SubGCache: Accelerating Graph-based RAG with Subgraph-level KV Cache

Qiuyu Zhu[1], **Liang Zhang**[2*], **Qianxiong Xu**[1], **Cheng Long**[1*], **Jie Zhang**[1]

[1]Nanyang Technological University
[2]Hong Kong University of Science and Technology (Guangzhou)
{qiuyu002, qianxion001}@e.ntu.edu.sg, liangzhang@hkust-gz.edu.cn, {c.long, zhangj}@ntu.edu.sg

Abstract

Graph-based retrieval-augmented generation (RAG) enables large language models (LLMs) to incorporate structured knowledge via graph retrieval as contextual input, enhancing more accurate and context-aware reasoning. We observe that for different queries, it could retrieve similar subgraphs as prompts, and thus we propose SubGCache, which aims to reduce inference latency by reusing computation across queries with similar structural prompts (i.e., subgraphs). Specifically, SubGCache clusters queries based on subgraph embeddings, constructs a representative subgraph for each cluster, and pre-computes the key-value (KV) cache of the representative subgraph. For each query with its retrieved subgraph within a cluster, it reuses the pre-computed KV cache of the representative subgraph of the cluster without computing the KV tensors again for saving computation. Extensive experiments on three datasets across multiple LLM backbones and graph-based RAG frameworks demonstrate that SubGCache consistently reduces inference latency with comparable and even improved generation quality, achieving up to 6.68× reduction in time-to-first-token (TTFT).

Extended version — https://github.com/qiuyu111/appendix

Introduction

Retrieval-augmented generation (RAG) (Borgeaud et al. 2022; Lewis et al. 2020; Ram et al. 2023; Trivedi et al. 2023) enhances large language models (LLMs) (Achiam et al. 2023; Chowdhery et al. 2023; Dubey et al. 2024) by retrieving and integrating external knowledge based on text similarity, enabling more accurate and contextually enriched generation. Building on its success in language-focused tasks (Zhang, Haddow, and Birch 2023; Zhang et al. 2024b), recent efforts (Guo et al. 2024; He et al. 2024; Hu et al. 2025) have extended RAG to graph data (Jin et al. 2024a; Li et al. 2024; Wang et al. 2023), giving rise to graph-based RAG (He et al. 2024; Hu et al. 2025), which leverages textual graphs as external knowledge sources to help model entity relations across documents and support complex reasoning over structured knowledge. As illustrated in Figure 1(a), upon receiving a user query q_k and a textual graph G, graph-based RAG first retrieves the most relevant subgraph from G and constructs a subgraph prompt. This prompt is then combined with the query to form an augmented input for the LLM to generate the final response.

While proven effective, existing graph-based RAG systems are primarily designed for single-query settings, where each query is processed independently by the LLM, as shown in Figure 1(a). However, in many real-world scenarios (Ding et al. 2011; Choudhury et al. 2018; Zhang et al. 2024a; Bouros et al. 2024), such as scholarly question answering over academic graphs (Sinha et al. 2015), queries are batch-submitted, arrive in large volumes simultaneously, and are processed jointly, naturally forming in-batch workloads for graph-based RAG. Figure 1(b) illustrates a typical in-batch scenario, where a group of queries q_1, q_2, and q_3 are submitted and processed together. Each query triggers the retrieval of a relevant subgraph from the external textual graph, resulting in subgraphs s_1, s_2, and s_3. In practice, these retrieved subgraphs may exhibit significant overlap. For instance, we can observe that s_1 and s_2 are identical, while s_3 shares large structural components with them. Despite such redundancy, existing methods process each query in isolation, repeatedly encoding and reasoning over the overlapping subgraph content, leading to unnecessary computation. These observations call for a rethinking of graph-based RAG in a new in-batch setting and raise a natural question: how can we effectively exploit structural redundancy across different queries to eliminate redundant computation and improve overall system efficiency?

An intuitive answer to this question is to introduce a caching mechanism that stores and reuses previously computed results from the LLM to avoid repeated computation. In fact, recent efforts (Gim et al. 2024; Jin et al. 2025; Zheng et al. 2024) have explored similar strategies in purely textual settings, where each cached unit corresponds to an independent sentence or document chunk. For instance, Prompt Cache (Gim et al. 2024) stores the pre-computed attention states of frequently occurring text segments, improving efficiency through inference-time reuse. However, these approaches are inherently limited to sequential text data and assume exact lexical repetition. They are not applicable to graph-based RAG, where redundancy manifests at the structural level, and each cached unit should be a structured subgraph composed of interconnected nodes and edges, with

*Corresponding author.

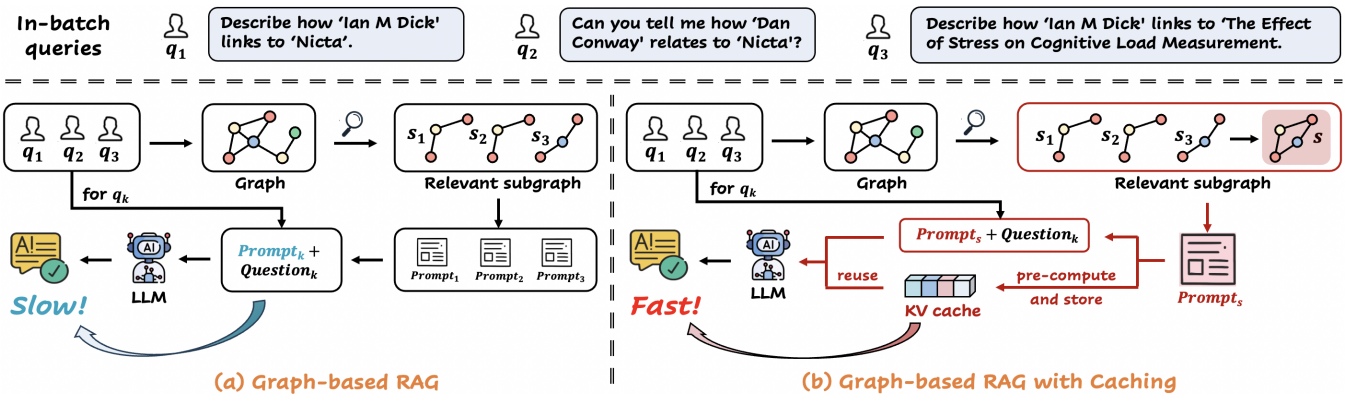


Figure 1: Overview of graph-based RAG without and with caching.

information organized topologically rather than sequentially. This structural nature of graph-based RAG introduces two critical and unique challenges:

- **Challenge 1: Structural redundancy identification.** In-batch queries may retrieve subgraphs that are structurally and semantically similar, but such overlap is neither explicitly known beforehand nor easily detectable. Here, the key challenge lies in effectively comparing retrieved subgraphs, which may differ in node identifiers, local context, or graph topology, to determine whether meaningful overlap exists.
- **Challenge 2: Structural redundancy exploitation.** Even when overlap is correctly identified across queries, the retrieved subgraphs are generally partially shared. Unlike existing methods for sequential text (Gim et al. 2024; Jin et al. 2025; Zheng et al. 2024), which assume reuse over identical units, overlapping subgraphs may differ in size, topology, or node alignment. Here, another key challenge is to effectively reason over these partially shared structures across queries to reduce redundant computation, while still preserving useful relational context necessary for accurate response generation.

To tackle these, we propose SubGCache (**Subg**raph-level key-value **Cache**), a lightweight and efficient plug-and-play caching framework for graph-based RAG under the in-batch query setting. It comprises two main components:

- **Design 1: Query clustering based on subgraph similarity.** SubGCache performs hierarchical clustering to in-batch queries based on the embeddings of their retrieved subgraphs, generated by the pretrained Graph Neural Network (GNN) (Zhu et al. 2025a; Wu et al. 2020) encoder used in graph-based RAG. These embeddings encode both semantic and structural information, allowing the system to automatically identify subgraph-level redundancy across queries. Queries with highly overlapping subgraphs are then effectively grouped together for shared processing, thereby addressing the challenge of structural redundancy identification.
- **Design 2: Representative subgraph construction and subgraph-level cache reuse.** To facilitate effective reasoning over partially overlapping subgraphs while preserving the useful relational context, SubGCache introduces the concept of representative subgraph as shared structural input for each query cluster. Specifically, for each cluster, it merges the retrieved subgraphs from all queries within this cluster into a single representative subgraph that preserves the topology necessary for accurate response generation. To exploit this shared structure and eliminate redundant computation, the key-value (KV) cache mechanism is further introduced to pre-compute KV tensors of the representative subgraph and reuse them across all queries in the cluster. This cluster-wise strategy addresses the challenge of reusing partial structural overlaps by aligning similar subgraphs into a unified representation and caching its computation. As illustrated in Figure 1(b), assume queries q_1, q_2, and q_3 are clustered together. SubGCache generates a representative subgraph s by merging their retrieved subgraphs s_1, s_2, and s_3, constructs the corresponding prompt prefix $Prompt_s$, and computes its KV tensors within the LLM, which are then stored in GPU memory. For each query $q_k \in \{q_1, q_2, q_3\}$, SubGCache directly appends the query-specific question tokens to the cached prefix, allowing the model to bypass recomputation of the shared subgraph context. By reusing the newly proposed subgraph-level KV cache across all queries in the cluster, SubGCache significantly reduces inference latency while maintaining strong generation quality.

Experiments across datasets and LLM backbones validate the latency reduction and generation quality of SubGCache. Our main contributions are summarized below:

- **Conceptually:** We formulate a new research problem under the in-batch query setting, aiming to accelerate graph-based RAG via batch-level processing. To the best of our knowledge, this is the first work to accelerate graph-based RAG and explore batch-level execution in this context.
- **Methodologically:** We propose a lightweight and plug-and-play framework SubGCache for subgraph-level prompt caching that addresses the unique challenges of structural redundancy identification and exploitation in retrieved subgraphs. It is simple to implement, and both highly effective and efficient in practice. Notably, this is also the first attempt to introduce prompt caching into

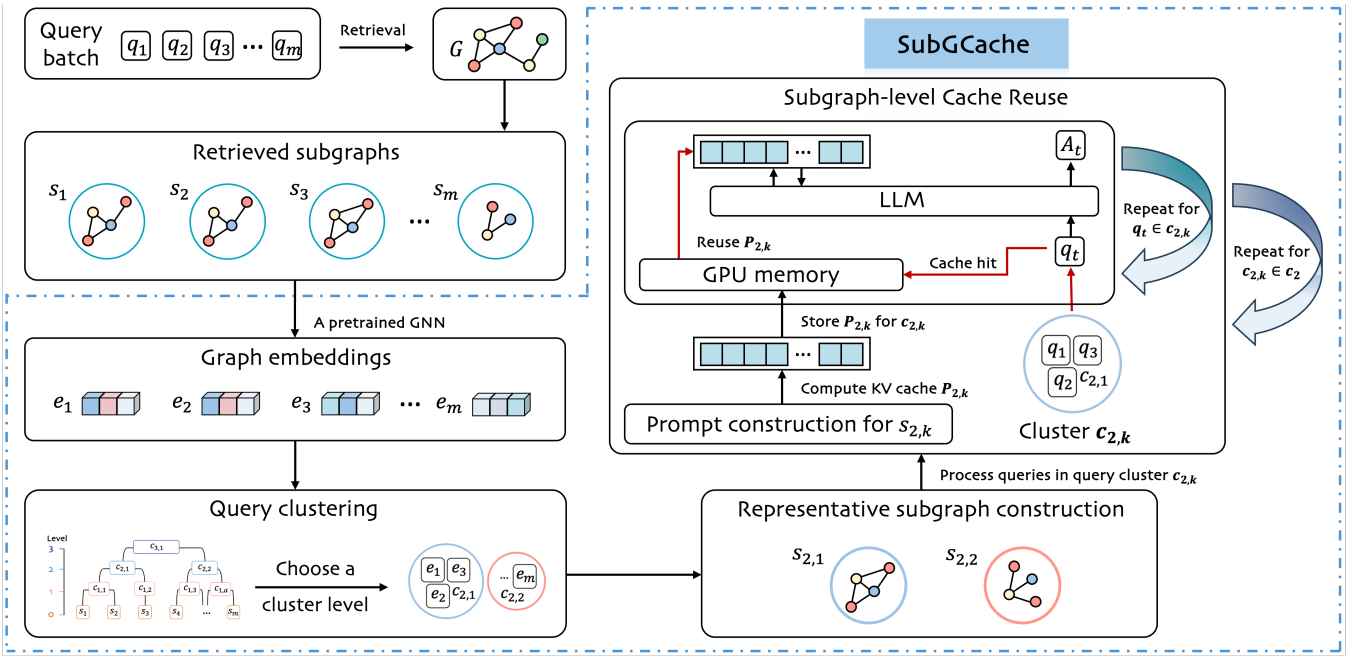


Figure 2: Overview of SubGCache and its integration into the standard graph-based RAG pipeline.

graph-based RAG.

- **Empirically:** Experiments on three datasets across multiple LLM backbones and graph-based RAG frameworks demonstrate that SubGCache consistently reduces inference latency while maintaining or even enhancing generation quality. For example, with Llama-3.2-3b, it achieves up to 5.69× speedup with 2.00% accuracy gain on the Scene Graph dataset, 6.52× speedup with 1.00% accuracy gain on the OAG dataset, and 6.07× speedup without accuracy loss on the DBLP dataset.

Related Work

RAG. RAG (Fan et al. 2024; Hu et al. 2025; Huang et al. 2025; Lewis et al. 2020; Ram et al. 2023; Trivedi et al. 2023; Yu et al. 2024; Zhao et al. 2024) enhances LLMs by retrieving external knowledge to mitigate hallucination (Huang et al. 2025) and improve reliability (Gao et al. 2023). Recently, graph-based RAG was proposed (Guo et al. 2024; He et al. 2024; Hu et al. 2025), which retrieves query-relevant subgraphs from textual graphs and performs generation by jointly leveraging text and structures. For example, G-Retriever (He et al. 2024) retrieves individual nodes and edges and reconstructs query-specific subgraphs for generation, while GRAG (Hu et al. 2025) retrieves subgraphs directly by embedding k-hop ego networks and pruning irrelevant components. These graph-based RAG methods focus primarily on single-query processing and overlook the holistic optimization opportunities enabled by in-batch query execution. Moreover, they pay little attention to inference efficiency, concentrating solely on improving retrieval and generation quality. In this paper, we aim to improve the inference efficiency of graph-based RAG by exploiting structural redundancy through batch-level processing.

KV cache reuse. Recent efforts (Gim et al. 2024; Jin et al. 2025; Zheng et al. 2024; Yao et al. 2025; Jin et al. 2024b; Lu et al. 2025) have explored reusing KV cache to reduce redundant computation during LLM inference, primarily within text-based scenarios. For instance, SGLang (Zheng et al. 2024) identifies reusable intermediate states across different requests in multi-turn conversations, while Prompt Cache (Gim et al. 2024) enables flexible token reuse by ensuring each prompt module is self-contained and semantically independent. Furthermore, RAGCache (Jin et al. 2025) exploits the retrieved document sequences to construct a multilevel caching system, improving efficiency without altering generation outputs. However, these approaches are tailored to text-only settings and do not address the unique challenges associated with graph retrieval, where the retrieved subgraphs are inherently interconnected and leveraging their topological structure is critical to maintain generation quality. To bridge this gap, we introduce a novel caching paradigm based on structured subgraphs and propose SubGCache, a lightweight and efficient framework for subgraph-level prompt caching that identifies and exploits the structural redundancy in retrieved subgraphs.

Methodology

We consider a new in-batch setting for graph-based RAG, where a batch of queries $\{q_1, q_2, \ldots, q_m\}$ are issued simultaneously to a shared system. In the standard graph-based RAG pipeline, each query q_i retrieves a subgraph s_i from a textual graph G, and then an LLM generates a response a_i based on the augmented input formed by q_i and s_i.

While effective, this per-query processing paradigm results in substantial redundant computation. To address this limi-

tation, we propose SubGCache, a lightweight and efficient plug-and-play caching framework that identifies shared subgraphs across queries and eliminates redundant computation by caching and reusing their KV tensors.

Architecture Overview

Figure 2 illustrates SubGCache and its integration into the standard graph-based RAG pipeline. Specifically, given a textual graph G, a batch of queries $\{q_1, q_2, \ldots, q_m\}$, and their corresponding retrieved subgraphs $\{s_1, s_2, \ldots, s_m\}$, SubGCache is designed to reduce redundant computation by leveraging structural redundancy across queries through the following three key steps: (1) Query clustering: In-batch queries are grouped based on structural and semantic similarities in their retrieved subgraphs, enabling the identification of shared subgraph components. (2) Representative subgraph construction: For each cluster, we merge the nodes and edges of all associated subgraphs to create a representative subgraph that preserves the relational context required for high-quality response generation. (3) Subgraph-level cache reuse: SubGCache processes queries in a cluster-wise manner. For each cluster, it computes the KV cache for the representative subgraph only once, reuses it across all associated queries, and releases it before moving to the next. This substantially reduces redundant computation and improves inference efficiency, without compromising generation quality.

Query Clustering

Graph Embedding via Pretrained GNN. The key intuition behind query clustering is that in-batch queries may retrieve subgraphs that are structurally and semantically similar. However, such overlap is neither known beforehand nor trivial to detect, as retrieved subgraphs may differ in node identifiers, local context, or overall topology. To address this challenge, we encode each retrieved subgraph into a graph embedding using a pretrained GNN initialized with SentenceBERT-based node features—the same setup used for soft prompt construction in existing graph-based RAG. These embeddings capture both semantic and structural characteristics, enabling effective comparison across subgraphs.

Hierarchical Clustering. Once the subgraph embeddings $\{e_1, e_2, \ldots, e_m\}$ are obtained, we perform hierarchical clustering over these embeddings to group similar subgraphs. As a result, subgraphs with substantial overlap (*i.e., subgraph-level redundancy across queries)* are automatically assigned to the same cluster. Their corresponding queries are thus grouped for shared processing, effectively addressing the challenge of structural redundancy identification.

Example. As illustrated in Figure 2, given a batch of queries $\{q_1, q_2, \ldots, q_m\}$ and their corresponding retrieved subgraphs $\{s_1, s_2, \ldots, s_m\}$, we first encode each subgraph into an embedding $\{e_1, e_2, \ldots, e_m\}$ using the pretrained GNN. Hierarchical clustering is then applied with a predefined number of clusters (*i.e.*, $c = 2$) to group similar embeddings together. For instance, embeddings e_1, e_2 and e_3 are assigned to cluster $C_{2,1}$, while the remaining form cluster $C_{2,2}$. Consequently, both the retrieved subgraphs and their associated queries are grouped accordingly, laying the foundation for downstream subgraph-level cache reuse.

Representative Subgraph Construction

Although queries with significant structural redundancy can be effectively identified through GNN-based clustering, the retrieved subgraphs are generally partially shared, as they may differ in size, topology, or node alignment. This contrasts with text-based reuse methods, where cached units such as sentences or document chunks are typically assumed to be identical and easily shareable.

To address this challenge, we introduce a simple and effective representative subgraph as the shared structural input and natural cached unit for each query cluster. It is constructed by taking the union of all nodes and edges from the subgraphs retrieved by the queries within a specific cluster. The resulting structure captures the full relational context shared across the cluster and serves as a comprehensive, reusable input that supports both accurate response generation and structural redundancy elimination.

Example. As presented in Figure 2, suppose the subgraph embeddings are grouped into two clusters: $C_{2,1}$ containing s_1, s_2, and s_3, and $C_{2,2}$ containing the remaining subgraphs. For cluster $C_{2,1}$, we construct the representative subgraph $s_{2,1}$ by merging all nodes and edges from the corresponding retrieved subgraphs $\{s_1, s_2, s_3\}$. Likewise, another representative subgraph $s_{2,2}$ is constructed for $C_{2,2}$ by merging the subgraphs assigned to that cluster.

Subgraph-level Cache Reuse

While representative subgraphs provide unified structural input for each query cluster, efficiently leveraging them during response generation remains non-trivial. To achieve this, SubGCache adopts a cluster-wise processing strategy, enabled by a subgraph-level caching mechanism that pre-computes attention states once and reuses them across all queries within the same cluster. Specifically, for each cluster, SubGCache constructs a prompt based on its representative subgraph, following standard graph-based RAG pipelines. The prompt is then fed into the LLM to pre-compute intermediate attention states across transformer layers, which are stored in GPU memory as a cluster-wise KV cache and reused by all queries in the cluster. When processing each query, SubGCache appends query-specific question tokens to the cached subgraph prompt, allowing the model to reuse the precomputed KV cache for the shared structural context tokens and compute new KV tensors only over the appended question tokens, thereby avoiding redundant computation.

Once all queries in a cluster are processed, the corresponding KV cache is released to free GPU memory before moving to the next. This cluster-wise cache management eliminates redundant computation, reduces memory usage, and ensures scalability for large in-batch query workloads.

Example. Continuing the example in Figure 2, after obtaining clusters $C_{2,1}$ and $C_{2,2}$ with their representative subgraphs $s_{2,1}$ and $s_{2,2}$, SubGCache processes them sequentially. Specifically, it first processes $C_{2,1}$ by constructing a prompt for $s_{2,1}$ and computing its KV cache $P_{2,1}$, which is then stored in GPU memory. All queries in $C_{2,1}$ achieve cache hits by reusing this shared KV cache. Once all queries in $C_{2,1}$ are served, the cache is released to free GPU memory. Then,

Model	Scene Graph				OAG			
	ACC↑	RT↓	TTFT↓	PFTT↓	ACC↑	RT↓	TTFT↓	PFTT↓
Backbone: Llama-3.2-3B								
G-Retriever	62.00	664.71	642.86	321.26	96.00	974.94	921.00	245.07
G-Retriever+SubGCache	**64.00**	**132.93**	**112.93**	**26.92**	**97.00**	**190.73**	**141.19**	**29.94**
$\Delta_{G-Retriever}$	↑ 2.00	↑ 5.00×	↑ 5.69×	↑ 11.93×	↑ 1.00	↑ 5.11×	↑ 6.52×	↑ 8.19×
GRAG	60.00	559.17	540.99	400.18	**98.00**	243.50	186.61	82.63
GRAG+SubGCache	**61.00**	**154.79**	**132.60**	**19.77**	97.00	**174.79**	**124.44**	**30.84**
Δ_{GRAG}	↑ 1.00	↑ 3.61×	↑ 4.08×	↑ 19.77×	↓ 1.00	↑ 1.39×	↑ 1.50×	↑ 2.68×
Backbone: Llama-2-7B								
G-Retriever	59.00	970.04	938.44	705.51	**94.00**	922.83	852.95	524.32
G-Retriever+SubGCache	**66.00**	**168.52**	**140.54**	**45.55**	**94.00**	**282.52**	**217.26**	**60.63**
$\Delta_{G-Retriever}$	↑ 7.00	↑ 5.76×	↑ 6.68×	↑ 15.49×	0.00	↑ 3.27×	↑ 3.93×	↑ 8.65×
GRAG	56.00	1299.79	1264.70	924.11	**99.00**	441.97	375.13	217.17
GRAG+SubGCache	**57.00**	**234.87**	**202.96**	**50.53**	**99.00**	**258.67**	**188.84**	**62.23**
Δ_{GRAG}	↑ 1.00	↑ 5.53×	↑ 6.23×	↑ 18.29×	0.00	↑ 1.71×	↑ 1.99×	↑ 3.49×
Backbone: Mistral-7B								
G-Retriever	**66.00**	960.42	930.76	742.55	**99.00**	766.29	687.10	552.65
G-Retriever+SubGCache	**66.00**	**236.21**	**204.32**	**52.11**	**99.00**	**315.35**	**237.74**	**63.42**
$\Delta_{G-Retriever}$	0.00	↑ 4.07×	↑ 4.56×	↑ 14.25×	0.00	↑ 2.43×	↑ 2.89×	↑ 8.71×
GRAG	57.00	1113.75	1081.97	966.54	**99.00**	539.39	458.70	243.82
GRAG+SubGCache	**66.00**	**194.68**	**164.01**	**52.44**	**99.00**	**237.04**	**159.25**	**63.04**
Δ_{GRAG}	↑ 9.00	↑ 5.72×	↑ 6.60×	↑ 18.43×	0.00	↑ 2.28×	↑ 2.88×	↑ 3.87×
Backbone: Falcon-7B								
G-Retriever	64.00	826.56	790.46	702.11	**98.00**	1049.20	964.67	526.74
G-Retriever+SubGCache	**66.00**	**195.29**	**159.81**	**52.16**	97.00	**374.53**	**294.55**	**59.66**
$\Delta_{G-Retriever}$	↑ 2.00	↑ 4.23×	↑ 4.95×	↑ 13.46×	↓ 1.00	↑ 2.80×	↑ 3.28×	↑ 8.83×
GRAG	57.00	1142.68	1105.78	954.17	**97.00**	483.21	400.54	198.88
GRAG+SubGCache	**60.00**	**272.45**	**238.04**	**50.49**	96.00	**249.28**	**169.03**	**59.18**
Δ_{GRAG}	↑ 3.00	↑ 4.19×	↑ 4.65×	↑ 18.90×	↓ 1.00	↑ 1.94×	↑ 2.37×	↑ 3.36×

Table 1: Overall performance. The best results are highlighted in bold.

SubGCache repeats the procedure for $C_{2,2}$ using $s_{2,2}$ and $P_{2,2}$. This cluster-wise reuse and release strategy ensures efficient memory usage and eliminates redundant computation, even with large in-batch workloads.

Discussion. SubGCache enables flexible control over cache reuse granularity by adjusting the clustering level. Finer clustering (*i.e.*, more clusters) yields more query-specific prompts, but limits reuse opportunities. In contrast, coarser clustering (*i.e.*, fewer clusters) promotes greater reuse by grouping more queries together and generating subgraphs with broader context. This often enhances generation quality, although it may also introduce minor noise in rare cases, as observed in our experiments. Notably, when each query forms its own cluster, the method naturally reduces to standard graph-based RAG.

Experiments

Datasets: We use Scene Graph (He et al. 2024), OAG (Zhang et al. 2019; Zhu et al. 2025b) and DBLP (Tang et al. 2008) datasets for in-batch query evaluation for graph-based RAG. The details of datasets are available at Appendix A in the extended version.

Setup: We adopt two representative graph-based RAG methods, G-Retriever (He et al. 2024) and GRAG (Hu et al. 2025), as our baseline models. SubGCache is then integrated as a plug-and-play module, resulting in G-Retriever+SubGCache and GRAG+SubGCache. All methods are tested with different LLM backbones: Llama-3.2-3B (Dubey et al. 2024), Llama-2-7B (Touvron et al. 2023), Mistral-7B (Team et al. 2023), and Falcon-7B (Penedo et al. 2023). All experiments are conducted in an inference-only setting with frozen LLMs. We evaluate performance with four metrics: accuracy (ACC), response time (RT), time-to-first-token (TTFT), and prefill and first token time (PFTT). ACC is reported as a percentage (%), and the other metrics in milliseconds (ms). Details on configurations and evaluation metrics are provided in Appendix B and C, respectively.

Main Results

Table 1 summarizes the overall results on Scene Graph and OAG datasets using four LLM backbones, and additional results on the DBLP dataset are provided in Appendix G.

Reduced latency with comparable effectiveness. Compared to the baseline models G-Retriever and GRAG, integrating

Methods	Scene Graph				OAG			
	ACC↑	RT↓	TTFT↓	PFTT↓	ACC↑	RT↓	TTFT↓	PFTT↓
50 in-batch queries								
G-Retriever	58.00	479.90	458.56	308.45	98.00	386.50	331.04	222.13
G-Retriever+SubGCache	**64.00**	**155.96**	**134.94**	**28.02**	**100.00**	**192.98**	**140.92**	**33.00**
$\Delta_{G-Retriever}$	↑ 6.00	↑ 3.08×	↑ 3.40×	↑ 11.01×	↑ 2.00	↑ 2.00×	↑ 2.35×	↑ 6.73×
GRAG	**58.00**	260.42	251.51	396.92	**100.00**	248.94	192.28	83.41
GRAG+SubGCache	**58.00**	**80.82**	**68.75**	**30.10**	**100.00**	**181.44**	**127.00**	**28.68**
Δ_{GRAG}	0.00	↑ 3.22×	↑ 3.66×	↑ 13.19×	0.00	↑ 1.37×	↑ 1.51×	↑ 2.91×
150 in-batch queries								
G-Retriever	64.00	643.15	621.96	316.34	**97.33**	547.08	491.47	221.23
G-Retriever+SubGCache	**65.33**	**145.15**	**123.35**	**28.07**	**97.33**	**184.59**	**134.36**	**29.00**
$\Delta_{G-Retriever}$	↑ 1.33	↑ 4.43×	↑ 5.04×	↑ 11.27×	0.00	↑ 2.96×	↑ 3.66×	↑ 7.63×
GRAG	58.67	543.09	786.74	400.69	**98.67**	237.65	184.10	80.52
GRAG+SubGCache	**59.33**	**162.81**	**206.61**	**29.76**	**98.67**	**179.81**	**130.32**	**29.91**
Δ_{GRAG}	↑ 0.66	↑ 3.34×	↑ 3.81×	↑ 13.46×	0.00	↑ 1.32×	↑ 1.41×	↑ 2.69×
200 in-batch queries								
G-Retriever	**64.50**	439.39	418.35	306.75	97.00	475.00	420.67	214.68
G-Retriever+SubGCache	**64.50**	**130.14**	**111.27**	**25.33**	**98.00**	**190.39**	**139.42**	**30.70**
$\Delta_{G-Retriever}$	0.00	↑ 3.38×	↑ 3.76×	↑ 12.11×	↑ 1.00	↑ 2.49×	↑ 3.02×	↑ 6.99×
GRAG	58.00	541.30	521.25	400.44	**99.00**	249.59	192.45	80.04
GRAG+SubGCache	**60.00**	**160.60**	**136.02**	**28.93**	98.50	**184.14**	**132.25**	**29.04**
Δ_{GRAG}	↑ 2.00	↑ 3.37×	↑ 3.83×	↑ 13.84×	↓ 0.50	↑ 1.36×	↑ 1.46×	↑ 2.76×

Table 2: Effect of different in-batch query size on the Scene Graph and OAG datasets. (Backbone: Llama-3.2-3B).

our SubGCache framework (*i.e.*, G-Retriever+SubGCache and GRAG+SubGCache) consistently reduces latency across both datasets. Specifically, for G-Retriever, SubGCache achieves up to 5.76× / 5.11× reduction in RT, 6.68× / 6.52× speedup in TTFT, and 15.49× / 8.19× reduction in PFTT on Scene Graph and OAG, respectively. For GRAG, it yields 5.72× / 2.28× reduction in RT, 6.60× / 2.88× speedup in TTFT, and 18.43× / 3.87× reduction in PFTT. These substantial latency reductions come with comparable or even improved accuracy: up to 9.00% gain on Scene Graph, and only a minor drop (*i.e.*, 1.00%) in rare cases on OAG.

Consistent improvement across LLM backbones. SubGCache consistently reduces latency with comparable generation quality across different LLM backbones, regardless of architectural or scale differences. This confirms its robustness and generalization as a plug-and-play optimization.

Understanding why SubGCache works. SubGCache significantly reduces inference latency with comparable accuracy by addressing two key challenges in graph-based RAG: identifying and exploiting structural redundancy. (1) It clusters in-batch queries based on the semantic and structural similarity of their retrieved subgraphs using pretrained GNN embeddings, enabling queries with overlapping context to be grouped and processed together. (2) For each cluster, it constructs a representative subgraph by merging the retrieved subgraphs into a unified structure. The KV cache for this shared input is computed once and reused across all queries in the cluster, avoiding redundant computation while preserving relational context. Only in rare cases, the merged context may introduce minor noise, leading to slight degradation in effectiveness (less than 1.00%). Together, these two designs explain the observed latency reduction and stable generation quality across datasets and LLM backbones, highlighting SubGCache's practical value as an efficient caching strategy for graph-based RAG.

Impact of Cluster Number

To evaluate the effect of cluster number, we compare G-Retriever with G-Retriever+SubGCache by varying cluster numbers in {1, 2, 3, 4, 5, 10, 20}, and report ACC (%) and TTFT (s) on both datasets using the Llama-3.2-3B backbone, as shown in Figure 3.

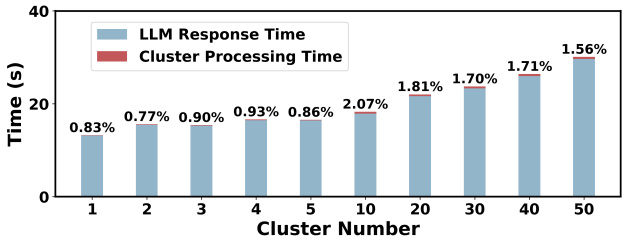
Trade-off between latency and accuracy. As observed, finer clustering (*i.e.*, more clusters) tends to preserve more query-specific context that can improve accuracy, while coarser clustering boosts cache reuse and reduces latency. However, this trade-off is not strictly monotonic. Both latency and accuracy fluctuate across cluster settings due to competing factors. Fewer clusters enable more frequent reuse but larger representative subgraphs, increasing prompt length and cache overhead. More clusters reduce reuse opportunities but produce shorter prompts. This results in a non-linear latency trend, where TTFT does not steadily increase with cluster number. On the accuracy side, coarser clustering may improve quality by aggregating richer subgraph context, or slightly degrades performance by introducing irrelevant information.

Despite these variations, SubGCache performs well even with small cluster number. On Scene Graph, the 1-cluster

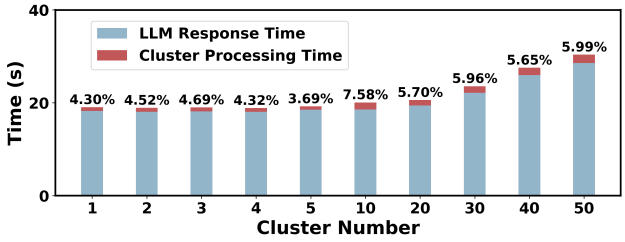
(a) Scene Graph

(b) OAG

Figure 3: Impact of cluster number.



(a) Scene Graph



(b) OAG

Figure 4: Cluster processing time vs. LLM response time by varying cluster numbers.

setting achieves 5.69× speedup in TTFT while surpassing baseline's accuracy. On OAG, the 2-cluster setting yields a favorable result, achieving a 1.00% accuracy gain alongside 6.52× speedup in TTFT, respectively. These results highlight the importance of selecting an appropriate clustering granularity to balance latency and accuracy.

Cluster Processing Time

Figure 4 compares the LLM response time (blue) and cluster processing time (red) of G-Retriever+SubGCache under different cluster numbers {1, 2, 3, 4, 5, 10, 20, 30, 40, 50} on both datasets. Cluster processing time includes graph encoding, hierarchical clustering, and representative subgraph construction. We summarize four key observations:

Minimal processing overhead. Cluster processing time remains low across all cluster configurations. On Scene Graph, it accounts for less than 2.1% of total latency, and below 6% even on the larger OAG dataset with 50 clusters. These results show that SubGCache's clustering stage introduces only modest overhead relative to total inference time.

Higher cost on larger graphs. OAG incurs higher processing time than Scene Graph, primarily due to its larger graph size. These properties result in larger retrieved subgraphs, increasing the number of nodes and edges to encode, and leading to higher computational cost during both GNN-based embedding and representative subgraph construction.

Non-monotonic variation. Cluster processing time does not increase linearly with cluster number. While more clusters require more representative subgraphs, each individual cluster is smaller, reducing per-cluster encoding time. Additionally, hierarchical clustering complexity depends on the number of inputs, rather than the number of output clusters, contributing to the non-linear trend.

LLM response time generally increases with cluster number. Finer clustering limits cache reuse across queries, leading to longer response times. Slight fluctuations arise from larger merged subgraphs, which generate longer prompts and incur higher inference costs.

Impact of in-batch size

We evaluate SubGCache under varying in-batch sizes: 50, 100 (from Table 1), 150, and 200, using Llama-3.2-3B. The results are in Table 2, with additional evaluations using Llama-2-7B, Mistral-7B, and Falcon-7B provided in Appendix E. As observed, SubGCache consistently reduces latency while preserving and often improving generation quality across different in-batch sizes. The results demonstrate SubGCache scales well with in-batch size, supporting its practicality in real-world applications.

Remark. The case study is presented in Appendix D, and additional comparisons of different linkage strategies are provided in Appendix F.

Conclusion

This paper introduces a new research problem: in-batch query processing for graph-based RAG, aiming to reduce inference latency through batch-level optimization. To address this, we propose SubGCache, a novel subgraph-level caching framework that tackles the problem-specific challenges of identifying and exploiting structural redundancy in retrieved subgraphs. SubGCache is simple, plug-and-play, and easily integrable into existing graph-based RAG approaches. Experiments across various LLM backbones and graph-based RAG frameworks demonstrate that SubGCache significantly reduces inference latency, while preserving and even improving generation quality.

Acknowledgments

This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20224-0011 and Tier 1 Award (RG20/24)). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. This research is also supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No. 2024A03J0630).

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report.

Borgeaud, S.; Mensch, A.; Hoffmann, J.; Cai, T.; Rutherford, E.; Millican, K.; Van Den Driessche, G. B.; Lespiau, J.-B.; Damoc, B.; Clark, A.; et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, 2206–2240. PMLR.

Bouros, P.; Titkov, A.; Christodoulou, G.; Rauch, C.; and Mamoulis, N. 2024. HINT on Steroids: Batch Query Processing for Interval Data. In *EDBT*, 440–446.

Choudhury, F. M.; Culpepper, J. S.; Bao, Z.; and Sellis, T. 2018. Batch processing of top-k spatial-textual queries. volume 3, 1–40. ACM New York, NY, USA.

Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H. W.; Sutton, C.; Gehrmann, S.; et al. 2023. Palm: Scaling language modeling with pathways. volume 24, 1–113.

Ding, S.; Attenberg, J.; Baeza-Yates, R.; and Suel, T. 2011. Batch query processing for web search engines. In *Proceedings of the fourth ACM international conference on Web search and data mining*, 137–146.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The Llama 3 Herd of Models.

Fan, W.; Ding, Y.; Ning, L.; Wang, S.; Li, H.; Yin, D.; Chua, T.-S.; and Li, Q. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 6491–6501.

Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Guo, Q.; Wang, M.; et al. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey.

Gim, I.; Chen, G.; Lee, S.-s.; Sarda, N.; Khandelwal, A.; and Zhong, L. 2024. Prompt cache: Modular attention reuse for low-latency inference. volume 6, 325–338.

Guo, Z.; Xia, L.; Yu, Y.; Ao, T.; and Huang, C. 2024. Lightrag: Simple and fast retrieval-augmented generation.

He, X.; Tian, Y.; Sun, Y.; Chawla, N.; Laurent, T.; LeCun, Y.; Bresson, X.; and Hooi, B. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. volume 37, 132876–132907.

Hu, Y.; Lei, Z.; Zhang, Z.; Pan, B.; Ling, C.; and Zhao, L. 2025. GRAG: Graph Retrieval-Augmented Generation. In Chiruzzo, L.; Ritter, A.; and Wang, L., eds., *Findings of the Association for Computational Linguistics: NAACL 2025*, 4145–4157. Albuquerque, New Mexico: Association for Computational Linguistics. ISBN 979-8-89176-195-7.

Huang, L.; Yu, W.; Ma, W.; Zhong, W.; Feng, Z.; Wang, H.; Chen, Q.; Peng, W.; Feng, X.; Qin, B.; et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. volume 43, 1–55. ACM New York, NY.

Jin, B.; Liu, G.; Han, C.; Jiang, M.; Ji, H.; and Han, J. 2024a. Large language models on graphs: A comprehensive survey. IEEE.

Jin, C.; Zhang, Z.; Jiang, X.; Liu, F.; Liu, S.; Liu, X.; and Jin, X. 2025. RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. *ACM Trans. Comput. Syst.*, 44(1).

Jin, S.; Liu, X.; Zhang, Q.; and Mao, Z. M. 2024b. Compute or load kv cache? why not both?

Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-t.; Rocktäschel, T.; et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. volume 33, 9459–9474.

Li, Y.; Li, Z.; Wang, P.; Li, J.; Sun, X.; Cheng, H.; and Yu, J. X. 2024. A survey of graph meets large language model: progress and future directions. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 8123–8131.

Lu, S.; Wang, H.; Rong, Y.; Chen, Z.; and Tang, Y. 2025. Turborag: Accelerating retrieval-augmented generation with precomputed kv caches for chunked text. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 6599–6612.

Penedo, G.; Malartic, Q.; Hesslow, D.; Cojocaru, R.; Cappelli, A.; Alobeidli, H.; Pannier, B.; Almazrouei, E.; and Launay, J. 2023. The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only.

Ram, O.; Levine, Y.; Dalmedigos, I.; Muhlgay, D.; Shashua, A.; Leyton-Brown, K.; and Shoham, Y. 2023. In-context retrieval-augmented language models. volume 11, 1316–1331. MIT Press One Broadway, 12th Floor, Cambridge, Massachusetts 02142, USA ....

Sinha, A.; Shen, Z.; Song, Y.; Ma, H.; Eide, D.; Hsu, B.-J.; and Wang, K. 2015. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, 243–246.

Tang, J.; Zhang, J.; Yao, L.; Li, J.; Zhang, L.; and Su, Z. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 990–998.

Team, M. N.; et al. 2023. Introducing mpt-7b: A new standard for open-source, commercially usable llms.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models.

Trivedi, H.; Balasubramanian, N.; Khot, T.; and Sabharwal, A. 2023. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Wang, H.; Feng, S.; He, T.; Tan, Z.; Han, X.; and Tsvetkov, Y. 2023. Can language models solve graph problems in natural language? volume 36, 30840–30861.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2020. A comprehensive survey on graph neural networks. volume 32, 4–24. IEEE.

Yao, J.; Li, H.; Liu, Y.; Ray, S.; Cheng, Y.; Zhang, Q.; Du, K.; Lu, S.; and Jiang, J. 2025. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, 94–109.

Yu, H.; Gan, A.; Zhang, K.; Tong, S.; Liu, Q.; and Liu, Z. 2024. Evaluation of retrieval-augmented generation: A survey. In *CCF Conference on Big Data*, 102–120. Springer.

Zhang, B.; Haddow, B.; and Birch, A. 2023. Prompting large language model for machine translation: A case study. In *International Conference on Machine Learning*, 41092–41110. PMLR.

Zhang, F.; Liu, X.; Tang, J.; Dong, Y.; Yao, P.; Zhang, J.; Gu, X.; Wang, Y.; Shao, B.; Li, R.; et al. 2019. OAG: Toward linking large-scale heterogeneous entity graphs. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2585–2595.

Zhang, Q.; Teng, Z.; Wu, D.; and Wang, J. 2024a. An Enhanced Batch Query Architecture in Real-time Recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 5078–5085.

Zhang, T.; Ladhak, F.; Durmus, E.; Liang, P.; McKeown, K.; and Hashimoto, T. B. 2024b. Benchmarking large language models for news summarization. volume 12, 39–57.

Zhao, P.; Zhang, H.; Yu, Q.; Wang, Z.; Geng, Y.; Fu, F.; Yang, L.; Zhang, W.; Jiang, J.; and Cui, B. 2024. Retrieval-augmented generation for ai-generated content: A survey.

Zheng, L.; Yin, L.; Xie, Z.; Sun, C.; Huang, J.; Yu, C. H.; Cao, S.; Kozyrakis, C.; Stoica, I.; Gonzalez, J. E.; Barrett, C.; and Sheng, Y. 2024. SGLang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24. Red Hook, NY, USA: Curran Associates Inc. ISBN 9798331314385.

Zhu, Q.; Zhang, L.; Xu, Q.; Liu, K.; Long, C.; and Wang, X. 2025a. HHGT: hierarchical heterogeneous graph transformer for heterogeneous graph representation learning. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, 318–326.

Zhu, Q.; Zhang, L.; Xu, Q.; and Long, C. 2025b. HierPromptLM: A Pure PLM-based Framework for Representation Learning on Heterogeneous Text-rich Networks.