine-tune on your images, generate with fixed prompts, run a small blind survey.
- **Show in the viva:** Licensing and consent for training images.
- **Extensions:** Add ControlNet for layout control.

Report structure

A final-year report is marked on clarity as much as on code. A structure most universities accept:

1. **Abstract** (150 to 250 words): problem, method, main result.
2. **Introduction:** background, problem statement, objectives and scope.
3. **Literature review:** 8 to 15 papers or existing systems, and the gap your project fills.
4. **Requirements and design:** functional and non-functional requirements, architecture diagram, the system architecture (ingestion, retrieval or tools, model, guardrails, UI) and prompt templates.
5. **Implementation:** modules, tools, key algorithms and screenshots.
6. **Testing and results:** your evaluation set scores, latency and cost per query, and failure examples such as hallucinations.

7. **Conclusion and future scope.**
8. **References** in the format your department asks for (IEEE style is common).

Follow your own university's template if it gives one; it overrides this list.

Common mistakes to avoid

- Submitting a thin wrapper over a chat API with no design of your own.
- No evaluation set, so quality claims cannot be checked.
- Putting API keys in the code or GitHub repo.
- Using personal or copyrighted data without permission.

Frequently asked questions

Which generative AI project is best for a final year?

A RAG chatbot over your own documents with citations and an evaluation set is a strong, explainable choice. Agents and fine-tuning projects suit stronger teams.

Do I need to pay for an LLM API?

Not necessarily. You can run small open models locally with Ollama or on Colab. Hosted APIs are easier but cost money per use.

How do I evaluate a generative AI project?

Write 50 to 100 test questions with expected answers, score answers for correctness and faithfulness, and check a sample by hand.

Is a chatbot enough for a final year project?

Only if it has your own design: retrieval, tools, guardrails and evaluation. A plain chat window over an API is usually too thin.

Does this PDF include full source code?

No. It gives the problem, data, stack, modules and steps for each project. You write the code yourself, which is what examiners test in the viva.

Is this PDF free?

Yes. Fill in the short form on this page and the PDF downloads right away.

Sources and further reading

- LangChain documentation
- Hugging Face PEFT documentation
- Ollama
- OWASP Top 10 for LLM Applications

About this PDF

Prepared by the Uncodemy training team as practice project ideas. Datasets and tools named are public; check each dataset's licence before use. We update it every January. Corrections: info@uncodemy.com.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550