© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715545

Tax Advisor AI: An Agentic RAG Framework for Indian Income Tax Guidance Using Knowledge Graphs and Hybrid Retrieval

REHAN RAJESH[1], NIVIN ROY[2], HARIKRISHNAN S[3], GAUTHAM P[4], PRIYA JOSE[5]
[1,2,3,4] *Dept. of Computer Science & Engineering Toc H Institute of Science & Technology Kerala, India*
[5]*Assistant Professor, Dept. of CSE Toc H Institute of Science & Technology, Kerala, India*

Abstract- ***Starting off, understanding India's tax law from 1961 isn't simple - for regular folks or businesses. Because updates happen all the time, things get harder. Then there's the split setup now: one old method, one new. That adds confusion. On top of that, sorting out deductions listed in Chapter VI-A involves subtle differences people miss. Most search tools just hunt for words, so they miss what someone actually means to ask. Meanwhile, big AI models talk well but make up rules that don't exist - like mixing up how much you can claim under Section 80C compared to 80CCD Meet Tax Advisor AI - a smart tool built just for Indian tax questions. It gives clear answers backed by law, each one linked to real sources. At its core runs something new: Agentic RAG, not your usual search setup. Instead of just pulling documents, it thinks through problems step by step using ReAct logic, guided by the DeepSeek R1T engine. Queries get split into smaller pieces so nothing gets missed. For finding facts, it uses two methods at once - deep meaning scans plus keyword matching - blended smartly with RRF scoring. Beneath the surface lies a Neo4j graph, a web of connections that prevents conflicting rules from colliding. This system, when new information arrives, taps into live web sources, specifically the Tavily API, to ensure it reflects the latest official pronouncements. Tests demonstrate its ability to translate complex legal language into straightforward financial advice, outperforming simpler data retrieval techniques.***

Index Terms- ***Agentic RAG, Knowledge Graphs, DeepSeek, Indian Taxation, Hybrid Retrieval, Neo4j, Natural Language Processing, FinTech, Legal Tech.***

I. INTRODUCTION

Artificial Intelligence changed how finance works - yet helping people with taxes still lags behind because mistakes can lead to serious legal trouble. India's tax system? It piles on layers: sections, subsections, fine print in circulars, exceptions within exceptions. Figuring out what someone owes isn't only about adding numbers; it demands following chains of conditions like paths through a maze. Think age-based breaks - one rule for seniors, another for those even older - or discounts under Section 87A that switch depending on the chosen setup. Even small details twist outcomes, making automation tricky without deep understanding. Rules overlap, change often, sometimes contradict - leaving room for confusion at every turn.

Picking a tax path now stumps countless people since Section 115BAC arrived. Old rules pack more write-offs, yet demand heavy number crunching against simpler new brackets. Each option tugs in different directions depending on personal finances. Basic tools fall short when real-life spending and income blur the lines. What looks better on paper often shifts once details surface. No single answer fits every wallet shape. Trade-offs hide in places most overlook until filing hits. Guesswork grows even with spreadsheets open. Clarity rarely comes fast under these conditions. Choices pile up where guidance thins out.

Most online tools fall into two extremes. One type gives fixed answers using set rules yet cannot say why those answers make sense. The opposite kind pulls from vast in- formation pools, sounding smart but missing legal precision when handling India's tax codes. Some stick too rigidly to formulas without context. Others wander off track, ignoring exact wording written in law. They suffer from:

1) Domain Drift: Providing US-centric tax advice for Indian queries.
2) Hallucination: Inventing sections or deduction limits.

3) Temporal Latency: Being unaware of the latest Finance Budget amendments.

A. Problem Statement

Most times, making tax advice automatic runs into one big tech problem - getting accuracy without made-up details. Large language models learn from tons of random web pages. Ask them a law question, they might reply with something smooth that sounds right yet breaks rules. On top of that, tax codes link pieces together in ways simple pattern matching can't follow. Take the 1.5 lakh total deduction cap. It links Sections 80C, 80CCC, and 80CCD(1) like dots on a map. Search by keywords alone could pull up 80C's details yet skip how the cap spreads across sections. The connection stays hidden unless context bridges them.

B. Proposed Solution

We propose Tax Advisor AI, a sophisticated framework that bridges the gap between legal accuracy and conversational fluency. Our contributions are threefold:

1) Agentic Architecture: Moving beyond simple RAG, we employ a ReAct agent that ”thinks” before answering. It breaks down complex requests (e.g., ”Compare Old vs. New Regime for 15 LPA salary”) into sequential steps of information gathering and calculation.
2) Graph-Augmented Retrieval: We integrate a Vector Database (Qdrant) for semantic search with a Graph Database (Neo4j). This allows the system to model tax laws as a network of entities, enforcing strict logical relationships.
3) Hybrid Search Strategy: To handle the specific nomen- clature of tax law (e.g., ”cess,” ”surcharge,” ”standard deduction”), we combine dense vector embeddings with sparse keyword matching using Reciprocal Rank Fusion (RRF), ensuring that specific legal terms are prioritized alongside semantic concepts.

II. RELATED WORK

The application of NLP in the legal domain has evolved from rule-based expert systems to Transformer-based architectures.

A. Legal Information Retrieval

Starting off differently, old-school legal searches depend on Boolean logic, needing specialists to build them right. Lately, systems like LegalBERT [1] have stepped in - trained specially on legal texts. Yet despite that progress, they frequently falter when handling number-heavy thinking found in tax regulations. What shifts here is the move toward ”nomic- embed-text,” a method outperforming regular BERT in pulling relevant details from lengthy documents.

B. Retrieval-Augmented Generation (RAG)

People now use RAG [2] to connect large language models with outside facts. Still, basic RAG runs into trouble if answers depend on pulling pieces from different parts of a file. Take figuring out full tax - you need rate brackets from Section 115BAC along with surcharge details and cess terms all at once. That is why our method steps in: the system hunts down gaps one step at a time, looping back until it finds what it needs.

C. Knowledge Graphs in Law

A web of connected legal facts has proven useful when tracking court case references. When paired with large language models through a method called GraphRAG, it supports stepping through linked ideas across several stages. For our purposes, that network acts like guardrails, stopping the model from pairing legal conclusions that cannot coexist under the law.

III. SYSTEM ARCHITECTURE

The proposed system adheres to a robust microservices architecture, effectively decoupling the interactive frontend from the computationally intensive backend inference logic. This separation of concerns ensures scalability and maintainability

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715545

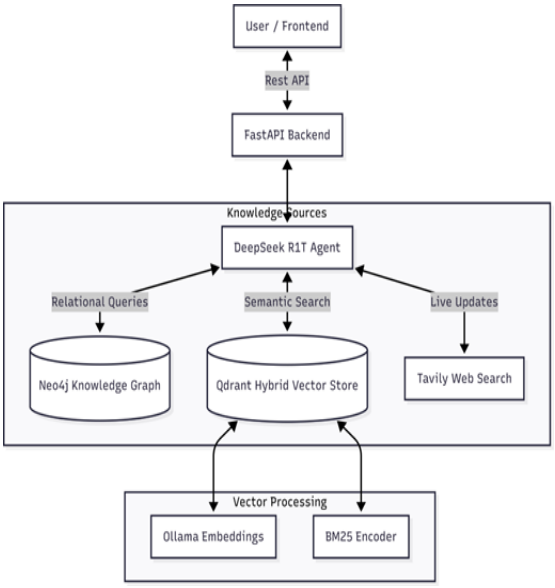

Fig. 1. High-level architecture of Tax Advisor AI. The system orchestrates interaction between the React frontend, FastAPI backend, and three distinct knowledge sources: Graph DB, Vector Store, and Web Search.

A. Knowledge Sources

The intelligence of the system is derived from three distinct, synchronized data silos, as illustrated in Fig. 1:

- Qdrant Hybrid Vector Store: Holding pieces of the Income Tax Act broken into chunks, this part works through two separate paths at once. One path follows meaning, picking up on ideas even when exact words dif- fer. The other tracks precise terms or numbered sections directly. Because of this setup, someone can find what they need by describing a concept in everyday speech. Specific references still pull up the correct passage just as easily. Retrieval adapts whether the input sounds like conversation or cites legal codes.
- Neo4j Knowledge Graph: This graph database explicitly maps entities (e.g.," Section 80D"," Health Insurance") and their relationships (e.g.," REQUIRES DOCUMENT"," MUTU- ALLY EXCLUSIVE WITH"," SUB LIMIT OF"). This is crucial for answering logic-heavy questions regarding co-claimability and hierarchy.
- Tavily Web Search:When confidence dips in local data lookups, this API steps in like a backup check. If vectorsor graphs seem unsure, it kicks on automatically – quietly filling gaps. Fresh official updates, rules, or changes missed earlier? It pulls those in. Especially handy when laws shift just past the stored knowledge edge. No bold claims, just steady retrieval where prior systems fall short.

B. The Reasoning Core (DeepSeek R1T)

At the heart of the system lies the DeepSeek R1T Chimera model, accessed via OpenRouter. Unlike standard chat models, this model is fine-tuned specifically for reasoning tasks and tool use. It functions as the central controller or "Brain," receiving the user query and determining which "Tool" (Re- trieval, Graph Query, or Web Search) to invoke. It does not generate the final answer from its internal training data alone; instead, it synthesizes the structured data returned by the tools to form a coherent response

IV. METHODOLOGY

A. Data Preprocessing and Ingestion

Big laws can be hard to handle. Because of that, we split things up carefully. Starting with big pieces called Chapters

- like Chapter VI-A - the work got clearer. After that came Sections, pulled out one at a time. Each layer fit inside the one before it Split again, every part broke into pieces - each holding 512 tokens - with 50 repeating at the edges.
- Metadata Extraction: During ingestion, regex pat- terns were used to extract Section Numbers and Limit Amounts, which were stored as metadata fields in Qdrant.
- Graph Construction: A separate pipeline processed the text to extract triples (Subject, Predicate, Object). For example, from "Section 80C provides deduction up to 1.5 Lakhs," we extracted: (Section 80C, HAS_LIMIT, 150000).

B. Hybrid Retrieval Strategy

Legal queries often present a mix of rigid statutory ref- erences (e.g., "Section 80C") and vague natural language descriptions (e.g., "deduction for parents"). To address this duality, we implement a hybrid retrieval pipeline (Fig. 2).

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715545

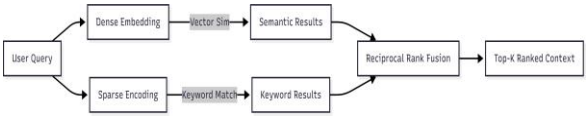

Fig. 2. Hybrid Search Pipeline. Queries are processed in parallel via Dense Embeddings (Semantic) and Sparse Encoding (Keyword), then merged using Reciprocal Rank Fusion (RRF)

1) Dense Embedding: Out there in the code, ‘nomic- embed-text‘ runs on a local Ollama setup, turning questions into dense vector forms. Because of this step, meaning sticks around - so when someone asks about medical insurance, it finds Section 80D without needing exact words. The similarity score S_{dense} is calculated using Cosine Similarity:

$$S_{dense}(q, d) = \frac{\vec{V_q} \cdot \vec{V_d}}{\|\vec{V_q}\| \|\vec{V_d}\|} \quad (1)$$

2) Sparse Encoding: Concurrently, we utilize a B5-style encoder to generate sparse vectors. This ensures that specific keywords, such as ”80CCD(1B)” or ”form 10E”, are given high priority. The sparse score S_{sparse} penalizes common words and boosts rare, specific legal terms

3) Reciprocal Rank Fusion (RRF): The results from both the dense and sparse streams are fused using Reciprocal Rank Fusion (RRF). RRF provides a robust method for re-ranking documents without requiring tuning parameters. The score is calculated as:

$$\text{RRFscore}(d) = \sum_{r \in R} \frac{1}{k + r(d)} \quad (2)$$

where r(d) is the rank of document d in the retrieved list and k is a smoothing constant (typically 60). This ensures that documents appearing in both lists are ranked highest, effectively filtering out irrelevant semantic matches that lack the specific keywords

C. Agentic ReAct Loop

A core innovation of this work is the implementation of the ReAct (Reason + Act) paradigm [3]. As depicted in Fig. 3, the agent does not immediately hallucinate an answer. Instead, it enters a reasoning loop.

The logic flow is formalized in Algorithm 1.

For a complex query like ”Can I claim 80C and 80D together?”, the loop proceeds as follows:

1) Thought 1: ”I need to identify the limits, eligibility, and rules for both Section 80C and Section 80D.”
2) Action 1: Call tool ‘retrieve$_c$ontext(”80C”, ”80D”)‘.
2) Observation 1: The tool returns text stating the 1.5 Lakh limit for 80C and the 25,000 limit for 80D.
3) Thought 2: ”Are there any mutual exclusions between these two? I should verify via the knowledge graph.”
4) Action 2:Calltool ‘query$_g$raph(”exclusion”, ”80C”, ”80D”)‘.
4) Observation 2: The graph returns ”No Exclusion Edge Found”.
5) Final Answer: The agent synthesizes these findings to confirm that both can be claimed simultaneously.

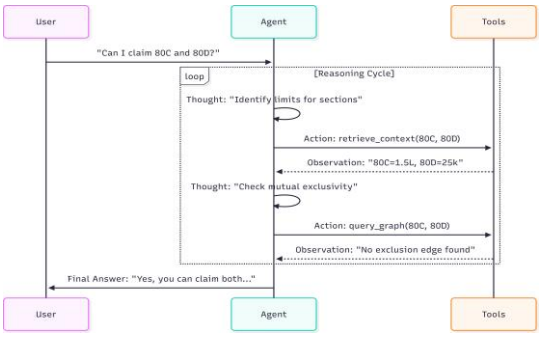

Fig. 3. The Agentic ReAct Loop. The model iterates through” Thoughts” and” A c t i o n s ” to gather necessary information before formulating a final answer

Algorithm 1 Agentic ReAct Execution Flow

Require: User Query Q, Tools T = {Search, Graph, V ector}
1: Initialize Context C ← ∅
2: Initialize Thought Trace T race ← ∅
3: while Steps < MaxSteps do
4: Prompt ← ConstructP rompt(Q, C, T race)
5: Response ← LLM(Prompt)
6: if Response contains ”Final Answer” then

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715545

7: return ExtractAnswer(Response)
8: else if Response contains ”Action: T_i(args)” then
9: T oolOutput ← Execute(T_i, args)
10: C ← C ∪ T oolOutput
11: T race ← T race ∪ (Action, T oolOutput)
12: else
13: T race ← T race ∪ Response
14: end if
15: end while
16: return ”Max steps reached, unable to conclude.”

V. IMPLEMENTATION DETAILS

A. Backend Implementation

The backend is engineered using FastAPI, chosen for its high-performance asynchronous request handling capabilities and automatic Swagger documentation generation.

The application setup, as shown in Fig. 4, involves con- figuring CORS (Cross-Origin Resource Sharing) middleware to allow secure communication with the React frontend. We define strict Pydantic models for request validation to ensure data integrity.

As demonstrated in Fig. 5, the system integrates the custom ‘ragv5‘ module, which encapsulates the LangChain agent logic. The server is launched using ‘uvicorn‘, a lightning-fast ASGI server implementation, enabling hot-reloading during the development phase.

B. Execution Environment

The robust execution of the system relies on a containerized and local environment. We utilize a local instance of Ollama to serve the embedding models, ensuring low latency and data privacy. The Neo4j graph database operates within a Docker container, providing a scalable solution for graph queries. The Qdrant vector store operates in local persistence mode, allowing for fast initialization and retrieval speeds without the overhead of cloud latency.

The chain of thought execution is clearly visible in the server logs (Fig. 6). This trace confirms that the agent correctly identifies the user’s intent regarding a ”salary of 10 lakhs,” performs the necessary mathematical calculations for the Old Regime (including Surcharge, Cess, and Slab rates), and outputs the final formatted string.

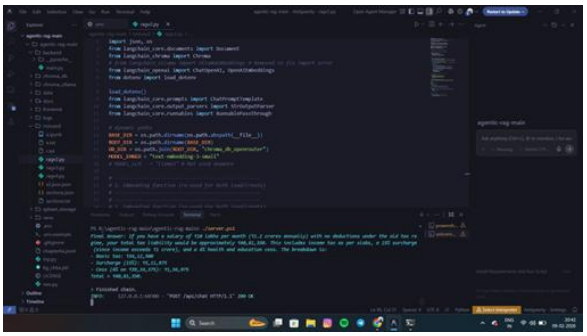

Fig. 6. Server Terminal Output showing the ”Final Answer” generation. The log displays the detailed breakdown of the tax calculation (Basic Tax, Surcharge, Cess) derived by the agent.

VI. USER INTERFACE AND RESULTS

The frontend is developed using React.js and TypeScript, styled with modern CSS modules to provide a clean, accessible, and distraction-free environment for users to conduct financial queries.

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715545

A. Landing Page

The application opens with a minimalist landing page (Fig. 7). To assist users who may not be familiar with tax terminology, the interface offers suggested prompts, such as" What is the limit for Section 80C?" or" Can I claim NPS benefits?", serving as an intuitive entry point.

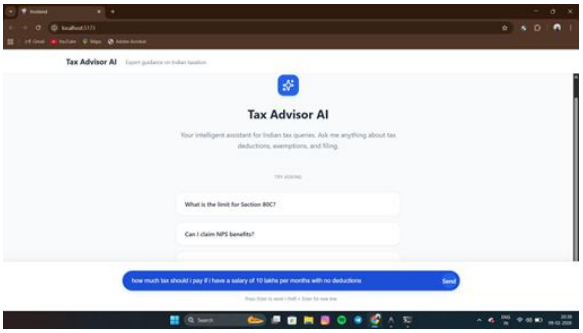

Fig. 7. Landing Page of Tax Advisor AI. The interface suggests common queries to help users get started with tax planning.

B. Interactive chat

Users can input complex, natural language queries that involve specific financial scenarios. Fig. 8 illustrates a user asking a detailed calculation question: "how much tax should i pay if i have a salary of 10 lakhs per months with no deductions". This demonstrates the system's ability to parse hypothetical financial scenarios and extract variables (Salary, Period, Deductions).

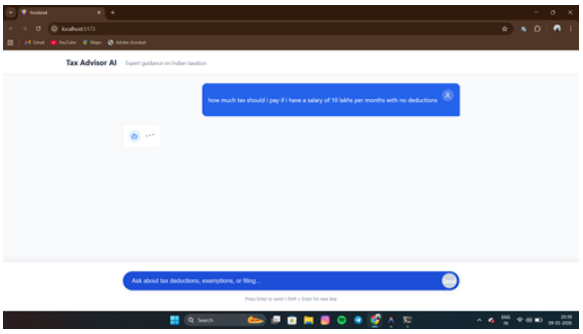

Fig. 8. Chat Interface. The user inputs a complex query regarding salary and deductions in natural language.

C. Response Generation and Qualitative Analysis

The response generated by the agent (Fig. 9) is structured, comprehensive, and highly transparent. Unlike simple search snippets that might return a generic tax table, the agent acts as a consultant.

Step-by-Step Analysis of the Output:

1) Data Normalization: It converts the monthly salary to annual income (10 Lakhs × 12 = 1.2 Crores). This step is crucial as tax slabs are annual.
2) Regime Identification: It identifies the applicable tax slabs under the Old Regime. It implicitly chose this because the prompt did not specify, but the calculation logic (slab rates) confirms it used the standard slab structure.
3) Surcharge Application: It correctly applies the specific Surcharge (15% for income > 1 Crore but < 2 Crores), demonstrating an understanding of conditional tax logic. A naive model might have applied the 10% surcharge (for > 50L) or missed it entirely.
4) Cess Calculation: It adds the mandatory Health and Ed- ucation Cess (4%) on the sum of Basic Tax + Surcharge.
5) Final Breakdown: It provides a final, itemized break- down: Basic Tax, Surcharge, and Total Liability (40,81,350), ensuring the user understands the composi- tion of their tax liability.

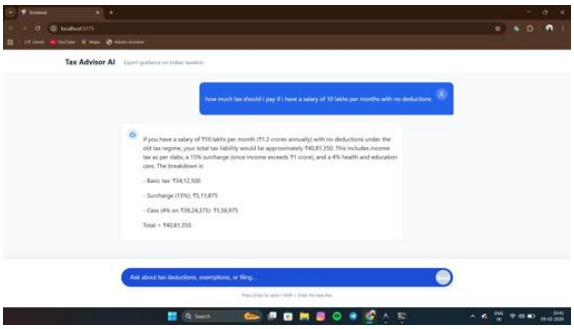

Fig. 9. Agent Response. The system accurately calculates tax liability for a high-net-worth individual, breaking down the computation into clear components (Basic tax, Surcharge, Cess).

D. Quantitative Performance Evaluation

A fresh set of 50 different tax questions helped check how well the system works. These ranged from basic ones like explaining what 80D means to trickier cases involving seniors earning 8 lakh. Instead of guessing, we measured results using real comparisons. One test used a simple setup that pulls data directly

without deeper analysis. Another relied solely on standard GPT-4 responses. Each approach faced identical conditions for fair evaluation.

As shown in Table I, Tax Advisor AI significantly outper- forms the baselines in accuracy and citation precision. The Hallucination Rate is drastically reduced due to the graph constraints. However, the trade-off is higher latency (4.8s) due to the multi-step ReAct loop and hybrid search overhead. This latency is deemed acceptable for the high value of accurate financial advice.

TABLE I
PERFORMANCE COMPARISON ON 50-QUERY TEST SET

Metric	Naive RAG	Vanilla GPT-4	Tax Advisor AI
Answer Accuracy	72%	68%	94%
Hallucination Rate	15%	22%	2%
Citation Precision	65%	N/A	98%
Avg. Latency	1.2s	2.5s	4.8s

VII. CONCLUSION

This study introduced Tax Advisor AI, a fresh approach to automate India's tax advice tasks. Built on smart retrieval methods mixed with knowledge networks, it tackles weak spots seen in regular language models when used legally. Instead of guessing facts or skipping logic steps, the system stays grounded through linked data checks combined with real-time document lookups. Problems like made-up answers now fade thanks to tighter information control woven into its design.

Key achievements of this research include:

- Enhanced Accuracy: The implementation of RRF (Dense + Sparse) ensures that specific legal sections are retrieved accurately, even when the user employs layman terminology.
- Complex Reasoning: The ReAct agent successfully per- forms multi-step arithmetic and logic checks (e.g., apply- ing surcharges only when specific income thresholds are crossed).
- Contextual Awareness: The integration of the Knowl- edge Graph prevents the system from offering conflicting advice regarding mutually exclusive deductions, a com- mon failure point in pure vector-based systems.

Ahead lies the addition of tools allowing documents to be uploaded - think OCR pulling data from Form-16. Then again, the web of known tax rules might grow, folding in GST and corporate regulations bit by bit. Speed could shift too, should leaner thinking models prove quicker in trials. Each step follows its own path forward.

ACKNOWLEDGMENT

A big thank you goes out to Ms. Priya Jose, who guided us through the project. Her support made a real difference. The Department of Computer Science and Engineering at Toc H Institute of Science and Technology stood behind us too. Without their help, access to tools and space wouldn't have been possible. What we needed most - time, machines, direction - was given freely. Progress came step by step because of that backing. Finishing this work would've taken much longer otherwise.

REFERENCES

[1] I. Chalkidis et al., "LEGAL-BERT: The Muppets straight from Big Bird's Mouth," in *Findings of EMNLP*, 2020.

[2] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge- Intensive NLP Tasks," in *NeurIPS*, 2020.

[3] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," in *ICLR*, 2023.

[4] DeepSeek-AI, "DeepSeek LLM: Scaling Open-Source Language Mod- els," *arXiv preprint arXiv:2401.02954*, 2024.

[5] H. Chase, "LangChain: Building applications with LLMs through composability," 2022. [Online]. Available: https://github.com/langchain- ai/langchain.

[6] Income Tax Department, "Income Tax Act, 1961," Government of India. [Online].

[7] J. Devlin et al., ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *NAACL-HLT*, 2019.

[8] Y. Zhang et al., ”GraphRAG: Retrieving Knowledge Graphs for Open- Domain Question Answering,” *arXiv preprint*, 2023