Retrieval on Verilog Repositories: A Knowledge-Graph Based Solution

Adi Szeskin[*†]
Intel AI Solutions Group
adi.szeskin@intel.com

Itay Lieder[*]
Intel AI Solutions Group
itay.lieder@intel.com

Divyasree Tummalapalli
Intel AI Solutions Group
divyasree.tummalapalli@intel.com

Amitai Armon
Intel AI Solutions Group
amitai.armon@intel.com

Abstract

We present a retrieval system for answering questions about Verilog / System Verilog code bases. Standard vector RAG (retrieval augmented generation) often fails on hardware description languages due to identifier renaming, coding-style variation, hierarchy, and concurrency. We instead construct knowledge graphs over the code and its LLM-generated explanations and retrieve based on the entities and relations. We achieve this by adapting the GraphRAG package, originally intended for natural language, to our specific code use-case. We compare (i) standard semantic retrieval on the explanations, (ii) GraphRAG over code and (iii) GraphRAG over the explanations. On a corpus of $\sim$3.5K files and a benchmark of 29 questions, using top-1 *file-level* recall, the first baseline reaches 31%. GraphRAG consistently outperforms it, achieving 55–59% when utilizing the explanations, and up to **79%** when considering retrieved equivalent files. Constructing the graph with `GPT-4o-mini` worked well without requiring the larger GPT-4o, but `GPT-4o` was required for answering the queries better. Our results indicate that the suggested graph-based approach could be useful for answering questions of hardware designers on the code base.

1 Introduction

Retrieval–Augmented Generation (RAG) pipelines [1] retrieve relevant context to ground an LLM. This works well for natural language but degrades on hardware description languages (HDLs): identical functionality can appear under different names and modular decompositions; behavior depends on hierarchy, parameterization, preprocessor macros, event controls and concurrency, rather than just on local tokens. Consequently, keyword or embedding similarity alone may miss the correct file/module.

We take a graph-based approach that makes structure explicit [2]. Our system builds a Knowledge Graph (KG) over Verilog sources and their natural-language explanations and performs retrieval by traversing relations between entities (modules, instances, ports, parameters, signals). We utilize textual explanations that we generated for each code file using GPT-4o. The KG is constructed by adapting the GraphRAG package [3, 4] to our use-case, utilizing LLMs for identifying the relevant System Verilog entities and the realtions between them. We evaluate three strategies: (i) standard hybrid semantic retrieval based on the explanations, combining embedding retrieval with a TF-IDF

[*]Equal contribution
[†]Alternative email: adiszes@gmail.com

search, (ii) GraphRAG over the code explanations and (iii) GraphRAG over the code itself. We found GraphRAG over the explanations to be the strongest on our benchmark.

HDL-specific considerations. There are multiple challenges in performing standard retrieval over HDLs. Designs are hierarchical and reused (parameterized instantiations across files); the preprocessor (`‘define`, `‘ifdef`, `‘include`) and missing include paths change the effective structure; `generate` constructs create topologies that exist only post-elaboration; concurrency and timing semantics (blocking vs. non-blocking, `always` and events) are weakly signaled in tokens; vendor extensions (e.g., interfaces/modports, pragmas) complicate parsing; and multiple modules per file create a granularity mismatch when applying file-level evaluation. To address these challenges, we explored the use of LLM-built knowledge graphs (GraphRAG/LightRAG) [3–5].

2 Related Work

Semantic code search. Beyond lexical or generic sentence embeddings, semantic methods capture source-code meaning using conceptual and structural information. Pre-trained models such as CodeBERT and GraphCodeBERT enrich token representations with data-flow and context[6, 7].

KGs and RAG. KGs support multi-hop/explainable retrieval and can outperform flat vector RAG when structure matters [1, 8]. In software engineering, KGs are known to aid question/code retrieval and interpretability [9]; planning over KGs for LLM-RAG has also been explored [10]. Structured knowledge improves text comprehension for retrieval [11].

GraphRAG for code. Graphical RAG builds graphs from code or explanations and retrieves by traversing relations; examples include bridging natural and programming languages [12] and repository-level completion with code-context graphs [13]. Our work targets HDLs, where hierarchy, elaboration, and concurrency dominate semantics, and compares standard retrieval with GraphRAG over explanation or code KGs [3–5].

3 Method

Adaptations for HDL. Unlike prior GraphRAG setups, which assume free-form natural language text, our pipeline required extensive prompt adaptation. The original prompts were tuned for generic documents and did not handle source code or structured explanation fields. We therefore re–engineered all prompts, using GPT-4o to assist in defining entity and relation extraction rules specific to Verilog / System Verilog code and to our explanation format (modules, ports, parameters, instances). In the retrieval stage, we restricted traversal to local graph neighborhoods, without additional global search heuristics, to better match HDL query patterns. Additional details are provided below.

Graph construction. We applied the GraphRAG steps for creating the KG using an LLM: entity/community detection and relation extraction [3, 4]. We created nodes for *modules*, *instances*, *ports/signals*, *parameters*, and *files*, plus *functional concepts* extracted from explanations. Edges encoded INSTANTIATES, CONNECTS, PARAMETERIZES, DEFINES, IMPLEMENTS, and DESCRIBED_BY. Each node also carried a textual *surface*: name and explanation sentences, which were embedded by GraphRAG [3]. For explanation –based graphs, LightRAG served as a lighter version for comparison [5].

Query processing. Given a natural-language question, we applied GraphRAG to (i) retrieve seed nodes by semantic match over node *surfaces*; (ii) expand to h-hop neighborhoods, $h \in \{1, 2\}$, and (iii) rank candidate nodes and their ancestor files [3].

Baseline. *Hybrid semantic retrieval on code explanations*: Retrieving embedded chunks from LLM-generated explanations, using the Text-Embedding-3-Large model [14], combined with retrieved chunks by TF-IDF keyword search. The scores of both methods were normalized and averaged, and the results were ranked by the combined score.

Table 1: Top-1 file-level recall for GraphRAG across data types, sizes, and models.

Data type	# Files	Sampling method	Graph build model	Query model	Recall@1
Explanations - full	**3.5K**	**All files**	**GPT-4o-mini**	**GPT-4o**	**55%**
Explanations - full	**3.5K**	**All files**	**GPT-4o-mini**	**GPT-4o-mini**	**45%**
Explanations sample	*1K*	*Top-15 / q + all correct*	*GPT-4o-mini*	*GPT-4o*	*55%*
Explanations sample	*1K*	*Top-15 / q + all correct*	*GPT-4o-mini*	*GPT-4o-mini*	*38%*
Explanations sample	*160*	*Top-2 / q + all correct*	*GPT-4o-mini*	*GPT-4o*	*59%*
Explanations sample	*160*	*Top-2 / q + all correct*	*GPT-4o-mini*	*GPT-4o-mini*	*45%*
Code sample	*160*	*Top-2 / q + all correct*	*GPT-4o-mini*	*GPT-4o*	*38%*
Code sample	*160*	*Top-2 / q + all correct*	*GPT-4o-mini*	*GPT-4o-mini*	*31%*

The baseline method achieved 31% on the 29 questions. The sampled top distractors were fixed per question.

4 Experimental Setup and Results

We evaluated 29 user questions, for which we measured performance on both explanation- and code-based retrieval. The metric used was **top-1 file-level recall**. The baseline (file retrieval over explanations) achieved **31%**. The corpus had $\sim$3.5K files with code and explanations. For efficiency we first sampled up to 1K files, later expanding to the full dataset. Sampling always included each question's ground-truth file and strong distractors (top-2 or top-15 results for each question from the baseline). We included the sampling results for the explanations, even though we later used the full dataset, since they indicated the accuracy behavior for smaller sizes of datasets. Apparently, the sampling with distractors roughly reflected the accuracy on the full dataset. The code results were weaker than those for the explanations, and they are provided for the sampled data to indicate the differences between the two methods and the impact of the sampling with distractors.

We built KGs with GraphRAG and LightRAG [3–5]; queries used `GPT-4o` or `GPT-4o-mini`. `GPT-4o-mini` is economical for graph construction; `GPT-4o` performs better at answering queries. For our top result, when considering functionally equivalent files as a successful retrieval, our recall improved to 79%. A LightRAG run provided substantially lower recall (45% vs. 79%), and we omitted these results from this manuscript.

5 Discussion

Why graphs help. GraphRAG makes relationships explicit (modules, instances, parameters, signals, interfaces) and enables multi-hop reasoning over HDL hierarchy and reuse [4]. The explanation layer acts as a semantic normalizer, reducing sensitivity to local syntax and identifier choice.

Cost/latency. Graph construction with `GPT-4o-mini` scales well, while `GPT-4o` yields better query accuracy. For larger graphs, caching/AST seeding may reduce rebuilds.

Error modes and validity. Common failures include (i) granularity mismatch (right module, wrong file), (ii) hierarchy/parameter drift (near-miss along parent/child), (iii) dialect/preprocessor gaps obscuring edges, and (iv) LLM graph artifacts (missing or merged relations). Module/subgraph-level evaluation and AST-seeded edges may mitigate these. Our benchmark of 29 questions is modest, and equivalence scoring involves human judgment; costs/latencies are hardware/provider dependent, and they may change for different codebases. Still, we believe the large accuracy increase following the use of GraphRAG in this setting is of significant interest.

Practical implications and next steps. Our recommended configuration: build the explanation graph with `GPT-4o-mini` and answer queries with `GPT-4o`. A next step could be re-ranking a KG shortlist with a cross-encoder or AST-aware scorer. **Potential future work may include:** (1) AST-assisted GraphRAG using PyVerilog or vendor toolchains for deterministic cores [15]; (2) hybrid retrieval with learned reranking over KG+vector pipelines [8]. These directions may further improve the retrieval results.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-

augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[2] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE symposium on security and privacy*, pages 590–604. IEEE, 2014.

[3] Microsoft. Microsoft graphrag documentation, 2024. `https://microsoft.github.io/graphrag/`.

[4] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[5] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2024.

[6] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[7] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, 2020.

[8] Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, Sunil Patel, and Stefano Pasquali. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 608–616, 2024.

[9] Mingwei Liu, Simin Yu, Xin Peng, Xueying Du, Tianyong Yang, Huanjun Xu, and Gaoyang Zhang. Knowledge graph based explainable question retrieval for programming tasks. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 123–135. IEEE, 2023.

[10] Junjie Wang, Mingyang Chen, Binbin Hu, Dan Yang, Ziqi Liu, Yue Shen, Peng Wei, Zhiqiang Zhang, Jinjie Gu, Jun Zhou, et al. Learning to plan for retrieval-augmented large language models from knowledge graphs. *arXiv preprint arXiv:2406.14282*, 2024.

[11] Aakash Mahalingam, Vinesh Kumar Gande, Aman Chadha, Vinija Jain, and Divya Chaudhary. SKETCH: Structured knowledge enhanced text comprehension for holistic retrieval. *arXiv preprint arXiv:2412.15443*, 2024.

[12] Kounianhua Du, Jizheng Chen, Renting Rui, Huacan Chai, Lingyue Fu, Wei Xia, Yasheng Wang, Ruiming Tang, Yong Yu, and Weinan Zhang. Codegrag: Bridging the gap between natural language and programming language via graphical retrieval augmented generation. *arXiv preprint arXiv:2405.02355*, 2024.

[13] Wei Liu, Ailun Yu, Daoguang Zan, Bo Shen, Wei Zhang, Haiyan Zhao, Zhi Jin, and Qianxiang Wang. Graphcoder: Enhancing repository-level code completion via coarse-to-fine retrieval based on code context graph. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 570–581, 2024.

[14] OpenAI. Openai text-embedding-3-large api documentation, 2024. `https://platform.openai.com/docs/models/text-embedding-3-large`.

[15] Shinya Takamaeda-Yamazaki. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In *International Symposium on Applied Reconfigurable Computing*, pages 451–460. Springer, 2015.