Stack2Graph: A Structured Knowledge Representation of Stack Overflow Data for Retrieval-based Question Answering

Lukas Amadeus Kleybolte, Viviana Ventura, Alessandra Zarcone
Technical University of Applied Science Augsburg
An der Hochschule 1, 86161 Augsburg, Germany
{lukas.kleybolte, viviana.ventura, alessandra.zarcone}@tha.de

Abstract

Community-based platforms like Stack Overflow (SO) offer a vast and diverse source of software development knowledge, combining natural language data with code snippets. Resources built from SO have been widely used to support downstream tasks in software engineering and natural language processing. However, no existing resource fully reconstructs and connects the complete range of information available on SO, leveraging its structure. We introduce Stack2Graph, a large-scale resource that preserves the forum's structural relationships in a semantically explicit form by combining a knowledge graph with a vector database. This hybrid design captures the intrinsic links between questions, answers, comments, tags, and cross-references, bridging symbolic and vector-based representations to enable structured and multi-hop retrieval. The goal is to make SO knowledge more efficiently accessible for LLM-based systems and easier to integrate into downstream applications. To evaluate its impact, we integrate Stack2Graph into a zero-shot pipeline for multiple-choice question answering on CodeMMLU. Results show that retrieval augmentation particularly benefits mid-sized general-purpose models, with substantial gains in API- and framework-oriented tasks.

Keywords: Stack Overflow, Knowledge Graph, Hybrid Embedding, Dataset, Question Answering, Information Retrieval

1. Introduction

Answering software-related questions – ranging from code implementation and debugging to conceptual, design, and configuration issues – remains challenging for natural language processing and software engineering research, as building automatic systems for answering such questions requires alignment between natural language, source code and the underlying reasoning context. Previous work (Hasan et al., 2021; Huang et al., 2021; Kocetkov et al., 2023; Li et al., 2024; Lozhkov et al., 2024) proposed pretraining Large Language Models (LLMs) with large collections of code and natural language data in order to align code and natural language for downstream tasks such as synthesizing code from natural language descriptions, code retrieval, documentation generation for function or retrieval-based Question Answering (QA).

Platforms such as Stack Overflow (SO)[1] represent major knowledge bases for information exchange among developers, where users seek and share solutions to specific programming problems. As the largest and most widely used knowledge-sharing platform for software developers, SO contains over 24 million questions (70% answered) and 36 million answers, covering a wide range of programming languages, frameworks, and problem-solving strategies. The complete SO unifies the discussion language to English. Its rich, diverse content makes it an invaluable foundation for building domain-specific QA and retrieval systems. On SO, knowledge is usually organized in question threads. A user posts a question, and one or more answers are provided in response. Many of these threads also include links to other SO posts or to external resources such as documentation or blogs. Together, these references form a hidden network of interconnected knowledge, where important information is dispersed across various locations.

SO has become a valuable resource for building datasets that combine natural language and code in the form of questions, answers, and comments (Yao et al., 2018; Wang et al., 2023; Li et al., 2024), supporting downstream tasks such as QA, and code retrieval, search and generation. Studies leveraging SO as a resource have focused on linking natural language to source code (Yin et al., 2018; Yao et al., 2018). Other lines of research have investigated how to improve information retrieval on SO itself, developing methods to better search, rank and recommend posts within the forum, or to better retrieve answers or code (Ahasanuzzaman et al., 2016; Nie et al., 2016; Liu et al., 2017, 2018; Xu et al., 2018b,a; Silva et al., 2019; Oliveira et al., 2024). However, most existing approaches treat each question-answer pair or each code snippet as two single, isolated piece of knowledge, disregarding the relational structure of SO, where questions, answers, comments, tags, votes, and cross-references form a densely interconnected network of developer knowledge. This

[1] https://stackoverflow.com

Proceedings of the Knowledge Graphs and Large Language Models Workshop 2026 (KG-LLM@LREC26), pages 120–133
May 16, 2026.

limitation prevents models from fully exploiting the platform's context and relationships.

We present Stack2Graph: a coding domain-specific dataset with a dual structure, a Knowledge Graph (KG) and a Vector Database (VD). The code for Stack2Graph is open source and available on GitHub[2]. The dataset encompasses over 117 million nodes and 275 million edges. Unlike previous SO-based datasets, Stack2Graph explicitly preserves the forum's intrinsic network structure within a coherent, unified, and semantically-interpretable vector-based KG. The two representations preserve the data structure of the forum in different ways: the KG captures relationships between questions, answers, tags, links, duplicates, comments, votes, and more, while the VD provides dense and sparse embeddings for semantic similarity, hybrid retrieval and context-aware reasoning. Together, these components offer a strong foundation for future programming-domain QA systems and for code understanding, enabling deeper exploration and reasoning over the collective knowledge of the developer community.

Stack2Graph complements existing efforts by introducing a dual representation and is thus uniquely placed between textual QA benchmarks and KG resources. Such a position in the dataset landscape makes it especially valuable for programming-focused QA tasks, where isolated question-answer pairs may be insufficient to address complex or multi-layered queries. We evaluate Stack2Graph by integrating it into a retrieval-augmented pipeline for coding-domain multiple-choice question answering. Across different configurations, results show that mid-sized general-purpose models benefit most consistently from structured retrieval, particularly in API- and framework-oriented tasks. Overall, our findings indicate that including structured community knowledge can improve retrieval-augmented reasoning in programming-related QA without requiring task-specific fine-tuning.

2. Related Work

KGs structure information through entities (nodes) and relations (edges), enabling reasoning over interconnected data. The Resource Description Framework (RDF)[3], originally standardized by the W3C [4], and its extension RDFS[5], provide the graph-based foundation for representing such information as triples. KG-based representations are particularly well-suited to capture the structured knowledge embedded in SO. Datasets such as StaQC (Yao et al., 2018), CoNALA (Yin et al., 2018), Repo4QA (Chen et al., 2022), CodeInsight (Beau and Crabbé, 2024), and ProCQA (Li et al., 2024) have been built by retrieving natural language questions, answers, descriptions and code from SO in order to enable code retrieval or generation, while SOSum (Kou et al., 2022) augments questions with manually added summarizations. Despite their utility, these resources extract content for task-specific learning and abstract away the relational dependencies among and inside posts. Consequently, they do not support structural reasoning, multi-hop navigation, or graph-based exploration across the broader SO ecosystem. Recent work has explored graph-based or relational perspectives on SO. MR^2-KG (Gong and Zhang, 2024) is a multi-relation, multi-rationale KG for SO, whose nodes represent questions, answers, and external webpages, and whose edges explicitly encode heterogeneous knowledge relations such as answer hierarchy, duplicate, containment, and working-example links. This graph is automatically constructed using a RoBERTa-based supervised classifier and is primarily evaluated for intent-aware answer generation and recommendation. Sahub (Oliveira et al., 2024) constructs a Neo4j graph that connects SO discussions with external open-source code repositories, using developer provided annotations. The mail goal is to align real-world code examples to specific technical questions. Liu et al. (2017) present KGQR, a KG-based framework for question-routing that models topics, users, and questions as entities and their interactions. They apply a TransR-style embedding model to recommend relevant experts for unanswered questions in community forums. Their graph is optimized for user recommendation rather than knowledge representation of SO content itself. Liu et al. (2023) presents KGXQR, a KG-based explainable question-retrieval approach that combines BERT-based sentence embeddings and concept-graph embeddings to train a relevance prediction model for reranking candidate SO questions. The KG contains 2,291,354 concepts and 4,220,946 relations and is built extracting concepts from SO posts and tags, enriching them with relations from Wikidata, WordNet, and other lexical resources. KGXQR follows a retrieve-and-rerank pipeline: BERT-based sentence embeddings are first used to retrieve candidate questions, and a relevance prediction model—combining sentence embeddings with concept-graph embeddings—is then applied for reranking. The KG operates at the concept level and is primarily used to bridge knowledge gaps and generate explanation paths between queries and candidate questions, rather than modeling structural relations among SO posts themselves. Others examine duplicate detection

[2] https://github.com/tha-atlas/Stack2Graph

[3] https://www.w3.org/TR/rdf11-concepts/

[4] https://www.w3.org/about/

[5] https://www.w3.org/TR/rdf-schema/

(Ahasanuzzaman et al., 2016) and question relatedness (Xu et al., 2018a), further revealing the latent network structure of SO. Other approaches, such as Post2Vec (Xu et al., 2022) and SOBERT (He et al., 2024), provide vector representations of SO posts, with the goal of relatedness prediction, post classification, and tag or API recommendation. Yet, these efforts remain locally limited to narrow tasks, specific relations, or small sub-graphs, and do not capture the platform's global interconnectivity or semantic hierarchy.

Stack2Graph differs fundamentally from prior SO-KG approaches in construction methodology, representation structure and size. Unlike MR^2-KG, which relies on supervised edge classification to model multi-rationale relations within individual threads, or Sahub, which links posts to external repositories via annotations, Stack2Graph reconstructs the full structural topology of SO directly from the official data dump, enabling faithful reconstruction of its global relational architecture at scale. Compared to KGQR, which models user–topic–question triplets for expert routing, and KGXQR, which builds a concept-level graph to enhance semantic retrieval, Stack2Graph operates directly at the post and interaction level, preserving the complete relational fabric of SO rather than abstracting it into task-specific embeddings or concept graphs.

3. Methodology

Stack2Graph is constructed through a multistage pipeline that transforms the raw SO data into two complementary representations: a structured KG and a VD. The KG captures the relational structure of SO posts and the VD provides semantic representations for efficient retrieval. The methodology is designed to ensure conceptual and structural consistency across both representations, while preserving the rich relational information of the original data. Both schemas are designed to be interconnected through shared identifiers, allowing hybrid use cases that combine explicit relations with semantic similarity.

3.1. Data Collection

The SO community offers monthly data dumps via Internet Archive[6], which contain the entire forum and metadata at a given point in time. The SO data dumps are a community-driven effort, as the official data export service was discontinued in 2024. For this work, we used the June 2025 dump as the basis of our dataset – roughly 333 GB of compressed XML data. The raw dumps are released in XML format and contain all questions, answers, comments, tags, user information, and post histories. For easier querying and processing, the raw XML files were first imported into a SQL database (MariaDB[7]). This intermediate step enabled efficient access to questions, answers, comments, votes, links, and tags, serving as the basis for constructing both the KG and the VD. The SO data is released under the Creative Commons Attribution-ShareAlike (CC BY-SA) license, with versions ranging from 2.5 to 4.0 depending on the contribution date. All versions allow reuse with attribution and require that modifications or derivative works are distributed under the same CC BY-SA license. Stack2Graph will be publicly released under the same license to foster further research in QA over programming-oriented knowledge representations.

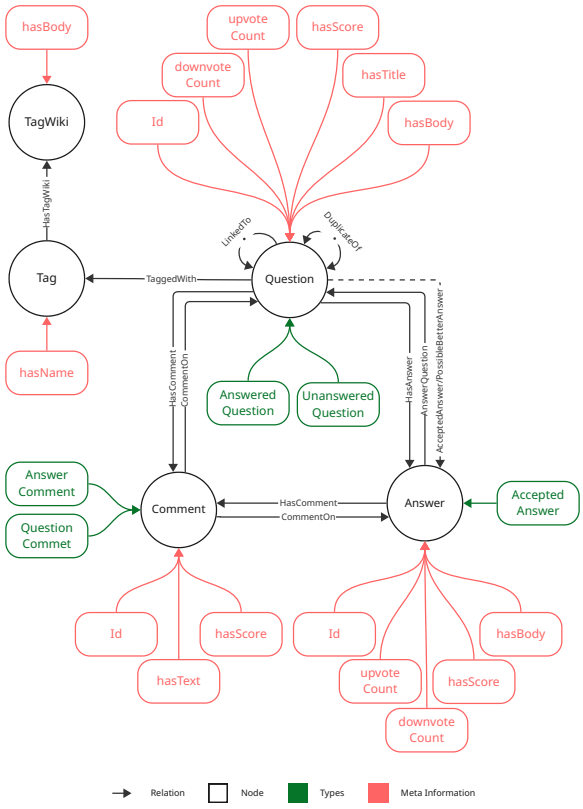

Figure 1: Overview of the SO KG schema.

3.2. Knowledge Graph

The main component of Stack2Graph is the KG, which represents SO posts and their relationships as structured entities and edges. For this work, only a subset of the available files of the data dump is used. Specifically, *Posts*, *PostLinks*, *Comments*, *Votes*, and *Tags* are included, while user-specific tables such as *Users* and *Badges* are not, as they are

[6] https://archive.org

[7] https://mariadb.org/

not relevant to the structural or semantic aspects of the dataset. Similarly, the *PostHistory* table is omitted since the focus is on the most recent version of each post rather than its revision history. After this filtering and transformation step, the original 335GB XML dataset is reduced to approximately 155GB of relational data. All identifier fields (`ID`) are indexed in MariaDB to accelerate join operations and multi-table queries.

After preparing the relational database, we transform the selected tables into RDF triples. The contents of the SQL tables are subsequently transformed into triples, formalized as described in the Resource Description Framework Schema (RDFS)[8], using the Python library RDFLib (Krech et al., 2025). All schema elements, such as question, answer, comment, and tag, are defined as RDF resources under a set of custom designated namespaces[9]. The base namespace `so` corresponds to `<http://stackoverflow.com/schema#>` and sets up the ontology of SO. Figure 1 illustrates the most important entity types and their relations. The graph contains concrete entity types such as `Question`, `Answer`, `Comment`, `Tag`, and `TagWiki`. In addition, we introduce two abstract superclasses, `Post` and `Content`, to provide a consistent hierarchy. For clarity, these superclasses are omitted from Figure 1. The superclass structure ensures that shared properties can be defined once and inherited by all relevant entities.

For example, all questions, answers, and tag wikis are instances of `so:Post`, and all posts and comments are instances of `so:Content`, which allows them to carry properties such as `so:score`. The core class hierarchy is therefore:

$$\begin{aligned} \texttt{so:Question} &\sqsubseteq \texttt{so:Post} \\ \texttt{so:Answer} &\sqsubseteq \texttt{so:Post} \\ \texttt{so:TagWiki} &\sqsubseteq \texttt{so:Post} \\ \texttt{so:Post} &\sqsubseteq \texttt{so:Content} \\ \texttt{so:Comment} &\sqsubseteq \texttt{so:Content} \end{aligned}$$

Edges in the graph represent relationships between entities. These include structural relations (e.g., `so:HasAnswer`, `so:HasComment`), tagging relations (`so:TaggedWith`, `so:HasTagWiki`), and cross-thread links (`so:DuplicateOf`, `so:LinkedTo`).

The `so:HasComment` edge can be directed from a question to a comment as well as from an answer to a comment. This relation type is distinct from the answer relation. Comments are usually shorter than answers and less structured, for example, they can not contain code blocks. Internal links that are explicitly specified in posts, such as duplicate markers or direct links, are also included to connect questions across threads. Cross-thread links explicitly connect related questions. The relation `so:DuplicateOf` is used when a question is marked as a duplicate of another. The relation `so:LinkedTo` represents manually inserted links indicating related content.

Each question is marked with one or more tags, with which it shares the relationship `so:TaggedWith`. In turn, the node tag has encyclopedic pages attached to it, which is why it has the `so:HasTagWiki` link. In addition to these base mappings, derived types such as `AnsweredQuestion`, `PossibleBetterAnswer`, and `AcceptedAnswer` are automatically inferred from the post metadata – particularly from the score values, vote statistics, and the accepted answer indicator[10]. The relation `PossibleBetterAnswer` is applied when an answer receives a higher score than the officially accepted one, indicating a strong likelihood that it provides a better solution. Each node also stores metadata inherited from the original SQL tables, such as post titles, bodies, up- and down-votes, scores, and other metadata. They are attached to the corresponding nodes as literal properties (e.g., `hasTitle`, `hasBody`, `score`).

In order to ensure interoperability with other resources, we linked our closed-world custom ontology to standard ontologies: SIOC[11], SKOS[12], Schema[13], DublinCore[14]. For example, the node `so:Post` can also be called using the label `sioc:Post`, and the relation `so:hasBody` is linked with the labels `sioc:content` and `schema:text`.

The generated triples are serialized in batches of 50,000 triples per file. Each file is stored in TRIG format and may contain multiple named graphs. Each named graph corresponds to a programming language namespace (e.g., `<http://stackoverflow.com/python>`). As files are created based on triple count rather than language boundaries, a single language graph may span multiple files, and a single file may contain multiple language graphs.

This layout supports streaming generation, incremental validation, and efficient bulk loading into graph databases such as Ontotext GraphDB[15], en-

[8] https://www.w3.org/TR/rdf-schema/

[9] In RDF terminology, a namespace is a URI prefix that groups related concepts and ensures that every class or property name is globally unique.

[10] The accepted answer is the answer marked as correct by the author of the post.

[11] http://sioc-project.org/

[12] https://www.w3.org/2004/02/skos/

[13] https://schema.org/

[14] http://dublincore.org/

[15] https://www.ontotext.com/products/gr

abling both standalone graph retrieval and hybrid strategies that combine explicit structural links with semantic similarity from the VD.

Within GraphDB, the KG is organized into programming-language-specific subgraphs rather than stored as a single unified graph. For this work, we selected the 39 most widely used programming languages on SO as the basis for constructing these sub-graphs.

At the same time, the boundaries between these sub-graphs are not strict. If a post links to or is marked as a duplicate of another post from a different programming language, this cross-language connection is also included in the graph. In this way, the dataset provides focused programming language-specific views as well as a broader, interconnected representation of programming knowledge.

3.3. Vector Database

In addition to the graph, we provide a VD that stores semantic representations of SO posts, organized into language-specific collections. While the KG schema captures explicit structural relations, the VD is designed for dense semantic retrieval. The VD is built by extracting questions for 39 programming languages from the KG via SPARQL.

To handle the variable length and technical nature of software discussions, we implement a parent-child indexing strategy. Raw HTML content is first preprocessed to remove noise (e.g., stack traces, minified code). The text is then tokenized and split into sliding window chunks (children) while maintaining a reference to the original full post (parent). We employ a hybrid embedding approach: dense vectors are generated using the nomic-embed-code model (Suresh et al., 2025) to capture semantic meaning, while sparse lexical vectors are computed using BGE-M3 (Chen et al., 2024) to retain keyword precision. The resulting embeddings are uploaded to Qdrant[16], an open-source VD and vector similarity search engine. Each programming language is assigned to its own collection, mirroring the language-based subdivision used in the KG.

Because all vector entries preserve their original graph identifiers, the VD and the KG can be jointly queried or aligned at retrieval time. This enables hybrid pipelines that combine explicit relational reasoning from the KG.

The KG preserves explicit structural relations for symbolic traversal and multi-hop reasoning, while the VD provides dense and sparse embeddings for semantic similarity search. This dual representation enables retrieval systems to jointly exploit structural connectivity and semantic proximity, making the forum significantly more navigable: users and downstream models can move beyond isolated QA pairs to explore related discussions, duplicate clusters, tag neighborhoods, and interaction paths in a unified retrieval pipeline. In this way, Stack2Graph transforms SO from a collection of loosely connected threads into a coherent, queryable KG.

aphdb/

[16] https://qdrant.tech/

Metric	Min	Med	Mean	Max
Ques.	1,787	112,820	433,291	2,531,840
Ans.	3,106	162,208	688,331	3,965,784
AccRat	28.3	36.5	37.3	52.1
Ans/Q	1.15	1.57	1.54	2.14
Com/Q	2.75	4.04	4.14	6.66

Table 1: Distributional statistics across all 39 programming language subgraphs. **Ques.**: number of questions; **Ans.**: number of answers; **AccRat**: percentage of answers marked as accepted; **Ans/Q**: mean number of answers per question; **Com/Q**: mean number of comments per question.

4. Descriptive Statistics

Stack2Graph comprises 275 million edges and 117 million nodes extracted from SO, forming a large-scale heterogeneous graph over questions, answers, comments, and tags. The dataset contains over 16 million question nodes connected to 26.8 million answers, including 9.1 million accepted answers and 1.4 million possible better answers. Approximately 2.1 million questions remain unanswered.

The graph encodes dense interaction structure. Comment relations account for 72.7 million edges, tag associations for 54.9 million edges, and cross-question links for 9 million edges. In total, 371K unique tags provide fine-grained topical categorization across programming domains. These statistics indicate that Stack2Graph captures both content and community validation signals at scale.

The dataset spans 39 programming languages organized into language-specific subgraphs. A detailed list of all the programming languages is provided in Appendix A. Activity follows a pronounced long-tail distribution, with large ecosystems coexisting alongside smaller, specialized communities. To quantify this variability, Table 1 reports distributional statistics across all 39 languages. The median language contains 112K questions, whereas the largest exceeds 2.5M, confirming substantial cross-language variability. Despite this structural skew, resolution behavior remains relatively stable: the median accepted-answer ratio is 36.5%, and questions receive on average 1.54 answers

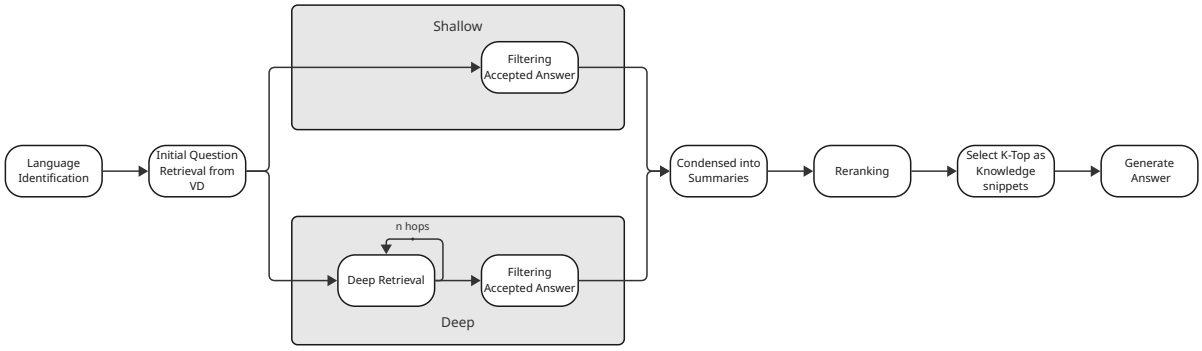

Figure 2: The multiple-choice QA pipeline including two different configurations for shallow and deep retrieval

and 4.14 comments. These findings suggest that Stack2Graph encodes heterogeneous ecosystem sizes while maintaining consistent patterns of interaction and community validation.

Overall, the dataset reflects both the structural complexity of SO's relational design and the behavioral diversity of its programming communities, providing a robust foundation for structured retrieval and multi-hop reasoning.

5. Experiments

To evaluate the potential of the proposed dataset Stack2Graph for multi-hop retrieval and reasoning over programming knowledge, we conducted a series of experiments on domain-specific multiple-choice question answering. The main objective was to assess whether combining the KG and the VD improves complex QA performance when using small LMs in a zero-shot setting.

5.1. Benchmark

We used the CodeMMLU benchmark (Manh et al., 2025), a domain-specific multiple-choice benchmark designed to evaluate LLMs on programming knowledge and reasoning across diverse computer science domains. From the full benchmark, we selected four subsets that reflect complementary types of software development knowledge: Database Management System (DBMS)/SQL, Software Principles, Programm Syntax, and API & Framework. These subsets jointly cover fundamental programming constructs, database reasoning, conceptual software design knowledge, and ecosystem-specific API usage, thereby enabling evaluation across both syntax-oriented and knowledge-intensive programming tasks.

DBMS/SQL: MySQL, PostgreSQL, and SQL, focusing on structured query reasoning and relational database concepts.

Software Principles: data structures, algorithms, computer architecture, system design, and software engineering, targeting higher-level conceptual and architectural reasoning.

Programm Syntax: C, C#, C++, Java, JavaScript, PHP, Python, R, Ruby, Matlab, HTML, CSS, and TypeScript, emphasizing language-specific constructs and code completion.

API & Framework: jQuery, Django, Pandas, NumPy, SciPy, Azure, Git, AWS, SVG, XML, Bootstrap, Node.js, AngularJS, React, and Vue, focusing on library- and framework-centric knowledge.

The multiple-choice tasks in all subsets consist of selecting the correct option to either complete a code snippet or to answer a conceptual programming question.

5.2. Experimental Pipeline

Each benchmark question was processed through a multi-stage retrieval and reasoning pipeline as visualized in Figure 2. The pipeline is designed to combine dense and sparse retrieval with structured graph expansion.

Given a benchmark question together with its answer options, the most relevant programming language is predicted, to determine the language-specific vector collections to be queried in the VD and the corresponding language-specific subgraph of the KG. The programming language is first identified using a lightweight rule-based component applying regular-expression heuristics to detect explicit language indicators (e.g., syntax patterns or language-specific keywords). If no language can be confidently determined, a language model is prompted with the full question and answer options to predict the most likely programming language.

We retrieve $4 \times k_{\text{top}}$ semantically related SO questions from the selected VD collection. Oversampling ensures sufficient coverage after filtering for accepted answers. Retrieval combines sparse lex-

ical matching and dense semantic embeddings derived from question titles and bodies (see Section 3.3). For each retrieved question candidate, all associated metadata are collected from the KG, including answers, comments, votes, and cross-references. This step reconstructs complete question–answer threads as structured bundles.

We consider two retrieval modes:

Shallow mode (hop=0): Only the initially retrieved questions with accepted answers are retained. No graph traversal is performed.

Deep mode (hop=1): From the initially retrieved questions, we traverse `so:DuplicateOf` and `so:LinkedTo` edges to include directly connected questions that also contain accepted answers.

Each question–answer–comment bundle can optionally be condensed into a summary of at most 256 tokens. The bundles (or summaries) are then reranked with the `bge-reranker-v2-m3` model (Chen et al., 2024).

A snippet is only passed to the LLM if it exceeds both the reranker score threshold (0.1) and the relevance gate (0.4). This filtering is applied uniformly in both shallow and deep mode. If no snippets pass the filters, the pipeline falls back to a pure zero-shot answer. The top k_{top} ranked snippets are finally concatenated and provided as context to the language model, which generates the final multiple-choice answer.

5.3. Retrieval Configurations

We compare five configurations:

Zero-Shot: No external knowledge; the model answers directly from the question.

Shallow (hop=0): Retrieval from the VD only, restricted to questions with accepted answers–with and without summarization.

Deep (hop=1): Retrieval augmented with graph-based expansion via duplicate and linked relations–with and without summarization. All configurations were evaluated without any task-specific fine-tuning of the language models, using Python 3.11.10. Model inference (language identification, summarization, and answer generation) was performed with vLLM (Kwon et al., 2023). For retrieval, we used the nomic-embed-code model to generate dense embeddings and bge-m3 for sparse embeddings and queried the Qdrant VD. Reranking was performed with bge-reranker-v2-m3. Graph queries were executed using SPARQL via the SPARQLWrapper library from the RDFlib project (Krech et al., 2025).

5.4. Results

Table 2 reports accuracy (%) for each model and dataset. For every model–dataset pair, we compare the zero-shot (ZS) baseline against the best-performing Stack2Graph-augmented configuration (S2G), selected among shallow (hop=0) and deep (hop=1) retrieval settings with and without summarization. The column Δ denotes the absolute improvement over the zero-shot baseline.

Across the evaluated models, Stack2Graph improves performance in a majority of configurations, although the magnitude of improvement varies depending on model size and domain. The strongest gains are observed for Qwen2.5 1.5B, with improvements of +2.7 on Software Principles, +5.0 on Programm Syntax, and +6.7 on DBMS SQL. On the API & Framework subset, the same model shows a substantial increase of +21.5 points.

Llama3 8B and Mistral 7B also exhibit consistent improvements in several subsets, particularly in Programm Syntax and DBMS SQL. In contrast, very small models such as Qwen2.5 0.5B show limited or negative changes across subsets. For code-specialized variants (Coder models), the impact of retrieval is heterogeneous: some configurations yield moderate gains (e.g., +2.8 on DBMS SQL for Qwen2.5 Coder 1.5B), while others show slight decreases.

A detailed breakdown of all configurations is provided in Appendix B. Summarization generally improves retrieval effectiveness compared to raw thread concatenation. Graph-based expansion (hop=1) provides additional gains in several small and mid-sized models but does not consistently outperform shallow retrieval across all settings.

6. Discussion

The results indicate that the Stack2Graph is not uniformly beneficial. Gains and losses are very close, which means that the effect of Stack2Graph is conditional rather than universal.

To enable a systematic and scalable qualitative interpretation of the heterogeneous retrieval effects, we performed a cluster analysis on the 9,257 CodeMMLU questions. Each unique question was embedded using an embedding model (Qwen3-Embedding-4B). The resulting representations were dimensionality-reduced via PCA and grouped into semantically coherent clusters using MiniBatchKMeans. These cluster assignments were then projected onto all model-question evaluation results. This methodology allows us to move beyond isolated error cases and instead aggregate improvements, degradations, and stable outcomes within groups of semantically similar questions. Consequently, it reveals for which question types, domains, and model classes retrieval augmentation consistently helps or harms performance. For a visual summary of how retrieval influences different question clusters, see Appendix C. The

Model	Software Principles			Programm Syntax			DBMS SQL			API & Framework		
	ZS	S2G	Δ	ZS	S2G	Δ	ZS	S2G	Δ	ZS	S2G	Δ
Llama3.2 1B	29.2	30.2†	+1.0	29.5	29.7†	+0.2	30.8	32.6†	+1.8	37.2	36.1†	-1.1
Llama3 8B	42.8	44.3*	+1.5	44.3	47.9*	+3.6	54.8	53.2†	-1.5	64.1	65.2*	+1.1
Qwen2.5 0.5B	32.3	31.9†	-0.4	32.5	31.4†	-1.1	36.0	34.2*	-1.8	40.2	38.1*	-2.1
Qwen2.5 1.5B	37.7	40.3*	**+2.7**	38.8	43.8*	**+5.0**	47.8	54.5†	**+6.7**	43.7	65.2*	**+21.5**
Qwen2.5 7B	63.1	62.9*	-0.1	61.2	60.1*	-1.1	66.3	67.6*	+1.3	83.7	82.0†	-1.7
Qwen2.5 Coder 0.5B	25.5	26.3†	+0.8	30.5	30.0†	-0.5	27.8	26.2†	-1.5	33.0	32.4*	-0.6
Qwen2.5 Coder 1.5B	41.8	40.6*	-1.2	45.2	43.3†	-1.9	41.4	44.2†	+2.8	53.9	56.3†	+2.4
Qwen2.5 Coder 7B	60.9	60.8†	-0.1	62.7	62.0†	-0.7	64.8	65.8†	+1.0	82.5	82.3†	-0.1
Qwen3 4B	62.9	62.4†	-0.5	64.0	63.6*	-0.3	66.8	69.7†	+2.8	80.5	79.9†	-0.6
Deepseek Coder 1.3B	23.4	25.1*	+1.6	27.4	27.4†	+0.1	27.5	28.3†	+0.8	27.0	27.2*	+0.3
Mistral 7B	34.5	36.3†	+1.9	39.9	43.2*	+3.3	43.4	48.8*	+5.4	47.2	60.6*	+13.4

† Shallow mode (hop=0) * Deep mode (hop=1)

Table 2: Accuracy by model and dataset. ZS = Zero-Shot, S2G = best approach from our retrieval experiments with Stack2Graph, Δ = S2G - ZS. Bold Δ marks the biggest improvement per dataset.

analysis reveals clear patterns. The effectiveness of Stack2Graph depends strongly on different aspects, among which, as already mentioned, the model size. Mid-sized general-purpose models (1.5B–4B parameters) benefit most consistently from retrieval augmentation, as structured external knowledge effectively compensates for limited parametric capacity. In contrast, code-specialized models exhibit smaller or inconsistent gains, suggesting that their strong parametric coding knowledge already covers a substantial portion of the benchmark distribution, thereby reducing the marginal benefit of external retrieval. In some cases, additional retrieved context even introduces noise or distracts from well-established internal representations. Smaller models (<1.5B) show the highest variability: they profit strongly when high-quality snippets are retrieved but are also most susceptible to noise. They often lack the reasoning capacity to use retrieved evidence effectively, so even when the context is relevant, they do not reliably convert it into the correct answer. Larger models (≥7B) already achieve high zero-shot performance and therefore display smaller marginal gains. Summarization helps mitigate noise from long discussion threads, especially for smaller models with limited context integration capacity.

A second important source of variability in the results is the content of the question. Retrieval augmentation is most effective for definition or concept questions, where stable external knowledge proves to be most useful. That is why API & Framework questions and many software-principles questions benefit from the KG: these items often ask for definitions, tool usage, framework conventions, or best practices, where retrieved SO context can provide exactly the missing fact. Here, the KG frequently supplies concrete, community-validated examples that pure vector retrieval misses. A representative case is the question "How can you drop rows with empty cells in a DataFrame?" from the API & Framework subset. A mid-sized Qwen2.5-1.5B model answered incorrectly in the zero-shot setting. After hop-1 retrieval, the graph expansion via `so:DuplicateOf` surfaced a duplicate thread containing the exact accepted answer (`dropna()`), which the model then used correctly. This illustrates how the structural relations in Stack2Graph can provide complementary information that vector search alone cannot surface. However, the same hop-1 mechanism can also introduce irrelevant or misleading snippets when relations are only loosely aligned with the question. Reasoning or calculation questions, such as binary-tree node counts, shortest-path, syntax-sensitive code execution and formal DBMS questions, benefit the less and sometimes are even harm from the retrieval, since internal reasoning dominates and noisy snippets can mislead. Bad results are achieved by questions where the answer depends on fine-grained execution semantics rather than general knowledge. These include C/C++ pointer arithmetic, macro expansion, Java overloading, Python edge cases, and PHP/JavaScript output prediction. Here, retrieval can easily fetch examples that are topically similar but semantically mismatched. That mismatch is especially harmful because many benchmark questions hinge on one tiny detail: a cast, array indexing, or undefined behavior. Retrieval that is "approximately relevant" is not good enough here. These questions often require exact parsing of language semantics, operator precedence, type conversion, pointer behavior, macro expansion, or execution order. In such cases, retrieved context can help when it is highly aligned, but it can just as easily introduce noise. The same logic explains the

DBMS/SQL behavior. Even though retrieval can surface explicit rules, SQL is especially vulnerable to ambiguity because syntax and behavior vary by dialect, version, and implementation details. Retrieved snippets may therefore be factually correct in a local context but still wrong for the benchmark question. That makes DBMS retrieval less robust than API/framework retrieval, where the target fact is often clearer and more directly documented.

A further analysis reveals that even after hop-1 expansion, only 36.6%–59.9% of questions actually receive at least one valid knowledge snippet, as shown in Table 3. The remaining cases fall back to pure zero-shot. Hop-1 consistently increases coverage by 12–14 percentage points compared to hop-0, confirming that the graph relations provide additional relevant threads, using he DuplicateOf and LinkedTo relations. However, Hop-1 yields more improvements than Hop-0, but also more worsening. Deeper graph traversal increases variance rather than producing a clean gain, in other words, Hop-1 increases recall, but also semantic drift. Also, substantial variance exists across models: small models ($\leq$1.5B) achieve markedly lower coverage (often only 20–40% even at hop-1), while mid-sized and larger models frequently reach 55–66%.

Table 3 shows the actual retrieval coverage after confidence filtering (aggregated across all models).

Task	Hop=0	Hop=1	ZS Fallback
API	41.7%	53.8%	52.2%
DBMS	46.1%	59.9%	47.0%
Syntax	44.1%	57.5%	49.2%
Principles	25.9%	36.6%	68.8%

Table 3: Percentage of questions with knowledge snippets after confidence filtering (aggregated across all models)

A potential source of performance variability lies in the language identification stage of the pipeline, which is based on a rule-based component, while an LLM-based prediction intervenes if no match is found. Misclassification at this stage directly affects both the selected vector collection and the language-specific subgraph used for retrieval. This is particularly challenging for subsets such as Software Principles, where questions often describe abstract algorithmic or architectural concepts that are not tied to a single programming language.

7. Conclusion

We introduced Stack2Graph, a large-scale resource that preserves SO's relational structure through a KG and complements it with a VD for semantic retrieval. This hybrid representation enables structured, multi-hop access to programming knowledge.

Experiments on CodeMMLU demonstrate that integrating Stack2Graph into a zero-shot retrieval pipeline improves performance across multiple programming domains, particularly for mid-sized general-purpose models and knowledge-intensive tasks such as API & framework usage. These results confirm that preserving and exploiting the structural relations of community knowledge provides measurable benefits for retrieval-augmented reasoning in programming-related QA.

Although our results on CodeMMLU are inconsistent, our analysis has shown that these inconsistencies are not random but stem from factors related to the architecture of our retrieval systems. Furthermore, certain indicators – such as the significant improvements to the API & framework – show that, when used correctly, KG can be truly useful. This leaves room for future improvements.

7.1. Limitations

Beyond the experimental findings discussed above, we identify several structural limitations and promising directions for future work.

First, the current retrieval strategy performs a static hop-1 expansion. While this occasionally yields highly relevant community threads, it remains partly "luck-based" – the correct answer is not guaranteed to be surfaced. Future work will therefore introduce a dynamic, confidence-aware retrieval mechanism. A reranker score threshold combined with an LLM-as-Judge step will decide adaptively whether and how deeply the KG is traversed, ensuring that only high-confidence, non-misleading snippets are passed to the model.

Second, the dataset represents a static snapshot of SO (June 2025). Programming ecosystems evolve rapidly, particularly in API & framework domains. Future work could incorporate temporal modeling to enable time-aware retrieval and analysis of how programming knowledge changes over time.

Third, although Stack2Graph combines symbolic structure with dense and sparse retrieval, the integration remains pipeline-based. The traversal depth, relation selection, and reranking steps are not jointly optimized. A promising direction is the development of learned hybrid retrieval to adaptively combine graph exploration and semantic similarity.

Fourth, while we restrict retrieval to accepted answers, community-generated content varies in quality. Incorporating credibility estimation, vote-weighted scoring, or temporal decay mechanisms could further improve robustness.

Fifth, the KG encodes rich structural and social signals – including vote counts, answer scores, comment interactions, and external references –

yet the current strategy uses only accepted answers and limited structural relations for graph expansion. Future work could investigate score-aware ranking, comment-level retrieval, relation weighting, or agent-based retrieval to dynamically explore the graph and selectively aggregate evidence.

Finally, the evaluation is limited to multiple-choice question answering in a zero-shot setting. Strong validation on open-ended code generation, debugging, and interactive developer assistance scenarios is necessary to fully assess the utility of Stack2Graph in realistic programming workflows.

8. Bibliographical References

Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. 2016. Mining duplicate questions in stack overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 402–412, New York, NY, USA. ACM.

Nathanaël Beau and Benoit Crabbé. 2024. CodeInsight: A Curated Dataset of Practical Coding Solutions from Stack Overflow. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 5935–5947.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2318–2335, Stroudsburg, PA, USA. Association for Computational Linguistics.

Minyu Chen, Guoqiang Li, Chen Ma, Jingyang Li, and Hongfei Fu. 2022. Repo4QA: Answering Coding Questions via Dense Retrieval on GitHub Repositories. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1580–1592, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Lina Gong and Haoxiang Zhang. 2024. MR2-KG: A Multi-Relation Multi-Rationale Knowledge Graph for Modeling Software Engineering Knowledge on Stack Overflow. *IEEE Transactions on Software Engineering*, 50(7):1867–1887.

Masum Hasan, Tanveer Muttaqueen, Abdullah Al Ishtiaq, Kazi Sajeed Mehrab, Md Mahim Anjum Haque, Tahmid Hasan, Wasi Uddin Ahmad, Anindya Iqbal, and Rifat Shahriyar. 2021. CoDesc: A Large Code-Description Parallel Dataset. *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 210–218.

Junda He, Xin Zhou, Bowen Xu, Ting Zhang, Kisub Kim, Zhou Yang, Ferdian Thung, Ivana Clairine Irsan, and David Lo. 2024. Representation Learning for Stack Overflow Posts: How Far Are We? *ACM Transactions on Software Engineering and Methodology*, 33(3):1–24.

Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. 2021. CoSQA: 20,000+ web queries for code search and question answering. *ACL-IJCNLP 2021 - 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, 1:5690–5700.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2023. The Stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research*, 2023.

Bonan Kou, Yifeng Di, Muhao Chen, and Tianyi Zhang. 2022. SOSum: a dataset of stack overflow post summaries. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 247–251, New York, NY, USA. ACM.

Daniel Krech, Gunnar Aastrand Grimnes, Graham Higgins, Nicholas Car, Jörn Hees, Iwan Aucamp, Niklas Lindström, Natanael Arndt, Ashley Sommer, Edmond Chuc, Ivan Herman, Alex Nelson, Jamie McCusker, Tom Gillespie, Thomas Kluyver, Florian Ludwig, Pierre-Antoine Champin, Mark Watts, Urs Holzer, Ed Summers, Whit Morriss, Donny Winston, Drew Perttula, Filip Kovacevic, Remi Chateauneu, Harold Solbrig, Benjamin Cogrel, and Veyndan Stuart. 2025. RDFLib.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, New York, NY, USA. ACM.

Zehan Li, Jianfei Zhang, Chuantao Yin, Yuanxin Ouyang, and Wenge Rong. 2024. ProCQA: A Large-scale Community-based Programming Question Answering Dataset for Code Search.

In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 13057–13067, Torino, Italia. ELRA and ICCL.

Mingwei Liu, Simin Yu, Xin Peng, Xueying Du, Tianyong Yang, Huanjun Xu, and Gaoyang Zhang. 2023. Knowledge Graph based Explainable Question Retrieval for Programming Tasks. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 123–135. IEEE.

Xueqing Liu, Chi Wang, Yue Leng, and ChengXiang Zhai. 2018. LinkSO: a dataset for learning to retrieve similar question answer pairs on software development forums. In *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering*, pages 2–5, New York, NY, USA. ACM.

Zhu Liu, Kan Li, and Dacheng Qu. 2017. Knowledge graph based question routing for community question answering. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10638 LNCS:721–730.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation.

Dung Nguyen Manh, Thang Phan Chau, Nam Le Hai, Thong T. Doan, Nam V. Nguyen, Quang Pham, and Nghi D.Q. Bui. 2025. Codemmlu: a Multi-Task Benchmark for Assessing Code Understanding & Reasoning Capabilities of Codellms. *13th International Conference on Learning Representations, ICLR 2025*, pages 24838–24896.

Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query Expansion Based on Crowd Knowledge for Code Search. *IEEE Transactions on Services Computing*, 9(5):771–783.

André Oliveira, Paulo Matos, and Pedro Filipe Oliveira. 2024. Sahub - Stackoverflow and Comments Integrations. In *2024 International Conference on Engineering and Emerging Technologies (ICEET)*, 2024, pages 1–7. IEEE.

Rodrigo F.G. Silva, Chanchal K. Roy, Mohammad Masudur Rahman, Kevin A. Schneider, Klerisson Paixao, and Marcelo de Almeida Maia. 2019. Recommending Comprehensive Solutions for Programming Tasks by Mining Crowd Knowledge. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, volume 2019-May, pages 358–368. IEEE.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. 2025. CoRNStack: High-Quality Contrastive Data for Better Code Retrieval and Reranking.

Zhiruo Wang, Grace Cuenca, Shuyan Zhou, Frank F. Xu, and Graham Neubig. 2023. MCoNaLa: A Benchmark for Code Generation from Multiple Natural Languages. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 265–273, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bowen Xu, Thong Hoang, Abhishek Sharma, Chengran Yang, Xin Xia, and David Lo. 2022. Post2Vec: Learning Distributed Representations of Stack Overflow Posts. *IEEE Transactions on Software Engineering*, 48(9):3423–3441.

Bowen Xu, Amirreza Shirani, David Lo, and Mohammad Amin Alipour. 2018a. Prediction of relatedness in stack overflow: deep learning vs. SVM. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, New York, NY, USA. ACM.

Steven Xu, Andrew Bennett, Doris Hoogeveen, Jey Han Lau, and Timothy Baldwin. 2018b. Preferred Answer Selection in Stack Overflow: Better Text Representations ... and Metadata, Metadata, Metadata. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*, pages 137–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, pages 1693–1703, New York, New York, USA. ACM Press.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 476–486, New York, NY, USA. ACM.

A. Selected Programming Languages

The dataset is organized into 39 programming language-specific sub-graphs. Table 4 lists all selected languages that form the basis of the dataset. The selection is based on the most widely used and active tags on SO at the time of the June 2025 data dump.

Language	Language
ABAP	Assembly
Bash	C
C#	C++
COBOL	CSS
Dart	Elixir
Erlang	F#
Fortran	Go
Haskell	HTML
Java	JavaScript
Julia	Kotlin
Lisp	Lua
MATLAB	.NET
Objective-C	Perl
PHP	Powershell
Prolog	Python
R	Ruby
Rust	Scala
Shell	SQL
Swift	TypeScript
VB.NET	

Table 4: The 39 programming languages selected for dataset construction.

B. Full Retrieval Configuration Results

Table B provides a detailed breakdown of all evaluated retrieval configurations across datasets and models. In contrast to Table 5.2, which reports only the best-performing retrieval setting (BestRAG), this table exposes the individual contributions of shallow retrieval (hop=0), graph-augmented retrieval (hop=1), and optional summarization.

C. Cluster-Level Analysis of Retrieval Effects

Each point in figure 3 represents a cluster of questions; its position on the horizontal axis denotes the net effect of retrieval (improved minus worsened cases) and the vertical axis measures the proportion of questions that remained wrong despite retrieval. Bubble size reflects the number of unique questions in the cluster, and color encodes the dominant dataset (API & Frameworks, DBMS/SQL, Program Syntax, or Software Principles). The distribution shows that approximately half of the 100 clusters lie to the right of the zero line (net improvement) and half to the left (net degradation), indicating that retrieval augmentation has heterogeneous effects rather than uniformly improving performance.

Clusters dominated by Software Principles exhibit the strongest positive trend: clusters focused on conceptual topics or algorithmic reasoning (e.g., trees and graphs) achieve a positive mean net effect and often benefit from retrieval because external snippets supply relevant explanations. In contrast, Program-Syntax clusters display a slight negative mean effect: large clusters centered on C++ syntax or HTML/CSS (e.g., C7, C36) suffer when retrieved context introduces noise or distracts from the symbolic reasoning required. API & Framework clusters show a mild positive shift, with React/Vue-centered clusters (e.g., C66, C59) benefiting most from retrieval because factoid questions are well-supported by community documentation. DBMS/SQL clusters are few but tend to be negative; SQL questions are prone to dialect differences and outdated solutions, so external snippets sometimes mislead more than they help. The correlation between the net effect and the proportion of unanswered questions is weak, underscoring that retrieval success hinges on question type and the quality of retrieved evidence rather than baseline difficulty. Overall, the analysis confirms that retrieval augmentation can substantially assist in concept- or fact-oriented domains, but provides little or even negative benefit for syntax-heavy or dialect-sensitive tasks.

	Model	ZS	Hop 0		Hop 1		Δ
			No Sum	Sum	No Sum	Sum	
Software Principles	Llama3.2 1B	29.2	29.8	**30.2**	29.9	29.0	+1.0
	Llama3 8B	42.8	40.3	42.6	39.9	**44.3**	+1.5
	Qwen2.5 0.5B	32.3	31.1	**31.9**	30.3	30.7	-0.4
	Qwen2.5 1.5B	37.7	37.9	39.2	37.9	**40.3**	**+2.7**
	Qwen2.5 7B	63.1	62.8	62.1	**62.9**	62.0	-0.1
	Qwen2.5 Coder 0.5B	25.5	26.2	**26.3**	25.4	26.0	+0.8
	Qwen2.5 Coder 1.5B	41.8	40.4	40.5	39.3	**40.6**	-1.2
	Qwen2.5 Coder 7B	60.9	60.2	**60.8**	59.3	60.3	-0.1
	Qwen3 4B	62.9	61.5	**62.4**	60.9	61.4	-0.5
	Deepseek Coder 1.3B	23.4	23.8	24.2	25.0	**25.1**	+1.6
	Mistral 7B	34.5	34.9	**36.3**	34.5	35.9	+1.9
Programming Syntax	Llama3.2 1B	29.5	28.9	**29.7**	28.7	29.5	+0.2
	Llama3 8B	44.3	42.5	47.0	40.6	**47.9**	+3.6
	Qwen2.5 0.5B	32.5	30.5	**31.4**	30.2	31.2	-1.1
	Qwen2.5 1.5B	38.8	42.5	42.4	43.1	**43.8**	**+5.0**
	Qwen2.5 7B	61.2	59.8	60.0	60.0	**60.1**	-1.1
	Qwen2.5 Coder 0.5B	30.5	29.9	30.0	29.8	**30.0**	-0.5
	Qwen2.5 Coder 1.5B	45.2	42.3	**43.3**	41.4	43.1	-1.9
	Qwen2.5 Coder 7B	62.7	60.8	**62.0**	60.9	61.4	-0.7
	Qwen3 4B	64.0	63.5	63.5	62.6	**63.6**	-0.3
	Deepseek Coder 1.3B	27.4	26.6	**27.4**	25.6	26.7	+0.1
	Mistral 7B	39.9	39.9	42.7	38.6	**43.2**	+3.3
DBMS/SQL	Llama3.2 1B	30.8	29.8	**32.6**	30.6	30.8	+1.8
	Llama3 8B	54.8	49.4	**53.2**	40.6	51.9	-1.5
	Qwen2.5 0.5B	36.0	30.3	33.9	26.5	**34.2**	-1.8
	Qwen2.5 1.5B	47.8	53.0	**54.5**	51.9	53.7	**+6.7**
	Qwen2.5 7B	66.3	66.6	67.1	64.3	**67.6**	+1.3
	Qwen2.5 Coder 0.5B	27.8	**26.2**	25.7	**26.2**	24.9	-1.5
	Qwen2.5 Coder 1.5B	41.4	40.1	**44.2**	43.7	41.6	+2.8
	Qwen2.5 Coder 7B	64.8	**65.8**	64.0	63.5	64.0	+1.0
	Qwen3 4B	66.8	68.1	**69.7**	65.6	68.1	+2.8
	Deepseek Coder 1.3B	27.5	27.2	**28.3**	25.4	25.4	+0.8
	Mistral 7B	43.4	45.8	48.3	41.6	**48.8**	+5.4
API & Frameworks	Llama3.2 1B	37.2	35.4	**36.1**	33.2	33.0	-1.1
	Llama3 8B	64.1	57.2	64.9	55.6	**65.2**	+1.1
	Qwen2.5 0.5B	40.2	37.1	37.2	36.4	**38.1**	-2.1
	Qwen2.5 1.5B	43.7	57.6	60.5	63.1	**65.2**	**+21.5**
	Qwen2.5 7B	83.7	81.6	**82.0**	79.3	79.7	-1.7
	Qwen2.5 Coder 0.5B	33.0	31.4	32.2	30.0	**32.4**	-0.6
	Qwen2.5 Coder 1.5B	53.9	50.5	**56.3**	49.4	53.6	+2.4
	Qwen2.5 Coder 7B	82.5	80.9	**82.3**	80.6	81.6	-0.1
	Qwen3 4B	80.5	**79.9**	78.9	78.9	79.5	-0.6
	Deepseek Coder 1.3B	27.0	26.7	27.0	**27.2**	**27.2**	+0.3
	Mistral 7B	47.2	51.6	55.3	53.4	**60.6**	+13.4

Table 5: Full configuration table (% accuracy). raw = no summarization, sum = summarization enabled. S2G is the best approach from our retrieval experiments. Best values are bolded per row, and Δ marks the biggest improvement per dataset.

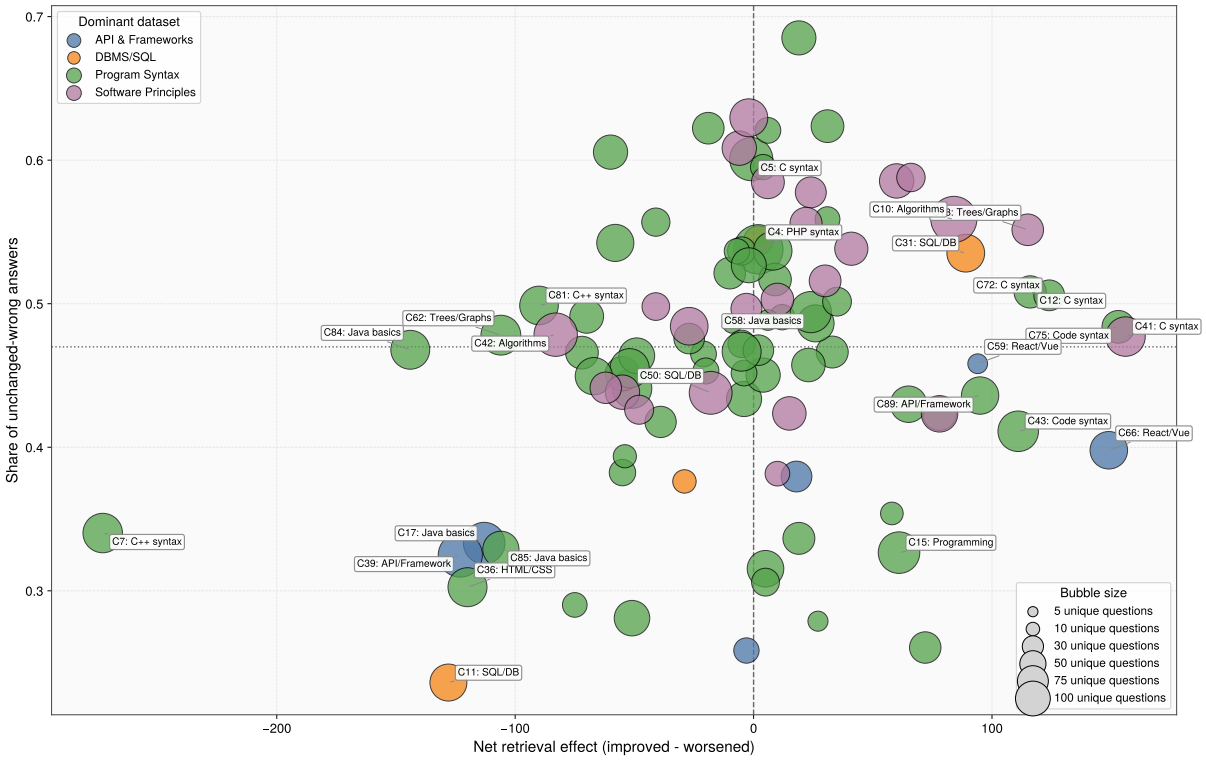

Figure 3: Cluster–level retrieval effects across question groups.