2024

How to approach GenAI Coding Task in an Interview

Prepared By

@genieincodebottle



Table of Contents

Introduction

Typically, companies present one of three types of GenAI-related coding tasks

1. ### Take-home Assignments

   Take-home assignments for GenAI coding interviews allow candidates to work on a given problem statement at their own pace and in their own environment. These assignments require candidates to clearly document their approach, code, and results, and provide necessary artifacts such as code files, notebooks, and output data as specified by the interview guidelines.

2. ### Onsite Coding Round

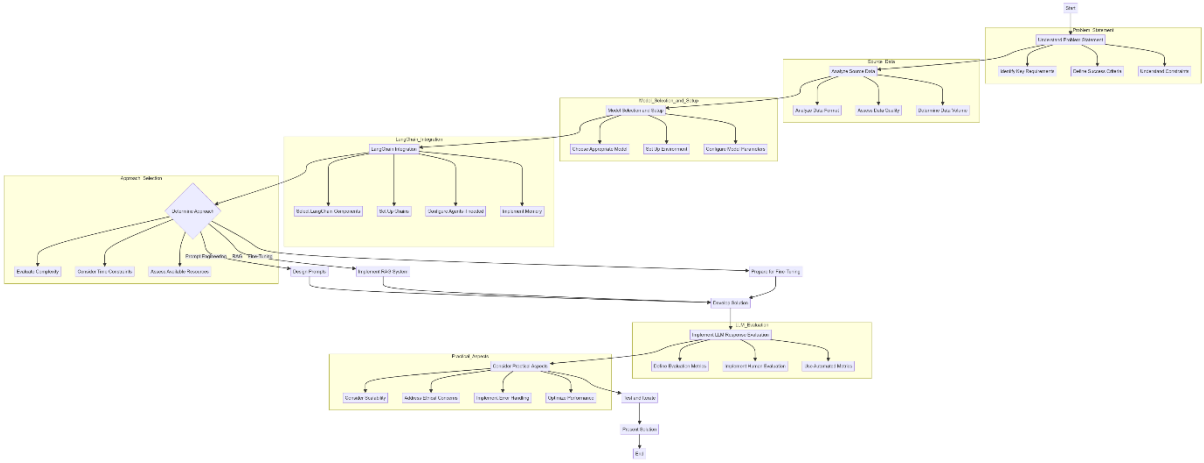
   The onsite coding round in GenAI coding interviews involves solving a problem in a controlled environment, often under time constraints. This round allows interviewers to assess candidates' technical skills, problem-solving abilities, and performance under pressure in a real-world, collaborative setting.

3. ### Online Coding Round

   The online coding round in GenAI coding interviews is a remote, timed session where candidates solve a given problem using an online platform. This round tests candidates' technical skills, problem-solving capabilities, and their ability to work efficiently under time constraints in a remote setting.

While many companies currently do not conduct coding interviews for GenAI positions, this approach may evolve with an increase in skilled GenAI candidates entering the job market.

Let's explore a more effective approach to tackling GenAI-related coding tasks that can boost your confidence during the interview and leave a positive, impactful impression on the interviewer.



Optimizing Approach: Flow Diagram for GenAI Coding Tasks

The flow diagram is not displayed clearly here due to its large size. For a high-resolution view, please visit the following link. You can download the image to zoom in and out as needed.

[High resolution image of Flow Diagram]

This flow diagram outlines the key steps in approaching a GenAI-related project coding task during an interview. Here's a breakdown of each major step:

Note: Topics highlighted in yellow can be discussed conceptually during the interview, as they can't be fully implemented within the time constraints of the coding task.

Sample GenAI Coding Setup for reference (Link)

Step – 1 Understand Problem Statement/Requirement

Ask clarifying questions to the interviewer if the following key topics are not explicitly mentioned in the task.

○ Identify key requirements

- **Functional Requirements**
  - **Define Input/Output:** Clearly specify the expected inputs (e.g. user queries, data formats) and desired outputs (e.g. response format, data structure).
  - **Specify User Interactions:** Outline how users will interact with the system, including any UI/UX considerations. Check with the interviewer whether you need to provide UI code or use tools like Google Colab or Jupyter Notebook to showcase task output without a UI. Confirm if IDEs such as VSCode or PyCharm can be used. Companies typically have their own setup for onsite or online coding rounds, but for take-home assignments, you usually have the liberty to use tools at your convenience and provide project artifacts (code, output, etc.) according to the guidelines provided by the interviewer
  - **Outline Core Functionalities:** List the main features the GenAI system needs to provide (e.g., Text Generation, Classification, Summarization, Text to Sql, Q&A chatbot, entity extraction etc).

- **Non-Functional Requirements**
  - **Performance Metrics:** Define expected response times, throughput, and accuracy levels.
  - **Scalability Needs:** Determine if the system needs to handle varying loads or grow over time. You can mention App scaling related details as per the given task and expected delivery of the task. It could be using load balancing, caching, horizontal scaling etc.
  - **Security Requirements:** Specify data privacy, user authentication, and other security measures.

- **Technical Requirements**

  - **Identify Tech Stack:** Determine which programming languages, frameworks, and tools will be used (e.g Python, LangChain LLM framework, Vector Database, Embedding Model, Database, Caching etc)
  - **Integration Points:** Specify how the GenAI system will integrate with existing systems or APIs.
  - **Deployment Environment:** Decide on the hosting environment (cloud, on-premise, hybrid).
  - **Business Requirements:**

    - ✓ **Align with Business Goals**: Ensure the project supports broader organizational objectives.
    - ✓ **Define Success Metrics**: Establish KPIs to measure the project's success.
    - ✓ **Consider Regulatory Compliance:** Identify any legal or industry-specific regulations that must be adhered to.

- Define success criteria

  What is expected output expected by Interviewer

- Understand constraints

  time to complete task, availability of resources, etc.

Step – 2 Analyse Source Data

- Analyse data format

  structured, unstructured, etc.

- Assess data quality

  Check if missing data or irrelevant source data

- Determine data volume

  Check if you need to handle large datasets or if you can work with smaller datasets.

Step – 3 Model Selection and Setup

Choose the Appropriate Model

Select the most suitable LLM based on the problem requirements, data characteristics, and project constraints. Determine if the task supports specific types of LLMs, if the interviewer provides any API-based or local LLMs, or if you have the flexibility to choose your own. Check the following sample GenAI project setup based on the LLM provider.

- **API Based Models**
- **Huggingface Open Source Model**
- **Ollama/gguf etc local model**

Link of sample coding setup

Set Up Environment

Prepare the necessary development environment, including installing required libraries and setting up any cloud resources if needed.

Configure Model Parameters

Set initial hyperparameters and other configuration options for the chosen model.

Step – 4 LangChain Integration

(You have the option to use other LLM frameworks, but I've found LangChain to be particularly mature with strong integration capabilities across various LLM providers, including open-source LLMs and Services.)

Select LangChain Components

Choose the appropriate LangChain modules and components based on the project requirements (e.g., prompts, llms, embeddings, vectorstores).

Set Up Chains

Configure LangChain chains to structure the flow of operations in your application.

- ### Configure Agents if needed
  If the project requires more complex decision-making, set up LangChain agents (In case need to build agentic GenAI system).

- ### Implement Memory
  Utilize LangChain's memory components to maintain context in conversations or across multiple interactions.

Step – 5 Determine Approach

- ### Evaluate Task Complexity
  Assess whether the task can be completed within the assigned timeline. Prioritize the development of key components based on the interviewer's priorities. If time permits, you can then focus on additional aspects such as the UI and evaluation.

- ### Prioritize Task
  This is crucial, as you may have limited time to develop a solution with all the essential deliverables (if not given as a home assignment). Prioritize efficiently to ensure key aspects are completed within the allotted time.

- ### Assess available resources
  Check with the interviewer if API-based services like LLM, database, and cache will be provided, or if it's acceptable to use in-memory solutions such as in-memory FAISS vector database, databases and cache etc.

- ### Choose among following strategy based on the requirement of the task
  - Prompt Engineering
  - Retrieval-Augmented Generation (RAG)
  - Fine-Tuning

Step – 6 Develop Solution

- ### Implement Solution

  Implement the chosen approach in previous steps

- ### Write clean, efficient code

  - **Follow Coding Standards:** Adhere to industry best practices and the coding standards of the language you are using.
  - **Modular Design:** Break down your code into manageable, reusable modules or functions to improve readability and maintainability.
  - **Comment and Document:** Add meaningful comments and documentation to explain the logic and flow of your code. This will help others understand your work and make it easier to maintain.
  - **Optimize Performance:** Ensure your code runs efficiently, utilizing appropriate data structures and algorithms to optimize performance.
  - **Error Handling:** Implement robust error handling to manage exceptions gracefully and ensure the program can handle unexpected situations without crashing.
  - **Testing and Validation:** Write unit tests to validate your code and ensure it works as expected. This will help catch bugs early and improve the reliability of your solution.
  - **Consistency:** Maintain a consistent coding style throughout your project to make it easier to read and understand. This includes consistent naming conventions, indentation, and formatting
  - Consider modular design for easier testing and maintenance

Step – 7 Implement LLM Response Evaluation

○ Define evaluation metrics (Use as per use case)

- **Automated Metrics**
  - ROUGE (Recall-Oriented Understudy for Gisting Evaluation): Measures overlap between generated text and reference text.
  - BLEU (Bilingual Evaluation Understudy): Compares machine-generated text to reference translations.
  - METEOR (Metric for Evaluation of Translation with Explicit ORdering): Considers synonyms and paraphrases.
  - BERTScore: Uses contextual embeddings to compute similarity.
  - Perplexity: Measures how well a model predicts a sample.
- **Human Evaluation**
  - Likert scales: Raters score responses on a scale (e.g., 1-5) for various attributes.
  - A/B testing: Comparing outputs from different models or versions.
  - Expert analysis: Subject matter experts review for factual accuracy and depth.
- **Task-Specific Metrics**
  - F1 score: For classification tasks.
  - Mean Average Precision (MAP): For information retrieval tasks.
  - BLEURT: For machine translation and text generation tasks.
- **Behavioural Testing**
  - CheckList: A task-agnostic methodology for testing NLP models.
  - Adversarial examples: Testing model robustness against manipulated inputs.
- **Contextual Coherence**
  - Coreference resolution accuracy
  - Discourse coherence measures

- **Factuality Checks**
  - Fact-checking against trusted sources
  - Knowledge graph consistency checks
- **Safety and Ethical Metrics**
  - Toxicity measures (e.g., using tools like Perspective API)
  - Bias detection in word embeddings and outputs
- **Efficiency Metrics**
  - Inference time
  - Model size and computational requirements
- **User Experience Metrics**
  - User retention rates
  - Task completion times
  - User feedback and ratings

- Use automated metrics where applicable

Step – 8 Consider Practical Aspects

- Address scalability
- Explain/Consider ethical implications
- Implement robust error/exception handling
- Optimize performance

Step – 9 Test and Iterate

- Conduct thorough testing (Unit, Integration, Smoke etc)
- Refine the solution based on evaluation results

Step – 10 Present Solution

- Explain your approach and reasoning
- Discuss trade-offs and potential improvements