WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Retrieval-Augmented Generation Framework for Placement Preparation: Design of a Document-Grounded Student Support Chatbot

Arun S R[1] ,Dhayanithi M[2], Abinayakarthik S[3],Abinesh S V[4], Guide: Mr. Suresh[5]

Department of Artificial Intelligence and Data Science
Sri Shakthi Institute of Engineering and Technology, Tamil Nadu, India

ABSTRACT :

Students preparing for campus placements often consult material distributed across notes, frequently asked questions, company preparation documents, aptitude resources, coding references, and interview guides. Finding relevant material and connecting it to a specific question can be time-consuming. This paper presents the design of a placement-preparation assistant based on retrieval-augmented generation (RAG). The proposed system uses a document-ingestion pipeline to prepare a searchable knowledge collection, retrieves relevant passages for a user query, and supplies the retrieved context to a language model to generate a response grounded in the available material. The project is implemented as a Python application with a Streamlit user interface and supporting ingestion and chatbot modules. The intended outcome is a conversational interface that helps learners locate and understand placement-related information while making the answer-generation process dependent on retrieved evidence. This draft describes the motivation, architecture, methodology, and implementation plan. Quantitative performance, user-study findings, and claims of improved accuracy are not reported because validated experimental measurements have not yet been supplied. These results must be added after testing the implemented system.

Index Terms—Retrieval-augmented generation, large language models, information retrieval, placement preparation, question answering, educational technology, document-grounded chatbot.

I. INTRODUCTION

Campus placement preparation requires students to review information from multiple sources and practice several types of tasks, including aptitude questions, programming problems, technical concepts, résumé-related topics, and interview questions. When resources are stored in separate files, locating the right explanation or example may require repeated manual searching. A conversational assistant can make this process more accessible, but a system that generates answers without consulting trusted material may produce statements that are incomplete, unsupported, or unsuitable for the learner's source collection.

Retrieval-augmented generation (RAG) addresses this design challenge by combining information retrieval with text generation. Rather than relying only on information encoded in model parameters, a RAG system retrieves relevant passages from an external knowledge collection and provides those passages as context to a language model. The system described in this paper applies this pattern to placement-preparation resources. A user submits a question through a Streamlit interface; the application retrieves context from indexed documents and uses that context to formulate a response.

The primary contribution of this work is a project-level design for a document-grounded placement-preparation chatbot, including document ingestion, retrieval, answer generation, and an evaluation plan. The paper does not claim that the system improves placement outcomes or outperforms a baseline. Such claims require controlled experiments and are intentionally left for future validation.

The remainder of the paper is organized as follows. Section II discusses related concepts and prior work. Section III describes the proposed methodology. Section IV presents the system architecture. Section V describes the implementation. Section VI defines the evaluation plan and identifies results that remain to be measured. Section VII discusses limitations and future work, followed by the conclusion and references.

II. RELATED WORK

A. Information Retrieval and Document Search

Traditional information retrieval ranks documents or passages according to their relevance to a query. Keyword-based methods can be effective when users know the terminology used in a document, but may be less effective when the wording of a query differs from the wording of the source. Modern semantic retrieval represents text as numerical embeddings and compares query and document representations in a vector space. The exact embedding model, similarity measure, index configuration, and ranking strategy used in the final project must be confirmed from the current source code.

B. Large Language Models and Question Answering

Large language models can generate fluent explanations and conversational answers. However, a generated answer is not automatically evidence that a statement appears in a student's reference material. In educational and preparation settings, it is useful to distinguish source-supported answers from general model knowledge and to provide a way for users to verify the information.

C. Retrieval-Augmented Generation

Lewis et al. introduced a retrieval-augmented generation approach that combines a parametric generator with a non-parametric memory retrieved for each input [1]. RAG-style pipelines commonly divide documents into passages, encode or index those passages, retrieve a subset for a query, and condition the generator on the retrieved context. This architecture is a suitable starting point for a placement assistant because its reference collection can be updated independently of the language model. The present project adopts this general design pattern; implementation-specific details and performance must be verified experimentally.

D. Research Gap and Project Motivation

The practical motivation for this project is to provide one conversational entry point to a curated set of placement-preparation documents. The system is intended to reduce manual search effort and make answers easier to trace to the supplied material. Whether it actually reduces search time, improves answer correctness, or increases learner satisfaction remains an empirical question and should be tested rather than assumed.

III. PROPOSED METHODOLOGY

A. Document Collection and Ingestion

The ingestion stage accepts the project's supported placement-preparation documents and extracts their text. The extracted content is normalized and divided into passages suitable for indexing. The final paper should specify the supported file formats, extraction libraries, corpus size, number of documents, and any preprocessing rules after these details are confirmed. Documents containing tables, diagrams, scanned pages, or code may require specialized extraction and quality checks.

B. Passage Segmentation and Indexing

Long documents are commonly divided into overlapping chunks so that retrieved passages remain small enough for efficient search while retaining useful context. Each chunk should retain metadata such as document name and, where available, page or section. The chunks are then represented in a searchable index. The chunk size, overlap, embedding model, vector-store technology, and index persistence format are implementation details that must be recorded from the running project rather than inferred.

C. Query Processing and Retrieval

When a learner submits a question, the application prepares the query for retrieval and ranks candidate passages by the configured relevance measure. The top-ranked passages are selected as context for generation. Retrieval quality depends on the document collection, chunk boundaries, embedding representation, ranking strategy, and selected number of passages. The final implementation should document any score threshold, filtering rules, and behavior when no passage is sufficiently relevant.

D. Context-Grounded Answer Generation

The retrieved passages and the user's question are passed to the configured language model through a prompt or message template. The prompt should instruct the model to prioritize supplied evidence, avoid presenting unsupported claims as facts, and state when the available context does not answer the question. If the application displays source references, those references should be traceable to the retrieved document or passage. The model provider, model identifier, prompt template, and citation implementation need confirmation from the current code.

E. User Interaction

A Streamlit interface provides the interaction layer. A learner enters a placement-related question, reviews the generated response, and may use the response to guide further study. The final version should document the exact user-interface features that are implemented, such as document upload, conversation history, source display, feedback collection, or error messages; these features should not be claimed unless present and tested.

IV. SYSTEM ARCHITECTURE

The proposed architecture is organized into six logical stages, summarized in Table I.

Stage	Responsibility
1. Knowledge sources	Placement notes, aptitude resources, coding references, technical material, and interview documents (actual corpus to be confirmed).
2. Ingestion	Extract and normalize text; retain document metadata.
3. Chunking and indexing	Split text into passages, create searchable representations, and persist the index.
4. Query and retriever	Accept a user query and select the most relevant passages.
5. Language-model generation	Generate a response conditioned on the query and retrieved context.
6. Streamlit interface	Present the question and response; show source evidence if implemented.

TABLE I. LOGICAL STAGES OF THE PROPOSED ARCHITECTURE

At query time, the logical flow is: user question → query processing → passage retrieval → context assembly → language-model response → user interface. The ingestion and indexing stages normally run when the reference collection is added or updated, rather than for every question. The actual deployment may combine or separate these stages across modules; the diagram above is a logical description, not a claim about process boundaries.

V. IMPLEMENTATION

A. Software Components

The project structure previously described includes Python modules for the Streamlit application and chatbot, an ingestion script, a data directory, and a requirements file. The application entry point has been identified as app.py; supporting files include chatbot.py and ingest.py. Earlier working copies also contained utility and vector-store-related directories. Because folder names and implementations can change between copies, the final submission should be checked against the exact version used for experiments.

B. Runtime and Dependencies

The reported interface uses Streamlit, with Python handling the ingestion and retrieval workflow. The exact Python version, operating system, embedding model, vector database or index, language-model provider, and dependency versions should be listed from the tested environment. API keys and other credentials must be supplied through environment variables or a secrets-management mechanism and must not be included in the paper's public repository or screenshots.

C. Reproducibility

For reproducibility, the final paper should record the corpus snapshot, document preprocessing configuration, chunk size and overlap, embedding model and version, retrieval depth, similarity threshold if used, language model and decoding parameters, prompt template, and hardware/software environment. A clean installation should be tested from the dependency file in a fresh virtual environment. Any stale absolute paths from a previous virtual environment should be removed before the project is packaged.

VI. EVALUATION AND RESULTS

A. Current Results Status

No validated experimental results have been provided for this manuscript. Therefore, this section does not report accuracy, precision, recall, response latency, user satisfaction, time saved, or comparisons against other systems. Such values must not be estimated or invented for journal submission.

B. Recommended Evaluation Protocol

A test set should be prepared from representative placement-preparation questions and reviewed reference answers. Questions should cover the actual supported content areas and include answerable, partially answerable, and unanswerable cases. Each test item should record the expected answer, supporting source passage, and evaluation criteria. If possible, at least two reviewers should assess answer correctness and evidence support independently. The measures recommended in Table II are subject to suitability for the project.

Measure	Purpose	How to report
Retrieval Recall@k	Checks whether a relevant reference passage appears in the top-k retrieved passages.	Relevant queries with a supporting passage in top k / all answerable test queries.
Answer correctness	Measures whether the response answers the question accurately.	Use a defined rubric and report the number of evaluated responses.
Evidence support	Checks whether factual answer claims are supported by retrieved sources.	Human-judged supported claims / all factual claims assessed.
Abstention behavior	Tests whether the system acknowledges missing evidence for unanswerable questions.	Report appropriate abstentions and unsupported answers separately.
Response latency	Measures user-perceived waiting time.	Report test count and median/percentile latency under stated conditions.
User usability	Assesses ease of use and perceived usefulness.	Use an approved questionnaire and report sample size and method.

TABLE II. RECOMMENDED EVALUATION MEASURES

C. Results Table to Complete After Testing

Experiment	Dataset / n	Result	Notes
Retrieval Recall@k	[ADD]	[MEASURE]	[k and relevance rule]
Answer correctness	[ADD]	[MEASURE]	[rubric / reviewers]
Evidence support	[ADD]	[MEASURE]	[claim-level method]
Median response latency	[ADD]	[MEASURE]	[hardware / network / model]

TABLE III. RESULTS TO BE COMPLETED AFTER TESTING — REPLACE ALL [ADD] AND [MEASURE] FIELDS

The [ADD] and [MEASURE] fields in Table III must be replaced with results from the tested implementation. If experiments have not been conducted,

this section should be retained as an evaluation plan only and not submitted as completed empirical research.

VII. DISCUSSION, LIMITATIONS, AND FUTURE WORK

The document-grounded architecture provides a practical way to connect a conversational interface with a maintained collection of placement-preparation resources. Its usefulness depends on the quality and coverage of the source documents, text extraction fidelity, retrieval relevance, and the language model's adherence to retrieved context. A response may still be incorrect even when retrieval succeeds, and a relevant passage may be missed because of segmentation or ranking choices. Source display and explicit uncertainty behavior can help users verify answers, but these features must be implemented and tested.

This manuscript has several limitations. First, the exact corpus composition and technical configuration have not been independently verified against the final project version. Second, no quantitative test results or user study have been provided. Third, no comparison with keyword search, a non-retrieval language-model baseline, or alternative retrieval settings has been conducted for this draft. Finally, the system is intended as a study aid and should not be presented as a guarantee of placement success or as a substitute for checking official company instructions and authoritative technical sources.

Future work should include a curated benchmark of placement questions, source-level attribution checks, evaluation of unanswerable queries, latency and cost measurements, feedback from student users under appropriate consent, and comparison with relevant baselines. Additional improvements may include document-level filtering, reranking, query rewriting, multilingual support, and safer handling of outdated or conflicting documents, if these are within the project scope.

VIII. CONCLUSION

This paper has presented a design for a Retrieval-Augmented Generation chatbot intended to support placement preparation through document-grounded question answering. The described workflow connects document ingestion and indexing with query-time retrieval, context-conditioned generation, and a Streamlit user interface. The architecture offers a clear basis for organizing placement resources and for evaluating the reliability of generated answers. However, the effectiveness of the implementation remains to be established through a reproducible test set, evidence-support assessment, latency measurements, and user evaluation. Completing these experiments and confirming the final implementation details are necessary before making empirical claims or submitting the manuscript as a completed research study.

REFERENCES

[1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 9459–9474. [Verify final bibliographic formatting and DOI/URL according to the target journal's requirements.]

[2] [ADD VERIFIED SOURCE] A primary source for the embedding model, vector database/index, and language model actually used in the implementation. Include exact model/tool versions and official publication or documentation details.

[3] [ADD VERIFIED SOURCE] Recent peer-reviewed work on retrieval evaluation, grounded question answering, or educational conversational agents that directly supports the discussion. Verify bibliographic metadata before inclusion.