

Generative AI engineer (fresher)

Company: [Company name]
Location: [Location] | Work mode: [Work mode]
Employment type: Full-time | CTC: [CTC for the full-time role]
Joining: [Joining date] | Apply by: [Apply by]

About [Company name]

[About your company]

About the role

[Company name] is building features on large language models and needs a fresher generative AI engineer in the [Team] team. You will help build tools such as an assistant that answers questions from our documents: you will write prompts, connect an LLM API, set up document search and check each answer against its source. A senior engineer will review your work and show you how we test, log and control the cost of model calls.

What you will do

- Call hosted LLM APIs from Python, set timeouts and a limited number of retries, and handle rate-limit errors and empty replies without crashing the feature
- Write prompt templates for tasks such as answering from documents, sorting messages into categories and extracting fields into JSON, keep them in Git and change one thing at a time
- Load the company documents your senior has approved for use with an outside model, such as PDFs, Word files and web pages, ask before adding a new source, clean the text and split it into chunks that keep the title, page number and date
- Create embeddings for approved document chunks, store them in a vector database and retrieve passages using access rules a senior defines; test that users receive only documents they may read, and update the index when documents change or are removed
- Check retrieval before blaming the model: read which passages came back for a failed question, then test a different chunk size, overlap or number of passages
- Build a small evaluation set of real questions with expected answers, rerun it after every change to a prompt, model or chunk size, and record the scores in a shared sheet
- Add guardrails: answer only from the supplied passages, show the sources, say 'I do not know' when nothing relevant is found, check the output format and run simple prompt injection tests on a test copy, not the live system
- Record latency, token usage, errors and estimated cost using the team's approved logging settings; keep credentials and unapproved prompt or document content out of logs
- Wrap the feature in a Python API, for example with FastAPI, add input checks and tests, and write a README that shows how to run it
- Review weak answers with your senior, trace each one to retrieval, prompt or model, and add it to the team's list of known failures

What we are looking for

- Python at working level: functions, classes, exceptions, virtual environments, and reading and writing JSON files
- Calling REST APIs with the requests library or an official SDK, and handling errors, timeouts and JSON responses

- One project that calls a language model, such as a document question-and-answer app, where you wrote the prompts and can explain how you handled a bad answer. Share a repository or demo, or describe it or walk through it if the work is private
- A plain-language idea of embeddings, similarity search and what a vector database stores
- Everyday Git (branches, commits and pull requests), and SQL basics if the feature will read from a database
- The habit of checking an answer against its source and keeping examples of the model getting it wrong

Good to have

- LangChain or LlamaIndex used in a project, with a note on what each step did underneath
- FastAPI or Flask for serving a feature, and Pydantic for checking structured output
- An evaluation tool such as Ragas or promptfoo, or your own scoring script
- Docker basics, and a demo you can run locally
- Streamlit or Gradio for quick demos, and a tracing tool such as Langfuse for reading LLM calls

Who can apply

Final-year students and graduates of B.E./B.Tech in computer science, IT, AI and data science or electronics fit well, as do BCA, MCA, B.Sc and M.Sc candidates. Judge every applicant on the same requirements. A project that calls a language model, shown or described, matters more than the degree stream or a course certificate.

How to apply

Apply through this job post by [\[Apply by\]](#). Shortlisted candidates will hear from us about the next steps.

Generative AI engineer intern

Company: [Company name]

Location: [Location] | Work mode: [Work mode]

Employment type: Internship | Duration: [Internship duration] | Stipend: [Stipend for the internship]

Joining: [Joining date] | Apply by: [Apply by]

About [Company name]

[About your company]

About the role

[Company name] is looking for a generative AI intern to help a mentor improve one LLM feature, such as a question-answering tool over company documents. You will test prompts, build a small set of test questions, read wrong answers and help find out why they went wrong. You will not train models. You will finish with a short demo and a note on what worked.

What you will do

- Write a Python script that sends a list of test questions to an LLM API and saves each answer with its response time and token count
- Build a spreadsheet of real questions with the expected answer and the source page, then mark each model answer right, partly right or wrong
- Compare two chunk sizes on the same documents by checking which passages come back for the same ten questions
- Try three versions of one prompt on the same questions and report which works best and why
- Write out-of-scope and trick questions, such as 'ignore your instructions', run them against a test copy of the assistant and list what it got wrong
- Prepare a short demo and a README so the next person can run your work
- Use the development key and spending limit your mentor provides, and keep that key out of the code and the README

What we are looking for

- Pursuing a degree in computer science, IT, AI or a related subject, or recently graduated
- Python basics: functions, files, JSON and calling an API
- Has called a language model from code at least once, through a hosted API or a model on a laptop, and can explain what the prompt did
- Can say in your own words what an embedding is and why it helps search
- At least one project you can share on GitHub or describe if it is private, and a habit of reading model output carefully

Who can apply

Final-year students and graduates of B.E./B.Tech in computer science, IT, AI and data science or electronics fit well, as do BCA, MCA, B.Sc and M.Sc candidates. Judge every applicant on the same requirements. A project that calls a language model, shown or described, matters more than the degree stream or a course certificate.

How to apply

Apply through this job post by [Apply by]. Shortlisted candidates will hear from us about the next steps.

Screening questions to add to your job post

1. **How would you reduce the chance that an assistant answers questions its documents do not cover?** Look for checking whether the retrieved passages support the answer, showing the supporting sources and returning 'I do not know' when support is missing. Include test questions the documents cannot answer. A candidate should recognise that finding a similar passage does not establish that it contains the answer.
2. **What is prompt injection, in one sentence?** Look for text in a user message or a retrieved document that tries to override the assistant's instructions. A good extra is naming ways to reduce the risk: treat retrieved text as untrusted data, limit what the assistant can do and test with attack examples. No single step removes the risk.
3. **Besides reading an answer yourself, how do you check that your app's answers are correct?** Look for a set of test questions with known answers, a comparison with the source passage and a score recorded after each change. If the candidate has only tried a few questions by hand, ask how they would turn those into a repeatable test.
4. **What is chunking, and why does chunk size matter?** Look for splitting documents into pieces before embedding them, and the trade-off: large chunks bring in noise and cost more to send, small chunks lose context. The best answers say sizes were tested on real questions.
5. **If your project answers questions from documents, what did it say when the documents did not contain the answer?** Look for a specific reply the candidate saw and what they changed afterwards. If the project does not use documents, or they have not tried it, ask what they would test first.
6. **The same prompt gave a good answer yesterday and a worse one today. What could have changed?** Accept normal variation between model responses even when the inputs and settings are unchanged. Also look for checks on the model version, the full prompt, the retrieved passages and the generation settings. A good next step is to repeat the test and compare results; a lower temperature, where supported, does not guarantee identical answers.
7. **Before you send a company document to a hosted language model, what would you check, and who would you ask?** Look for personal details, customer records, contracts and anything marked confidential, and a habit of asking a senior before sending. Accept examples from college or personal projects.