

SANKOFA: Online Query-adaptive Dynamic Graph Summaries

Ama Bembua Bainson
ama@cs.au.dk
Aarhus University

Kasper Overgaard Mortensen
km@cs.au.dk
Aarhus University

Klim Zaporjets
klim@cs.au.dk
Aarhus University

Davide Mottin
davide@cs.au.dk
Aarhus University

Panagiotis Karras
piekarras@gmail.com
U. of Copenhagen & Aarhus U.

ABSTRACT

Knowledge Graphs (KGs) model semantic relationships between entities and are widely used in tasks such as question answering, recommendation, and large language model training. As KGs grow to billions of triples, storing and querying them on local devices becomes infeasible and also raises privacy concerns. KG summarization methods produce compact representations, yet existing methods are either static or not attuned to user interests. In this paper, we propose SANKOFA, a query-adaptive, incremental, and on-line KG summarization method. SANKOFA maintains personalized summaries on-device by incrementally updating them in response to new queries and data using a random-walk-based adaptation of node-centric Personalized PageRank, efficiently identifying the most relevant nodes while bounding summary size. We provide formal guarantees for our PPR approximation and demonstrate experimentally that SANKOFA outperforms existing personalized KG summarization techniques in speed and accuracy, enabling adaptive, privacy-preserving KG summaries that evolve with user interests on dynamic knowledge graphs.

PVLDB Reference Format:

Ama Bembua Bainson, Kasper Overgaard Mortensen, Klim Zaporjets, Davide Mottin, and Panagiotis Karras. SANKOFA: Online Query-adaptive Dynamic Graph Summaries. PVLDB, 19(10): 2908 - 2921, 2026.
doi:10.14778/3828612.3828641

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/AU-DIS/SANKOFA.git>.

1 INTRODUCTION

Knowledge Graphs (KGs) represent semantic relationships between entities through *triples* and conveniently represent AI-driven systems for language question answering [23, 72, 73], training of and prompt construction in large language models (LLMs) [41], semantic search [66], prediction [46, 47] and recommendation [69]. KGs are also lucratively applied to domain-specific research, such as drug discovery [37], and general encyclopedic search [5, 54, 60]. Users and domain experts seek information in KGs by issuing queries [14, 18, 51].

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 19, No. 10 ISSN 2150-8097.
doi:10.14778/3828612.3828641

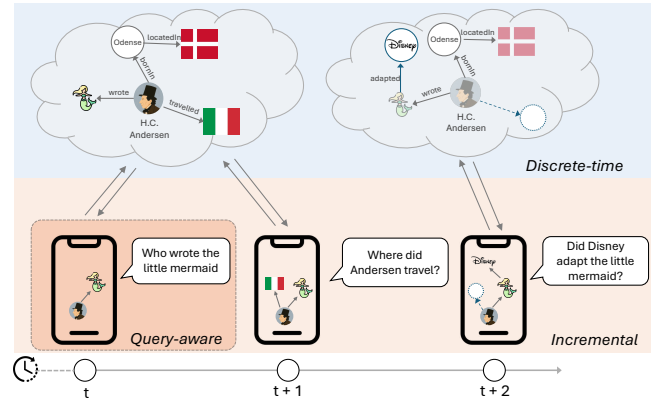


Figure 1: SANKOFA maintains a query-adaptive incremental summary of a server-side KG locally at the user's device, while the graph and queries change; prior works do not support incremental online summaries.

A KG should adapt to the evolution and growth of knowledge, raising requirements for storage to the order of billions of relationships. In effect, current KGs do not fit into ordinary devices and consumer hardware, requiring powerful cloud-based solutions, and hence raising privacy concerns [11, 39, 67].

KG summarization [10, 36] addresses the storage bottleneck by building a succinct version of the graph that reduces redundancy [9, 20, 25] and possibly allows information loss [9, 24, 27]. Such summaries are stored on-device, mitigating privacy concerns, and allow prompt query execution as they incur no additional communication overhead. However, these techniques are user-oblivious, as the summaries they provide do not cater to user preferences. Queries on such summaries may thus fail to yield the information most relevant to the user.

Query-aware KG summaries [32, 48, 56] aim to address this awareness problem: they retain only parts of the graph that are relevant to the user, based on past queries. Such summaries are stored on the user's device and accessed exclusively by them, enhancing privacy. Still, query-aware KG summaries are *static*, as they are constructed in a one-off manner based on the available data and past queries and do not adapt in response to incoming queries and data updates in an *online* manner. *Online KG summarization* methods [19, 28, 70], on the other hand adapt to data updates, but forfeit query-awareness.

In this paper, we propose SANKOFA,¹ a query-adaptive, incremental, and online KG summarization method. SANKOFA incrementally

¹From the Twi language, 'retrieve from the past', used in the proverb "se wo were fi na wosankofa a yenkyi" ("We do not blame you for reclaiming what slips from your mind").

updates a personalized summary in response to new queries and data through a random-walk-based approach [40]; it adapts incremental Personalized PageRank (PPR) [8] to reflect changes in the query nodes, while *sampling* random walks to efficiently maintain a PPR approximation, which guides the summary construction using the nodes of highest PPR.

The **key challenge** our method faces is that the produced summary must adapt to a changing query workload in online fashion and correctly answer both past and future queries.

Figure 1 shows an example illustrating how SANKOFA incrementally constructs a personalized, query-aware online summary in response to the user’s evolving information needs. At the beginning of the interaction, the user inquires about the author of ‘The Little Mermaid’ and hence SANKOFA generates a summary that identifies H.C. Andersen as the author and prioritizes information relevant to this query. As the user subsequently asks where H.C. Andersen had traveled, SANKOFA expands the summary with information about the author’s travels to Italy while retaining the information about his authorship. Thereby, SANKOFA may continue answering both past and newly introduced queries. In the next snapshot, the node *Italy* and the edge *travelled* (shown by dashed lines) are deleted and a new node, *Disney*, and the edge *adapted* are introduced (with existing connections grayed-out). As the user’s focus later shifts to the film adaptation of the tale, the information about Andersen’s travels becomes less relevant for the user and is gradually phased out to make room for new facts about the Disney adaptation of ‘The Little Mermaid’. Overall, SANKOFA maintains a personalized query-adaptive summary that evolves incrementally in online fashion and balances between the preservation of previously relevant information and the integration of new knowledge as the user’s interests change over time.

Our chief contributions are as follows:

- We frame the problem of query-adaptive, incremental, and online knowledge graph summarization.
- We propose SANKOFA, a random-walk-based adaptation of incremental node-centric PPR that maintains a summary within bounded size.
- We provide formal guarantees for our PPR approximation. We experimentally show that SANKOFA is up to 30× faster and 25% more accurate than the current state-of-the-art works.

2 RELATED WORK

We characterize KG summarization methods as *query-aware* (QA) if they integrate the user’s past queries, *online* (OS) if they quickly adapt to data changes, and *incremental* if they progressively adapt to new queries or data. For a complete overview of graph summarization, we refer the reader to surveys focusing on general graphs [36], semantic graphs [10], and static and dynamic KGs [61].

Query-aware (QA) methods aim to preserve relevant information considering recent user queries. KSD [62] generates an RDF summary tailored to the average of three coverage measures over a set of query keywords to cover their associated classes, properties and entities. KEYKG+ [50] identifies the nodes in a weighted graph that match each query keyword and finds a minimum-weight Group Steiner Tree (GST) containing at least one node from each keyword’s set. iSummary [56] follows frequent shortest paths to

link nodes related to a selected node according to a query log. WB-Sum [58] connects the top- k RDF schema nodes by importance to maximize the coverage of workload queries. GLIMPSE [48] maximizes the log-likelihood that a user prefers the summary content according to their past queries. SummaryGPT [57] merges nodes extracted from query logs on-the-fly into semantic categories using ChatGPT and the KG type hierarchy, yet does not maintain any memory of past graph changes. APEX² [32] incrementally updates the summary to reflect evolving user interests from the query history on a static graph using heat diffusion, and preserves relevant information from previous time steps.

Table 1: Characteristics of query-aware and discrete-time KG summarization methods.

Algorithm	Type	Backbone	query-aware	discrete-time	online	incremental
KSD [62]	non-quotient	weight max coverage	✓	✗	✗	✗
KEYKG+ [50]	non-quotient	group Steiner tree	✓	✗	✗	✗
iSUMMARY [56]	non-quotient	min-weight Steiner tree	✓	✗	✗	✗
WBSum [58]	non-quotient	Steiner tree	✓	✗	✗	✗
GLIMPSE [48]	non-quotient	submodular max	✓	✗	✗	✗
APEX ² [32]	non-quotient	heat diffusion	✓	✗	(✓)	✓
SummaryGPT [57]	quotient	LLM	✓	✗	(✓)	✗
RDFQuotient [19]	quotient	equivalence relation	✗	✓	✗	✓
DSSG [28]	patterns	MDL	✗	✓	(✓)	✓
ANOT [70]	rules	MDL	✗	✓	(✓)	✓
SANKOFA	non-quotient	PPR	✓	✓	✓	✓

Discrete-time KG summarization integrate the graph evolution through efficient updates, without full recomputation. RDFQuotient [19] incrementally clusters RDF nodes by structural similarity. DSSG [28] incrementally updates the summary of a dynamic graph to capture structural changes, maximizing information gain over the analyst’s current knowledge by the Minimum Description Length (MDL) principle. AnOT [70], also MDL-based, summarizes temporal knowledge graphs into rule graphs that encode complex semantic patterns for anomaly detection and incrementally updates the summary. APEX² [32] is *incremental* with respect to queries, yet not discrete-time with respect to the underlying graph.

Online KG summarization generates real-time summaries of the graph state without using past-query context. Only a few existing methods are genuinely designed for online operation. APEX² [32] produces incremental summaries for individual queries, with a reported latency of less than six seconds. AnOT [70], originally developed for anomaly detection, summarizes temporal graphs, with a reported latency of 44–2300 seconds for different graph sizes. SummaryGPT [57], presented only as a demonstration, outputs online summaries but does not report runtime.

Table 1 gathers existing query-aware and discrete-time methods and their properties. We mark with ‘(✓)’ methods that are claimed to be online but do not satisfy the requirement by our settings.

3 BACKGROUND

In this section we review knowledge graph summary types and review the PageRank algorithm on which our method is based.

3.1 Preliminaries

A **knowledge graph** (KG) $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{X})$ is a directed labeled graph comprising a set of n entities $\mathcal{E}$, a set of relations $\mathcal{R}$ and a

set of m triples $\mathcal{X} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$. A triple $x_{ijk} = (e_i, r_k, e_j) \in \mathcal{X}$ indicates a relation $r_k \in \mathcal{R}$ between $e_i, e_j \in \mathcal{E}$. KGs capture rich semantic information about entities via triples. An example of a triple is $(e_s, r_p, e_o) = (\text{Hans_Christian_Andersen}, \text{wrote}, \text{The_Little_Mermaid})$, where the subject entity id is `Hans_Christian_Andersen` and the object entity id is `The_Little_Mermaid`, connected by relation id `wrote` (Figure 1). The *neighborhood* of $u \in \mathcal{E}$ is the set $\mathcal{N}(u) = \{v \in \mathcal{E} \mid (u, \cdot, v) \in \mathcal{X} \vee (v, \cdot, u) \in \mathcal{X}\}$ of all entities connected to u by an incoming or outgoing triple. We denote the set of incoming neighbors as $\mathcal{N}^+(u)$ and that of outgoing neighbors as $\mathcal{N}^-(u)$.

A KG $\mathcal{G}$ may be extended to $(\mathcal{E}, \mathcal{R}, \mathcal{X}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{R}}, f, g, h)$ to include a set of semantic entity labels $\mathcal{L}_{\mathcal{E}}$ and a set of semantic relation labels $\mathcal{L}_{\mathcal{R}}$, equipped with non-surjective set-valued mappings f, g, h such that $f: \mathcal{E} \rightrightarrows \mathcal{L}_{\mathcal{E}}$ (where $f(\mathcal{E}) \subseteq \mathcal{L}_{\mathcal{E}}$), and likewise $g: \mathcal{R} \rightrightarrows \mathcal{L}_{\mathcal{R}}$ and $h: \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightrightarrows \mathcal{L}_{\mathcal{E}} \times \mathcal{L}_{\mathcal{R}} \times \mathcal{L}_{\mathcal{E}}$. These mappings allow the semantic information of labels to change without affecting the structural representation of entities and relations. This representation generalizes the Resource Description Framework (RDF) [13] and Labeled Property Graph (LPG) [3] abstractions by adding semantic annotations.

Dynamic Knowledge Graphs (DKGs) extend this notion to represent changes via a sequence of (real-time or archived) snapshots:

$$\mathcal{G}^D = \mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_T, \quad \mathcal{G}_t = (\mathcal{E}_t, \mathcal{R}_t, \mathcal{X}_t, \mathcal{L}_{\mathcal{E}_t}, \mathcal{L}_{\mathcal{R}_t}, f_t, g_t, h_t),$$

where each snapshot $\mathcal{G}_t$ captures the state of entities, relations, and facts at time t . Each snapshot may differ in nodes, edges, labels, or a combination thereof [17]. Dynamic graphs are also called *evolving* or *streaming* graphs [35, 38]. We consider *discrete-time* DKGs [12], which we distinguish from *continuous-time* KGs [12].

Query. A *query graph* is a connected graph $Q = (\mathcal{E}_Q, \mathcal{R}_Q, \mathcal{X}_Q)$, where $\mathcal{E}_Q = \mathcal{E} \cup \{\star\}$ is the set of entities, $\mathcal{R}_Q = \mathcal{R} \cup \{\star\}$ the set of relations, and $\mathcal{X}_Q \subseteq \mathcal{E}_Q \times \mathcal{R}_Q \times \mathcal{E}_Q$ the set of triples. The symbol $\star$ denotes an unspecified entity or relation. Each such unspecified element must be identified by the matching process. A **match** to a query $Q = (\mathcal{E}_Q, \mathcal{R}_Q, \mathcal{X}_Q)$ is a subgraph of $\mathcal{G}$ that is *subgraph isomorphic* to Q , i.e., there exists an *injective* mapping $\phi: \mathcal{E}_Q \rightarrow \mathcal{E}$ such that for every triple $(u, r, v) \in \mathcal{X}_Q$ there is a distinct corresponding triple $(\phi(u), r, \phi(v)) \in \mathcal{X}$. In such a mapping, the entities assigned to the nodes marked with $\star$ constitute **answers** to the query. This matching mechanism is consistent with standard graph query languages such as GQL [1, 14], Cypher [18], and SPARQL [21]. A **query workload** $\mathcal{W} = \{Q_t\}_{t \in \mathcal{T}}$ is a set of queries, each occurring at a valid timestamp $t \in \mathcal{T}$.

3.2 Knowledge Graph Summary Types

Knowledge graph summarization constructs a compact representation of a KG preserving the most relevant information to reduce storage and computational costs [16]. Existing summaries are classified as structural, statistical, pattern-based, and hybrid [30].

- (a) *Structural* summaries are based on the graph structure and can be *quotient* or *non-quotient*. Quotient summaries group similar nodes into *equivalence classes* while maintaining features such as edges and labels [49, 57]. Non-quotient summaries determine important nodes by measures such as *centrality* [43, 55, 71] or query workloads [48, 56, 58].

- (b) *Statistical* summaries utilize counts, frequencies, histograms, and relation ranking [45, 64].
- (c) *Pattern-based* summaries represent the graph via mined patterns or rules that may be used to reconstruct the graph [9, 52, 74].
- (d) *Hybrid* summaries combine two or more of the aforementioned techniques [44, 53].

We focus on generating a non-quotient structural KG summary.

3.3 PageRank

PageRank [40] is a widely used measure of node centrality in a graph; in its original formulation, it models a random surfer who, in each step, either follows a link with probability (damping factor) β or *teleports* to a random node with probability $\epsilon = 1 - \beta$.

$$\pi_i(u) = \frac{1 - \beta}{n} + \beta \sum_{w \in \mathcal{N}^+(u)} \frac{\pi_{i-1}(w)}{d^-(w)} \quad (1)$$

Teleportation ensures that the walk can reach *every* node, rendering the underlying stochastic process *ergodic*, i.e., admitting a stationary distribution [31, 40].

The classic PageRank algorithm computes an exact solution by matrix-algebra *power iteration*. This method, however, becomes prohibitive in terms of computation and memory usage for large-scale graphs, and especially problematic under frequent updates. *Monte Carlo* PageRank estimates values through random walks, allowing for parallelization and efficient adaptation to graph size and structural changes [6].

3.3.1 Monte Carlo estimation. The Monte Carlo approximation method derives R random walks of expected length $\frac{1}{1-\beta}$ for each node, with $R > q \ln n$ for a constant q . The number of times X_u the walks visit node u offers an approximation of the global PageRank [6, 8]:

$$\pi(u) = \frac{(1 - \beta)X_u}{nR} \quad (2)$$

This formulation scales to large or evolving graphs and supports incremental updates on nodes and edges by updating existing walks and introducing additional ones without full recomputation.

3.3.2 Updating the Monte Carlo Estimation. As the graph evolves, the R precomputed random walks must be updated to ensure correct PageRank value estimation. By the analysis of [8], each node stores R random walk segments of expected length $\frac{1}{1-\beta}$. The expected number of stored segments M_t that must be updated when the t^{th} edge arrives is [8, Proposition 2]:

$$\mathbb{E}[M_t] \leq \frac{nR}{1 - \beta} \mathbb{E} \left[\frac{\pi_{u_t}}{d^-(u_t)} \right]$$

By the random-permutation arrival model, it holds that [8, Lemma 3] $\mathbb{E} \left[\frac{\pi_{u_t}}{d^-(u_t)} \right] = \frac{1}{t}$. As extending a broken walk segment requires expected cost $\frac{1}{1-\beta}$, the expected work at time t is $\frac{nR}{(1-\beta)^2} \cdot \frac{1}{t}$. Summing over m edge arrivals yields a harmonic series [8, Theorem 4]:

$$\sum_{t=1}^m \frac{nR}{(1-\beta)^2} \cdot \frac{1}{t} \leq \frac{nR}{(1-\beta)^2} \ln m.$$

3.3.3 Personalized PageRank Walk. Certain scenarios require a *personalized* form of PageRank that express node relevance with respect to a user, topic, or entity [59]. Whereas the *global* PageRank random walk teleports to a node uniformly at random with a restart probability, the *local* PPR random walk restarts from a certain locality of a seed node w or a small set of seed nodes.

Algorithm 1: PPRBIGWALK

Input: Z (walk segment storage), w (query-aware dummy seed node), L (required PPR walk length), β (damping factor)

```

1  $P_w \leftarrow [w]$  // Start walk at  $w$ 
2  $C_w(v) \leftarrow 0$ 
3  $u \leftarrow w$  // The current last node of the walk
4  $\text{Fetched} \leftarrow \emptyset$  // Fetched nodes indicator

5 while  $|P_w| < L$  do
6   if  $\text{UNIFORM}(0, 1) \geq \beta$  then
7      $P_w \leftarrow P_w + [w]$  // Reset walk to  $w$ 
8      $u \leftarrow w$ 
9   else
10    if  $z(u) \neq \emptyset$  then
11       $\hat{z} \sim z(u)$  // Sample a walk segment from  $u$ 
12       $z(u) \leftarrow z(u) \setminus \hat{z}$ 
13       $P_w \leftarrow P_w + [\hat{z}]$  // Extend big walk
14      Increment  $C_w(v)$  by the visit-counter for each  $v$  in  $\hat{z}$ .
15       $P_w \leftarrow P_w + [w]$  // Reset walk to  $w$ 
16       $u \leftarrow w$ 
17    else
18      if  $u \in \text{Fetched}$  then
19         $(u, r, v) \sim \mathcal{N}^-(u)$  // Sample an out-neighbor  $v$ 
20         $\mathcal{N}(u) \leftarrow \mathcal{N}^-(u) \setminus (u, r, v)$ 
21         $P_w \leftarrow P_w + [v]$ 
22         $u \leftarrow v$ 
23      else
24        Fetch copy of walk segments  $z(u)$ , visit counters and outgoing
        edges  $\mathcal{N}^-(u)$  of  $u$  from  $Z$ 
25        Fetched  $\leftarrow \text{Fetched} \cup u$  // Mark  $u$  as fetched

Output:  $P_w, C_w$ 

```

Algorithm 1 [8] uses the Monte Carlo PPR to efficiently and incrementally maintain the top- k nodes by seed-based PPR scores in an evolving graph. For each seed node w , the algorithm generates a PPR random walk P_w that reaches a specified length L to guarantee the accuracy of the top- k result, balancing random exploration with the use of precomputed walk segments. The walk begins at a seed node w and is stored in a list P_w . At each step, it resets to w if a random number drawn from a uniform distribution in $[0, 1]$ is below the teleportation probability $(1 - \beta)$, to avoid entrapment in a local region. Otherwise, if an unused walk segment starting from the last visited node u exists in memory, it is appended to P_w , effectively reusing precomputed paths to reduce computations, and the walk resets to w to maintain the PPR structure. Otherwise, if u has been previously visited, the walk continues naturally by selecting a random outgoing edge from u and appending the next node v to P_w ; if u has not been previously visited, the algorithm fetches data about the walks and neighbors of u before proceeding. This iterative process continues until the length of P_w reaches L . By leveraging precomputed walk segments, the algorithm efficiently scales PPR walks to large-scale graphs with reduced calculations and memory overhead by reusing previously computed walks.

3.3.4 Other incremental PPR algorithms. Algorithm 1 has been extended to support multiple graph updates in each time step [26], enhancing efficiency. Similarly, IMCPR [33] incrementally adjusts previously computed PageRank scores as the graph evolves. Other approaches [22], leverage indexes to improve the efficiency of approximate single-source PPR by storing and maintaining random walks under graph changes. While these methods maintain PPR

scores in dynamic graphs, they do not adapt in response to changes in the PPR seed nodes. This issue, that we solve, prevents the adoption of these methods in query-adaptive summarization.

3.3.5 Network Distribution. Most real-world graphs are *scale-free*, i.e., follow a power-law distribution where a few hub entities have very high degrees. Under certain assumptions, a power-law graph's in-degree and PageRank follow a similar distribution [34], $\pi_i \propto i^{-\alpha}$, with $0 < \alpha < 1$, also expressed as $\pi_i = \eta i^{-\alpha}$, where η is a normalization factor. Since PageRank sums up to one, it follows that $\eta \sum_{i=1}^n i^{-\alpha} = 1$, hence [8]:

$$\pi_i \approx \frac{1 - \alpha}{n^{1 - \alpha}} i^{-\alpha} \quad (3)$$

The expected count of visits to the k^{th} ranked node in a length- L walk is $c = L\pi_k$, hence $L = \frac{c}{\pi_k}$. By Equation (3), we get [8, Lemma 6]:

$$L = c \cdot \frac{n^{1 - \alpha}}{1 - \alpha} k^\alpha = \frac{c}{1 - \alpha} k \left(\frac{n}{k}\right)^{1 - \alpha} \quad (4)$$

3.3.6 Estimating the top k nodes. To guarantee the quality of the results of Algorithm 1, we lower-bound the length L of a PPR walk P_w so that a top- k entity is visited at least c times in expectation [8]. As the distribution exponent α equals the damping factor β [34], we replace $1 - \alpha$ in Equation (4) by the teleportation probability $\epsilon = 1 - \beta$, hence the required length L is:

$$L = \frac{c}{1 - \beta} k \left(\frac{n}{k}\right)^{1 - \beta} \quad (5)$$

4 PROBLEM DEFINITION

We observe that prior KG summarization methods either consider a static data graph and query workload, or, in case they admit a dynamic nature, they do so either only with respect to the data or only the queries. However, in real-world applications such as personalized query answering, recommendation systems, and real-time analytics, both the underlying KG data and user interests are evolving with time. A KG summarization method should adapt to these changes; otherwise, it is bound to produce summaries that are both *outdated*, i.e., fail to reflect the latest information, and *irrelevant*, i.e., not aligned with the evolving user interests, and hence require full recomputation with each update. To address these challenges, we aim for *adaptive KG summaries* being:

- **Online:** efficiently adapting to changes in current information (facts, nodes, or relations) in real-time.
- **Incremental:** progressively incorporating evolving information rather than recomputing a whole summary.

Formally, given a discrete-time dynamic KG, $\mathcal{G}_{\mathcal{T}} = (\mathcal{E}_{\mathcal{T}}, \mathcal{R}_{\mathcal{T}}, \mathcal{X}_{\mathcal{T}}, \mathcal{L}_{\mathcal{T}}, \mathcal{L}_{\mathcal{E}}, \mathcal{T})$, with snapshots $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_T$ at time intervals $\mathcal{T} = t_1, t_2, \dots, t_T$, and query workload $\mathcal{W} = \{Q_t\}_{t \in \mathcal{T}}$ we aim to extract an *online, incremental, structure- and query-aware series of summaries* $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_T$, in which each summary $\mathcal{S}_t = (\mathcal{E}_S, \mathcal{R}_S, \mathcal{X}_S)$ is a subgraph of $\mathcal{G}_t$, i.e., $\mathcal{E}_S \subseteq \mathcal{E}_t$, $\mathcal{R}_S \subseteq \mathcal{R}_t$, and $\mathcal{X}_S \subseteq \mathcal{X}_t$. Further, the number of triples in the summary is bounded by b , typically defined as a fraction p of the number of edges in the graph, i.e., $b = |\mathcal{R}_S| = p|\mathcal{R}_t|$. For each summary $\mathcal{S}_{t-1}$ computed at time step $t - 1$, we aim to derive an updated summary $\mathcal{S}_t(\mathcal{S}_{t-1}, \Delta\mathcal{G}_t)$ as a function of $\mathcal{S}_{t-1}$ and the updated graph $\Delta\mathcal{G}_t$ representing the

changes in nodes, edges, and labels from snapshot $t-1$ to t , aiming to achieve high utility with respect to user queries while maintaining storage within a bound and low computational overhead. Following prior work [32, 48], we quantify the utility of a summary S_t using the F1 score over the query workload $\mathcal{W}$, i.e., the *harmonic mean* of precision (fraction of returned answers that are correct) and recall (fraction of correct answers that are returned). Accordingly, at each time t , we aim to compute a summary $S_t = (\mathcal{E}_S, \mathcal{R}_S, \mathcal{X}_S)$ of size b that maximizes the F1 score over $\mathcal{W}$:

$$\arg \max_{S_t} F1(S_t) \quad \text{subject to} \quad |\mathcal{X}_t| = b \quad (6)$$

Directly optimizing Equation (6) is infeasible, as it requires knowing future queries issued after time t . In particular, while guarantees on $A(S_t)$ are impossible against an arbitrary workload, as an adversary could always select queries disjoint from the summary, we optimize S_t for workloads that follow a local discovery pattern. The locality assumption under the random surfer model effectively captures both user preferences [7] and local communities [2].

4.1 Probabilistic interpretation.

The objective in Equation (6) admits a probabilistic interpretation. We associate each node $v \in \mathcal{E}$ with a probability p_v representing its relevance to a query workload $\mathcal{W} = \{Q_t\}_{t \in \mathcal{T}}$, via, e.g., the number of times the node is queried, user preferences, or node semantics. Then $C = \sum_v p_v$ is the total expected number of relevant nodes and $A(S) = \sum_{v \in S} p_v$ is the expected number of relevant nodes in a subset $S \subseteq \mathcal{E}$ of k nodes. The expected precision and recall of S are $\mathbb{E}[P(S)] = \frac{A(S)}{k}$ and $\mathbb{E}[R(S)] = \frac{A(S)}{C}$, hence the expected F1 score becomes:

$$\mathbb{E}[F1(S)] = \frac{2 \cdot A(S)}{C + k}. \quad (7)$$

As C and k are fixed given workload $\mathcal{W}$, the objective reduces to:

$$\arg \max_{S_t} A(S_t) \quad \text{subject to} \quad |\mathcal{X}_t| = b \quad (8)$$

This probabilistic interpretation provides a convenient abstraction. While Equations (6) and (8) optimize over summaries S_t consisting of b edges, Equation (7) requires the k nodes with the highest relevance probabilities. Section 5.2 shows how to construct a summary of b edges induced by (and, more generally, incident to) the top- k nodes, provided that $k \leq b$.

5 SANKOFA

We present our summarization method, SANKOFA. While the probabilities p_v introduced in Section 4.1 are not known in advance, as they require knowledge of the query workload, we surmise that they follow the distribution induced by PPR. This choice stems from prior KG summarization methods [32, 48] and is justified as a structural prior on the latent query distribution by the *locality* assumption that future queries likely occur in the neighborhood of the query history. PPR represents the Maximum Likelihood Estimate (MLE) of user interest under a random-surfer model, which captures user preferences [7] and local community boundaries [2]. However, such prior methods compromise either on query awareness or on supporting graph updates, or both. To overcome these

limitations, we rearchitect the prior Monte Carlo-based PPR estimation for dynamic graphs (Algorithm 1) [8] to incorporate updates on the query-aware seed nodes w .

Algorithm 2: SANKOFA

Input: $T = [t_1, t_2, \dots]$ (timesteps), $G = \{G_t = (\mathcal{E}_t, \mathcal{R}_t, \mathcal{X}_t, \mathcal{L}_{\mathcal{E}_t}, \mathcal{L}_{\mathcal{R}_t})\}_{t=t_1}^{|T|}$ (dynamic graph), $\mathcal{W} = \{Q_t\}_{t=t_1}^{|T|}$ (queries), β (damping factor), k (number of top nodes), c (visit threshold parameter), R (walk segments per node), $|win|$ (walk window size), b (summary edge budget)

```

1   $S_t \leftarrow \emptyset$  // Graph Summary
2   $Z_t \leftarrow \emptyset$  // Storage of walk segments
3   $\hat{C}_t(v) \leftarrow 0$  // Global frequency counter
4  for  $t$  in  $T$  do
5      Determine dummy seed node  $w_t$  for set of query nodes  $Q_t$ 
6      if  $t = t_1$  then
7          // Collect and store walk Segments */
8          for  $v \in \mathcal{E}_t$  do
9              Collect  $R$  walks starting from  $v$ .
10             Store walks and their node-visit counts in storage  $Z_t$ .
11             Increment global visit counters  $\hat{C}_t$  with walk counters.
12         else
13             // Update stored Walk Segments to reflect  $G_t$  */
14             Obtain changes  $R_{\mathcal{E}_t}, R_{\mathcal{X}_t}, A_{\mathcal{E}_t}, A_{\mathcal{X}_t}$ 
15              $Z_t \leftarrow Z_{t-1}; \hat{C}_t \leftarrow \hat{C}_{t-1}$ 
16             for  $v \in R_{\mathcal{E}_t}$  do
17                 In  $Z_t$ , remove walks starting from  $v$ .
18                  $\hat{C}_t(v) \leftarrow 0$ 
19             for  $x_{ijk} \in R_{\mathcal{X}_t}$  do
20                 Update walks in  $Z_t$  containing  $x_{ijk}$ .
21                 Update  $\hat{C}_t$  to reflect updated walks.
22             for  $a \in A_{\mathcal{E}_t}$  do
23                 Collect and add  $R$  walks to  $Z_t$  for  $a$ .
24                 Increment global visit counters  $\hat{C}_t$  with walk counters.
25         // Collect ranking statistics for summary construction */
26          $L \leftarrow \frac{c}{1-\beta} k \left( \frac{|\mathcal{E}_t|}{k} \right)^{1-\beta}$  // Length of PPR walk
27          $P_{w_t, C_{w_t}} \leftarrow \text{PPRBIGWALK}(Z_t, w_t, L, \beta)$  // Algorithm 1
28          $\tilde{P} \leftarrow \bigcup_{i=t-|win|}^t \{u \mid u \in P_{w_i}\}$  // Entities from latest walks
29         for  $u \in \tilde{P}$  do
30              $\tilde{C}(u) \leftarrow \sum_{i=t-|win|}^t C_{w_i}(u) \cdot \gamma^{t-i}$  // Sums of decayed counts
31          $\text{Top}k_t(\mathcal{E}) \leftarrow \arg \max_{v \in \tilde{P}}^k \tilde{C}(u)$  // Get top nodes based on counts
32         for  $u \in \text{Top}k_t(\mathcal{E})$  do
33              $\pi_{w_t}(u) \leftarrow (1-\beta) \frac{\tilde{C}(u)}{\sum_{v \in \tilde{P}} \tilde{C}(v)} + \beta \frac{\hat{C}_t(u)}{\sum_{v \in \hat{C}_t} \hat{C}_t(v)}$  // Sankofa rank
34         // Extract summary  $S_t$  (See Section 5.2) */
35         Obtain and add internal triples to  $S_t$  // Section 5.2
36         if  $|S_t| < b$  then
37             Obtain and add boundary triples to  $S_t$  // Section 5.2
38         if  $|S_t| < b$  then
39             Obtain and add fallback triples to  $S_t$  // Section 5.2

```

Output: S : Query-adaptive incremental top- k summaries

Rather than recomputing a random walk from scratch, Algorithm 1 splits random walks into segments starting from nodes in the walk, such that only the segments affected by a change on the graph need be updated. Likewise, SANKOFA maintains walk segments to efficiently determine, at each time t , the list of k nodes with the largest PPR scores relative to the query workload $Q_1, \dots, Q_t$. We build the summary S_t by selecting K edges induced by the top- k nodes. Whereas, in general, a query may return a subgraph, we assume for the sake of simplicity that it returns a *result set* of entities matching the variable $\star$. This assumption naturally covers structured query answers, treating all nodes in a matching as part of the *result set*.

SANKOFA builds and continuously maintains two key building blocks: (i) a Monte Carlo-based long Personalized PageRank walk P_w

for each query node w (Section 5.1), and (ii) a query-aware incremental summary using the top- k nodes ranked by PPR from P_w (Section 5.2). Algorithm 2 outlines the complete solution.

5.1 Maintaining Random Walks

At time t , given a TKG $\mathcal{G}_{t_1}, \mathcal{G}_{t_2}, \dots, \mathcal{G}_t$ and a query workload $\mathcal{W} = \{Q_1, \dots, Q_t\}$, we link all nodes appearing in the query answers at time t to a dummy node w_t , which represents the starting point of the random walks. By utilizing only query nodes, our method remains agnostic to node labels, edge types, and schema definitions.

In the first time step t_1 , we store R random walk segments of expected length is $\frac{1}{1-\beta}$ for each entity $v \in \mathcal{E}_t$ and initialize counters $\hat{C}_t(v)$ of the number of visits to each v across walk segments (Lines 6–10). The precomputed segments facilitate efficient access in later operations. Fetching a node v retrieves its R precomputed walks along with its neighbors.

At each subsequent time step (Line 11), we determine a dummy seed node w_t connected to query nodes and identify the changes $R_{\mathcal{E}_t}$, $R_{\mathcal{X}_t}$, $A_{\mathcal{E}_t}$, and $A_{\mathcal{X}_t}$ between consecutive snapshots (Line 12). $R_{\mathcal{E}_t}$ and $R_{\mathcal{X}_t}$ capture deleted nodes and edges, whereas $A_{\mathcal{E}_t}$ and $A_{\mathcal{X}_t}$ record the new nodes and edges. We maintain these sets incrementally as updates arrive, introducing negligible runtime cost. Based on these changes, we update the walk segments Z_{t-1} at time $t-1$ to reflect the modifications introduced in the snapshot $\mathcal{G}_t$. The update occurs in three steps: (i) For $R_{\mathcal{E}_t} \subseteq \mathcal{E}_t$, we remove all walks that start from each node $v \in R_{\mathcal{E}_t}$ and decrease counter $\hat{C}_t(v)$ to reflect the deletion (Lines 15–16). (ii) For each deleted edge $x_{ijk} \in R_{\mathcal{X}_t}$, we truncate any walk containing this edge at its first occurrence and expand the truncated walk to restore the required walk segment length (Line 18), allowing the use of newly added edges $A_{\mathcal{X}_t}$ during expansion. We then update $\hat{C}_t$ to reflect the revised visit counts (Line 19). (iii) For each $v \in A_{\mathcal{E}_t}$, we add R new random walk segments (Line 21) and update $\hat{C}_t$ to account for the additional visits introduced by those (Line 22).

Next, we compute the required walk length L (Line 23) and collect a random walk of length L starting at w_t by Algorithm 1 (Line 24), which incrementally computes the Monte Carlo PPR by tracking the number of visits to each entity in the walk P_{w_t} using the visit counter C_{w_t} . We also collect all nodes present in walks P_{w_t} in previous timesteps up to t in $\tilde{P}$ (Line 25).

To gradually phase out older information and emphasizing more recent information, we weight a count at timestep j by γ^{t-j} , where i is the latest timestep and $\gamma \in (0, 1)$ a decay factor that controls how rapidly the influence of older walks decays (Line 27): smaller γ values emphasize recent activity, while values closer to 1 preserve long-term memory.

We extract the top- k most frequently visited nodes in the PPR walk $Topk(\mathcal{E})$ (Line 28) from the decayed counter $\tilde{C}(u)$, normalize their counts through division by their sum to obtain PPR $\pi(w, v)$, ensuring that values sum up to 1 (Line 30), and use them to filter the relevant entities in the subsequent ranking step. Notably, the top- k ranking does not consider the dummy seed node w_t .

For the nodes in $P_{w_t} \setminus Topk(\mathcal{E})$, which are subsequently used to identify boundary (Section 5.2) and fallback triples (Section 5.2), we assign scores based on their normalized ranks within P_{w_t} . Nodes in the parent graph $\mathcal{E} \setminus P_{w_t}$ have a PPR of zero; we therefore derive

their scores from the global PageRank. These global scores are utilized only when the candidates in $P_{w_t} \setminus Topk(\mathcal{E})$ are insufficient to meet the summary requirements.

As we explain in Section 5.1.1, we derive weights for PPR values from these normalized ranks, considering $Topk(\mathcal{E})$, $P_{w_t} \setminus Topk(\mathcal{E})$ and $\mathcal{E} \setminus P_{w_t}$ and build a summary S_i targeting a high number of triples per node in $Topk(\mathcal{E})$ (Line 31), followed by boundary triples (Line 33) and fallback triples (Line 35). Figure 2 presents an overview of the SANKOFA framework. SANKOFA maintains random walks derived from the query logs and graph snapshots, and constructs a summary based on the $Topk(\mathcal{E})$ nodes. We outline the summarization process in Section 5.2.

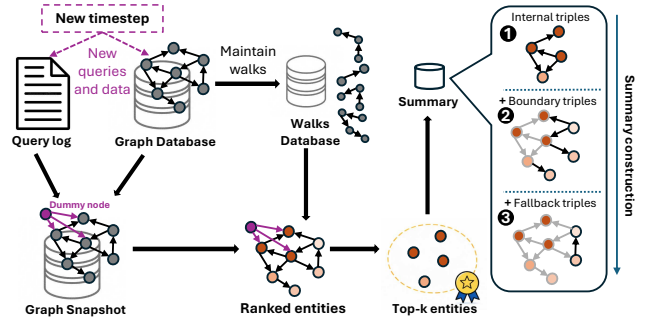


Figure 2: SANKOFA Framework.

5.1.1 Reweighting PPR. We experimentally observed that the summary quality achieved using the PPR alone is unstable for small graph and summary sizes. That is due to two reasons. First, when the big walk P_w is shorter, some important nodes may not be visited often enough to appear in the top- k list. Second, PPR inherently favors nodes near the seed node w [68] as the teleportation with probability $1 - \beta$ may inadvertently lead to overrepresenting less important nodes and underrepresenting globally important nodes that are farther away from w .

To mitigate this effect and ensure that structurally central nodes are not overlooked due to being far from seed nodes, we *reweight* each seed node's SANKOFA PPR score w using its global PageRank, with the damping factor β serving as the reweighting coefficient:

$$\begin{aligned} \pi_w(u) &= (1 - \beta) \cdot \pi(w, u) + \beta \cdot \pi(u) \\ &= (1 - \beta) \frac{\tilde{C}_{w_t}(u)}{\sum_{v \in \mathcal{E}} \tilde{C}_{w_t}(v)} + \beta \frac{\hat{C}_t(u)}{\sum_{v \in \mathcal{E}} \hat{C}_t(v)} \end{aligned}$$

where $\pi(w, u)$ is the MC PPR rank of u using seed w , $\tilde{C}_{w_t}(u)$ is the normalized decayed cumulative frequency of node u in the big walk over the recent sliding window, i.e., $\tilde{C}_{w_t}(u) = \sum_{j \in \text{window}} C_{w_t}(u) \cdot \gamma^{t-j}$ for the current timestep t , and $\pi(u)$ is the global PageRank for node u , i.e., the count of the number of times a node appears in all R precomputed random walks, normalized into a probability distribution. Using the damping factor β for calculating PageRank in Equation (1) as a reweighting coefficient makes sense intuitively: for $\beta = 1$, there is neither teleportation nor reweighting. Besides, we experimentally determined that different values do not significantly improve performance.

LEMMA 1. Let $\pi(w, u)$ be a probability distribution over nodes induced by a MC PPR walk starting from node w (hence $\sum_{u \in \mathcal{E}} \pi(w, u) = 1$), and $\pi(u)$ be a global PageRank distribution (hence $\sum_{u \in \mathcal{E}} \pi(u) = 1$). For any $\beta \in [0, 1]$, the linear equation:

$$\pi_w(u) = (1 - \beta) \cdot \pi(w, u) + \beta \cdot \pi(u) \quad (9)$$

also defines a probability distribution, i.e., $\sum_{u \in \mathcal{E}} \pi_w(u) = 1$.

PROOF. Summing both sides of (9) over all $u \in \mathcal{E}$ we get:

$$\sum_{u \in \mathcal{E}} \pi_w(u) = (1 - \beta) \sum_{u \in \mathcal{E}} \pi(w, u) + \beta \sum_{u \in \mathcal{E}} \pi(u).$$

Since $\pi(w, u)$ and $\pi(u)$ are probability distributions, by assumption, $\sum_{u \in \mathcal{E}} \pi(w, u) = 1$ and $\sum_{u \in \mathcal{E}} \pi(u) = 1$. Hence

$$\sum_{u \in \mathcal{E}} \pi_w(u) = (1 - \beta) \cdot 1 + \beta \cdot 1 = 1$$

Each component of this equation is generally cheap to maintain as the graph evolves. Node frequencies update in $O(1)$ per timestep, the decayed windowed totals are updated in constant time using simple multipliers, and the global PR values are computed once in each timestep from the existing R random walks and then used over an already pruned set of nodes. Entities outside $\tilde{C}_{w_t}$ are ignored in the MC PPR calculation, as they do not contribute to the summation; the rank of such entities, $\pi_w(u) = (1 - \beta) \cdot 0 + \beta \cdot \pi(u) = \beta \cdot \pi(u)$, is only computed if explicitly invoked. These properties render the update rule exceptionally lightweight, enabling true online operation in each timestep. $\square$

Empirically, we found that this reweighting is especially beneficial for small summaries, which select only a tiny fraction of nodes. In such cases, missing a globally important node can noticeably reduce quality. For larger summaries, however, this concern is less critical. As the number of nodes grows, the Monte Carlo PPR walk naturally explores more of the graph, increasing the chance of visiting globally important nodes. Likewise, larger summaries include more nodes in the top- k set, capturing more influential nodes. In these cases, reweighting has only a minor effect, or none at all.

Nevertheless, by combining the MC PPR estimate with global PageRank, we maintain a balance between capturing local query-aware importance and global structural influence. Thereby, our method provides reliable top- k summaries across a variety of graph sizes and summary scales, without overemphasizing either local or global information. Given our focus on large graphs in Section 6.2, we investigate the effects of weights in improving results for small summaries. We find that this mechanism provides high accuracy for small summaries and significantly improves the results for a conventional choice of β . Further, in Section 5.2 we apply the same formulation to determine the scores of non-top- k nodes during summary construction.

5.2 Computing the Summary

In each timestep t , we build a compact summary including node and edge labels $\mathcal{S}_t = (\mathcal{E}_t, \mathcal{R}_t, \mathcal{X}_t, \mathcal{L}_{\mathcal{E}_t}, \mathcal{L}_{\mathcal{R}_t}, f_t, g_t, h_t)$ that preserves the most important structural information of the knowledge graph within a fixed budget of triples b . We utilize the weighted top- k nodes $Topk(\mathcal{E})$ and their corresponding MC scores $\pi(w_i, v)$, which rank nodes by importance to the user. We construct the summary

incrementally, beginning with the strongest and most direct connections among the top- k nodes and then expanding outward to other regions only when necessary to meet the desired size. As Figure 2 shows, summary construction occurs in the following three phases.

1 Internal triples selection. We commence the construction by collecting all candidate triples that interact with the top- k nodes. For each $v \in Topk(\mathcal{E})$, the algorithm examines all outgoing edges in which the top- k node appears as subject:

$$\Gamma^-(v) = \{(v, \cdot, \cdot) \mid (v, \cdot, \cdot) \in \mathcal{X}_t, v \in Topk(\mathcal{E})\},$$

and incoming edges, where it appears as object:

$$\Gamma^+(v) = \{(\cdot, \cdot, v) \mid (\cdot, \cdot, v) \in \mathcal{X}_t, v \in Topk(\mathcal{E})\}.$$

We assume that relevant triples are obtainable from $\mathcal{G}_t$ via the implemented storage Z . We obtain outgoing triples using an adjacency dictionary and ingoing triples using a reverse index for all subjects and predicates pointing to a given entity. This bidirectional exploration ensures that the summary captures structural information flowing into and out of the top- k region, providing a balanced representation of important entities.

If we restricted the exploration to outgoing triples only, the summary would ignore parts of the local neighbourhood structure around the top- k nodes [29]. By exploring both directions, we avoid structural imbalance in the resulting summary and preserve the true connectivity patterns of the original graph. We aggregate all collected triples into a candidate pool

$$C = \bigcup_{v \in Topk(\mathcal{E})} (\Gamma^+(v) \cup \Gamma^-(v))$$

Within this candidate pool, we first isolate the set of *internal* triples, whose both endpoints lie in the top- k set:

$$C_{\text{int}} = \{(s, \cdot, o) \in C \mid s, o \in Topk(\mathcal{E})\}$$

For a summary budget b , if $|C_{\text{int}}| \leq b$, we include all the internal triples in the summary. If $|C_{\text{int}}| > b$, we extract the most important internal triples in linear time, without sorting, by partial selection on a score function being the sum of importance weights $\text{score}((s, \cdot, o)) = \pi_{w_t}(s) + \pi_{w_t}(o)$ assigned to a triple $(s, \cdot, o)$. The summary graph thus forms a core structure that captures the strongest and most important connections among the top- k nodes.

As the scoring of triples is based on node importance estimated via the MC PPR, and these estimates are reliable only for nodes that are visited a sufficient number of c times during the big random walk, we prioritize these internal triples. This strategy expands the summary from nodes with well-estimated importance to incorporate triples involving nodes with more uncertain or less accurately estimated scores. Thereby, the backbone of the summary reflects the most trustworthy structural information, while allowing expansion to less certain regions if additional triples are required to reach the budget.

2 Boundary triples selection. If the number of directly relevant triples is insufficient to meet the summary budget, we add to the remaining candidate triples, the *boundary* triples $C_{\text{bnd}} = C \setminus C_{\text{int}}$ which have *exactly one* node inside the top- k set. To score these edges, we expand the rank calculation to all entities u that belong to triples in the boundary set C_{bnd} and lie outside the set of entities in

recently considered walks $\tilde{P}$; for such entities, we calculate $\pi_w(u) = (1 - \beta) \cdot \pi(u)$ and use it in the calculation of triple scores.

Next, we select a subset of the boundary triples to use since the set of boundary triples C_{bnd} can be orders of magnitude larger than the set of internal triples. To ensure that we only select what is needed from boundary triples, we find the difference between the budget b and the number of internal triples C_{int} . Further, to ensure we efficiently prioritize highly weighted triples while avoiding duplication, we adopt the Efraimidis–Spirakis algorithm [15] for weighted random sampling without replacement, which bases the probability that a triple is selected on its relative score. For each boundary triple with score a_i , we draw r_i from a random distribution $r \sim U(0, 1)$ and recompute the score as $q_i = \frac{-\ln r_i}{a_i}$. This formulation of q_i ensures that boundary triples of higher weight are assigned lower, numerically stable keys, as $-\ln(\cdot)$ is decreasing in $[0, 1]$ and $r \in [0, 1]$. To avoid the overhead of explicit sorting, we extract the smallest keys in priority order by a partial selection algorithm, limited to the remaining budget size.

We add these boundary triples to the summary in small batches while also ensuring that each sampled triple is included only if it has not already been added. Altogether, boundary triples contribute additional local context to the summary, capturing how top- k nodes connect to their immediate neighbors while retaining the focus around the important entities.

③ Additional triples selection In case the internal and boundary triples do not suffice to reach the summary budget, we include a fallback step that includes any remaining triple from the parent graph, i.e., $C_{\text{fb}} = \mathcal{X}_t \setminus (C_{\text{int}} \cup C_{\text{bnd}})$. By the same procedure as for the boundary triples, we determine the number of triples to be added, apply the Efraimidis–Spirakis sampling algorithm, and partially select the highest-weighted triples. For triple scoring, we further expand the rank calculation to cover all remaining entities. Yet, now we consider only non-top- k nodes whose triples have not been included in the summary, i.e., $\text{score}((s, \cdot, o)) = \pi_{w_s}^{\text{ntk}}(s) + \pi_{w_o}^{\text{ntk}}(o)$ until we reach the desired summary size. Thereby we guarantee that the summary always satisfies the fixed budget $b < |\mathcal{X}_t|$, even in sparse graphs and situations where the top- k region has sparse structure.

Eventually, the resulting summary graph prioritizes the most important nodes and their strongest connections and ensures the summary budget is achieved in a controlled and interpretable manner. The final summary preserves the core structure of the original knowledge graph in a compact form centered around the entities deemed most relevant to the user. Moreover, the summary is continuously communicated to the user at the end of each timestep, ensuring an online process.

5.2.1 Bound on the top- k . The summary size budget also implies a corresponding constraint on the number of top- k entities required. Too few top- k hinder approximations, while too many top- k introduce computational overhead. We derive an upper bound on the top- k based on the density of the graph.

In a knowledge graph snapshot $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{X})$ with $n = |\mathcal{E}|$ entities (nodes) and $m = |\mathcal{X}|$ triples (edges). The graph’s average degree $d = m/n$ reflects the average number of triples an entity participates in. For large d , the graph is *dense*, hence entities tend to be well connected and a small number of top entities effectively summarizes most interactions. For small d , the graph is *sparse*,

hence connectivity is limited and a summary requires more entities. We therefore define an adaptive upper bound on the number of top- k entities we need for a summary as the ratio of the count of all entities to average global connectivity.

PROPOSITION 1. Consider a KG snapshot $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{X})$ with $n = |\mathcal{E}|$ entities and $m = |\mathcal{X}|$ triples. If each top entity accounts for d triples on average, then an adaptive approximate upper bound of the number of top entities needed to summarize the graph is

$$k_{\text{max}} = \frac{n^2}{m} \quad (10)$$

PROOF. By definition, average degree $d = m/n$ represents global connectivity. It follows that the ratio of the count of all entities to average global connectivity is $k_{\text{max}} = \frac{n}{d} = \frac{n^2}{m}$. $\square$

The value of k_{max} computed as above may still be relatively large, raising high computational cost without raising summary quality. We use a scaling factor $r \geq 1$, which we determine empirically (see Section 6.3), we set the number of top entities as:

$$k = \frac{k_{\text{max}}}{r} \quad (11)$$

5.3 Complexity Analysis

We consider three phases of a single iteration of SANKOFA: walk segment maintenance, PPR calculation by Algorithm 1, and summary construction. As observed in [8], the cost of updating walk segments is only a logarithmic factor larger than the initialization cost. Initializing nR , with $n = |\mathcal{E}|$, walk segments takes $O\left(\frac{nR}{1-\beta}\right)$ time, while maintaining them over m edge updates takes $O\left(\frac{nR \ln m}{(1-\beta)^2}\right)$. The number of walks is $R \propto \ln n$, hence $R = O(\ln n)$ [8, Theorem 1] and the total expected update cost becomes $O\left(\frac{n \ln n \ln m}{(1-\beta)^2}\right)$. According to [8, Corollary 8], the cost of Algorithm 1 is bounded by the number of segment fetches, which are dominant when memory access is required. When all fetches are in memory, the dominant cost becomes the maintenance of visit counters C_w over the L steps of the walk. In this case, $L = \Theta\left(k^\beta n^{1-\beta}\right) = O(n)$, for $k < n$ and $0 \leq \beta < 1$. Summary construction requires time in $O(n + m)$. Ranking entities and triples using partial selection takes $O(n)$ and $O(m)$ time, respectively, assuming $k \ll n$ and $b \ll m$. The cost of edge selection thus does not exceed $O(m)$, as the scanning and sampling of internal, boundary, and fallback edges are linear in the number of edges. Combining all phases, the total runtime per iteration is $O\left(\frac{n \ln n \ln m}{(1-\beta)^2} + m\right)$, for sufficiently large n and m . This complexity represents an extreme and unrealistic scenario where random walks traverse the entire graph, which is not encountered in practice.

6 EXPERIMENTS

This section presents the results of our experimental study.

Datasets: We employ three large publicly available datasets. Table 2 outlines data characteristics.

- **DBPedia** [5] is an encyclopedic KG extracted from Wikipedia, featuring relationships among people, locations, and organizations. We use five snapshots² spanning the period from 2010

²<http://downloads.dbpedia.org/>

to 2013, each represented as a directed graph. We map all entities and relations to unique integer identifiers. The final graphs contain 2 to 4 million triples and 1 to 2 million entities.

- **IMDB**³ is a KG about movies, series, actors, and directors, and their relations. We use the IMDB-30SP version [65], built from 30 daily snapshots obtained from Jan 22 to Feb 20, 2019.
- **Wikidata** [60] is a large knowledge graph featuring real-world entities and their relationships. We obtain the full historical logs from Wikidata dumps⁴ and the complete edit history with temporal granularity of a second. As in [63], we retain only entities that are present in Wikipedia to ensure data quality and prune potentially noisy entries. We construct 10 temporal snapshots from April 2024 with daily granularity.

Table 2: Statistics on employed datasets.

Dataset	Entities	Relations	Triples	Timestamps
DBpedia (DBP)	1–2M	4	2–4M	5
IMDB-30SP (IMDB)	160–300K	14	500–700K	30
Wikidata (WIKI)	6M	1310	37M	10

Queries. For each graph snapshot, we generate 200 user queries following the framework in [48]. The workload includes 5 to 15 topics per snapshot; between timesteps, 40% of these topics shift, while 10% are random. We split queries into 80% for training and 20% for testing. We evaluate test queries under two regimes, *present* and *past*. In the *present* regime, we evaluate each method on unseen queries in the current timestep, while in the *past* regime we test it on unseen queries aggregated from all previous timesteps. To assess how well models retain past information, we additionally weight test queries by timestep employing the same scheme that we used for training.

Baselines. We compare SANKOFA against three query-aware summarization methods, using the provided implementations.

- **GLIMPSE** is a static personalized KG summarization method that infers user interests from queries and constructs a compact budget-constrained summary. Queries serve as seeds for expansion to related entities and relations. The summary is obtained by optimizing inferred utility under a fixed size budget via a greedy iterative triple selection strategy based on marginal gain⁵. To support dynamic settings, we retrain GLIMPSE at each timestep using queries from a two-step window. The time complexity of GLIMPSE is $O(m)$ [48, Theorem 2].
- **iSUMMARY**⁶ is a static personalized method that yields KG summaries by expanding from a user-specified entity, selecting the top- κ most query-relevant neighbors, and connecting them via frequently queried paths. To adapt iSUMMARY to our setting, we generalize it to a global personalized summary by aggregating entity-level summaries over all entities in the query workload. We replace the node-based κ constraint with a triple-based budget allocated proportionally to entity query frequency, yielding a unified, query-aware summary that captures the most frequently queried regions of the knowledge graph. The time of this approach is $O(b^2 + |Q| \cdot |E_Q|^2)$ [56]. While effective for low-hop

queries, the quadratic dependence on the workload’s query nodes $|E_Q|$ leads to poor scalability as query complexity grows, making it impractical for large query logs.

- **APEX**²⁷ is an adaptive extreme summarization method for dynamic query workloads on static graphs. It models user interests via a *heat diffusion* process and supports incremental updates. To accommodate evolving graphs, we extend APEX² to reflect structural changes by reusing and adjusting heatmaps from previous timesteps. We also extended APEX² to support multi-source query workloads, as its default version updates interest scores for only a single query per timestep. The time complexity is $O(c|E_Q|^2 \log(c|E_Q|))$ [32, Theorem 4.2], where c is the average number of h -hop neighbors. Designed for 1-hop queries, it scales poorly with larger or more complex workloads.

6.1 Query Answering

We commence our evaluation by assessing each method’s quality in terms of F1 score on test queries and its runtime. With DBP and IMDB, we measure runtime on a 378GB RAM machine running Ubuntu 18.04.6 LTS, equipped with an Intel Xeon E5-2687V v3 CPU (40 threads) @3.1GHz. For the larger WIKI graph, we use a 1TB RAM machine running Ubuntu 24.04.2 LTS, equipped with an Intel Xeon Silver 4316 CPU (80 threads) @2.3GHz. The walk component of Algorithm 2 (Lines 5–30) is implemented in C++ using pybind11 and compiled using g++ with o2 optimization, while summary construction (Lines 31–35) runs in Python to match state-of-the-art graph representations. We average results over 10 simulated query workloads for DBP and IMDB and 5 for WIKI, owing to its higher runtime. We use 20 random walks per node with walk length 5 in all datasets. We set the damping factor to $\beta = 0.9$. We justify these choices in Section 6.2.

6.1.1 Present-time and past queries. We evaluate performance using unseen test queries from the current timestep and a weighted test set aggregating unseen queries from previous timesteps, while applying a decay factor of γ^{t-j} to past queries, consistent with the training procedure described in Section 5.

We focus on the effect of varying the number of query topics on DBP and IMDB with summary sizes 0.001% and 0.5%, respectively. As Table 3 shows, SANKOFA achieves high F1 across topic settings for both the present and past variants, closely followed by GLIMPSE, as it effectively balances responsiveness to recent queries while retaining relevant information from earlier timesteps. Most importantly, it outperforms GLIMPSE on DBP while GLIMPSE does comparatively better in IMDB. In contrast, APEX² and iSUMMARY perform better in the present setting, suggesting that they are more suited to recent queries and less effective at incorporating historical context.

6.1.2 Summary size ratio. We assess the model in terms of the ratio of summary size to graph size, where the summary size determined in the first timestep is retained in subsequent timesteps to ensure constant size across timesteps, regardless of graph growth. Figure 3 presents the runtime and F1 across timesteps for different summary sizes on DBP. SANKOFA and GLIMPSE retain stability across summary sizes, whereas APEX² and iSUMMARY struggle to maintain performance of small summaries as the graph evolves. In terms of

³<https://datasets.imdbws.com/>

⁴<https://dumps.wikimedia.org/>

⁵<https://github.com/tsafavi/glimpse-summary>

⁶<https://github.com/giannisvasiliou/iSummary>

⁷<https://drive.google.com/file/d/apex>

Table 3: Average F1 score across timesteps in *present* vs *past* queries for 10 simulated workloads; 0.001% summary for DBP and 0.5% summary for IMDB; best method highlighted.

Topics	Dataset	Present				Past			
		SANKOFA	GLIMPSE	APEX ²	iSUMMARY	SANKOFA	GLIMPSE	APEX ²	iSUMMARY
5 (few topics)	DBP	0.865 $\pm$ 0.087	0.662 $\pm$ 0.114	0.576 $\pm$ 0.187	0.647 $\pm$ 0.195	0.770 $\pm$ 0.107	0.553 $\pm$ 0.143	0.577 $\pm$ 0.164	0.520 $\pm$ 0.181
	IMDB	0.947 $\pm$ 0.031	0.947 $\pm$ 0.031	0.947 $\pm$ 0.032	0.883 $\pm$ 0.074	0.881 $\pm$ 0.141	0.582 $\pm$ 0.224	0.520 $\pm$ 0.226	0.387 $\pm$ 0.246
10 (many topics)	DBP	0.574 $\pm$ 0.093	0.438 $\pm$ 0.083	0.407 $\pm$ 0.089	0.420 $\pm$ 0.091	0.582 $\pm$ 0.119	0.390 $\pm$ 0.129	0.354 $\pm$ 0.094	0.412 $\pm$ 0.093
	IMDB	0.949 $\pm$ 0.030	0.949 $\pm$ 0.030	0.949 $\pm$ 0.030	0.884 $\pm$ 0.056	0.810 $\pm$ 0.190	0.605 $\pm$ 0.197	0.536 $\pm$ 0.199	0.411 $\pm$ 0.213
15 (most topics)	DBP	0.427 $\pm$ 0.088	0.388 $\pm$ 0.094	0.336 $\pm$ 0.090	0.396 $\pm$ 0.080	0.356 $\pm$ 0.086	0.316 $\pm$ 0.073	0.365 $\pm$ 0.079	0.306 $\pm$ 0.088
	IMDB	0.948 $\pm$ 0.032	0.948 $\pm$ 0.032	0.947 $\pm$ 0.032	0.876 $\pm$ 0.056	0.759 $\pm$ 0.227	0.601 $\pm$ 0.191	0.566 $\pm$ 0.167	0.395 $\pm$ 0.195

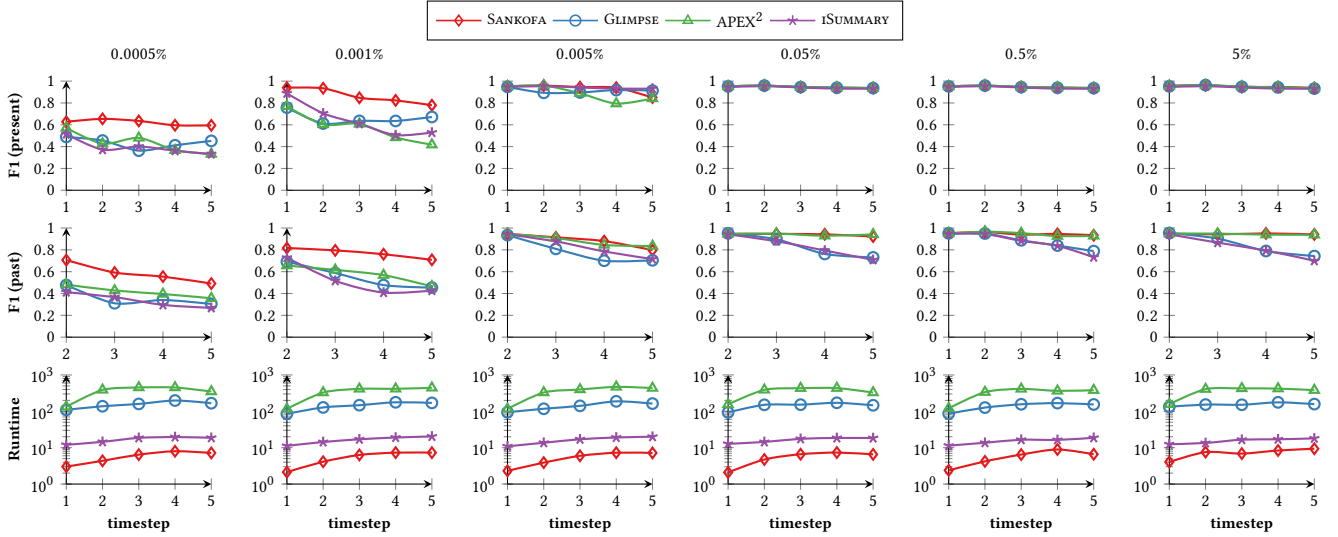


Figure 3: F1 vs summary size across timesteps, 5 topics, DBP: present queries (top), past queries (middle), runtime (bottom).

runtime, SANKOFA is the fastest, with average runtime below 10 seconds regardless of summary size, followed by iSUMMARY at about 20 seconds, GLIMPSE at 3 minutes, and APEX² at about 8 minutes.

Table 4: F1 score with a sliding window on past queries, 0.001% summary, DBP.

		Test window							
Timestep		1-2	2-3	3-4	4-5	1-2	2-3	3-4	4-5
	2	0.94				0.70			
	3	0.90	0.86			0.45	0.56		
	4	0.79	0.86	0.91		0.46	0.64	0.72	
	5	0.72	0.78	0.86	0.65	0.27	0.33	0.42	0.49
(a) SANKOFA									
Timestep		1-2	2-3	3-4	4-5	1-2	2-3	3-4	4-5
	2	0.51				0.57			
	3	0.42	0.44			0.49	0.49		
	4	0.36	0.39	0.39		0.43	0.57	0.74	
	5	0.42	0.42	0.45	0.34	0.1	0.27	0.42	0.45
(c) APEX ²									
Timestep		1-2	2-3	3-4	4-5	1-2	2-3	3-4	4-5
	2								
	3								
	4								
	5								
(d) iSUMMARY									

6.1.3 Memory Retention. We test the model on the recall of past query information. Table 4 presents the results of a sliding-window

experiment on past queries with a 0.001% summary. Rows indicate summaries at time step 2 to 5, while columns stand for test windows of consecutive time steps (1-2, 2-3, 3-4, 4-5). Each cell reports the F1 score of a summary evaluated on the given window.

SANKOFA consistently achieves high F1 scores, demonstrating resilience in both present and past settings. In contrast, GLIMPSE’s performance declines as the evaluation window moves away from the diagonal, consistently with its training procedure that emphasizes recent information. Although APEX² is an incremental method, its recall of past information is on par with iSUMMARY.

6.1.4 Scalability. We test the scalability of our method using three datasets of variant size: (a) IMDB (medium); (b) DBP (large); and (c) WIKI (very large). Figure 4 reports results for summary size 0.005% for IMDB, 0.001% for DBP, and 0.0001% for WIKI. SANKOFA compares favorably in F1 across datasets, on both present and past queries, and also maintains the lowest runtime among methods. In contrast, iSUMMARY is competitive only on present queries and declines in past queries on the large DBP and WIKI graphs, despite its relatively low runtime. APEX² and GLIMPSE exhibit less stable performance across all queries and also falter in terms of runtime.

6.1.5 Challenging workloads. Last, we study robustness against more adversarial queries. We design a workload with topics changing at every timestep, where the next query is at least 3 hops away from any previous topic node or any node in shortest path between

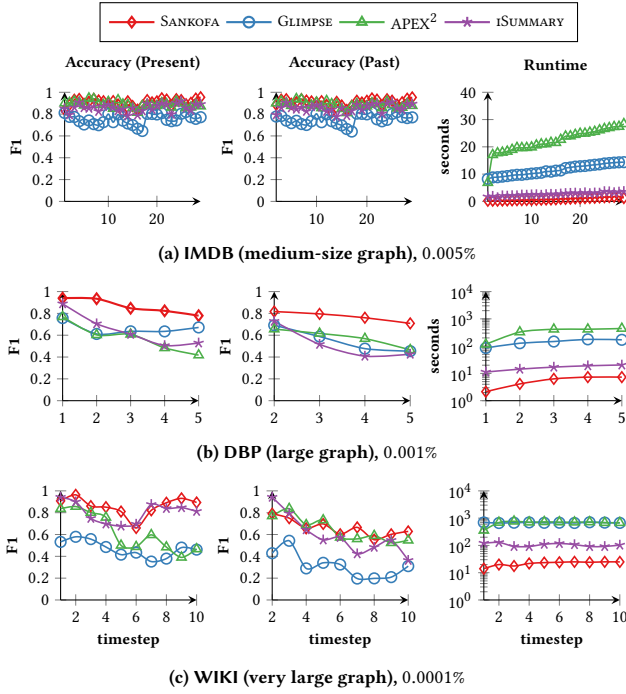


Figure 4: Scalability test on present (from current timestep, left) and past (from all previous timesteps, middle) unseen queries, summarization time (right).

topic nodes. From these topics we sample 200 2- or 3-hop queries. Figure 5 shows that SANKOFA maintains high quality summaries across timesteps, while the other methods cannot adapt promptly to the changes. Notably, APEX² outperforms iSUMMARY and GLIMPSE. While iSUMMARY is the fastest baseline, its exclusive reliance on shortest paths between query nodes causes significant overfitting, leaving it unable to capture relevant neighborhood structure.

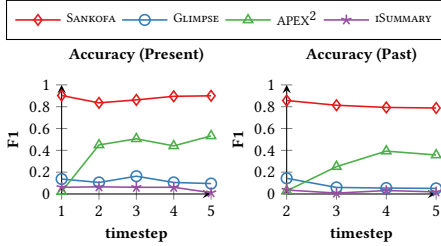


Figure 5: Hard queries, 0.5%-summary, DBP.

6.2 Different configurations

6.2.1 Robustness on updates. We assess SANKOFA’s performance under three update scenarios: (i) dynamic graph with dynamic queries (DD), our default setting; (ii) dynamic graph with static queries (DS); and (iii) static graph with dynamic queries (SD). As Figure 6 shows, SANKOFA demonstrates stronger stability across settings, while iSUMMARY performs best on SD, but is less competitive in the default DD. APEX², although designed for dynamic queries, struggles to be competitive in DD, as it assumes a single-source query per timestep. SANKOFA is thus robust across a variety of environments, in which the graph, the queries, or both are dynamic.

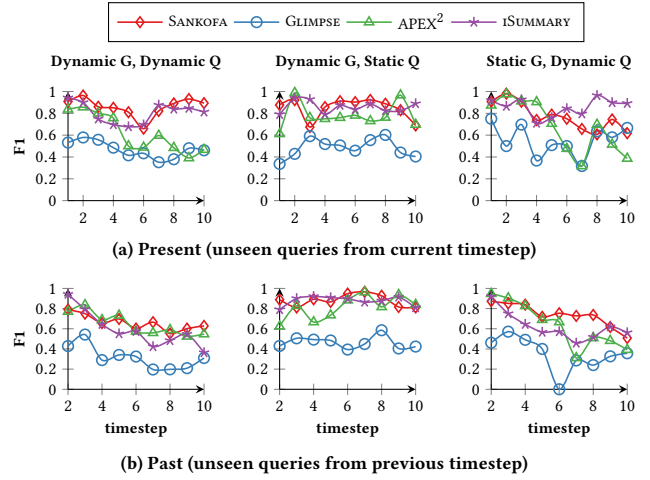


Figure 6: Robustness test, 0.0001%-summary, WIKI.

6.2.2 Effect of the decay factor. Next, we examine the behavior without a decay factor γ , which balances the retained past information with the influence of current queries. Section 6.3.1 details the decay factor choice with each graph. Here, we compare a SANKOFA variant without decay, SANKOFA-ND, to SANKOFA for a 0.005% summary on IMDB. Figure 7 shows that the stability of SANKOFA-ND drops significantly as time steps progress. We conclude that it is crucial, for the sake of long-term stability, to gradually phase out older information and give more weight to recent timesteps. This effect is particularly pronounced with dynamic graphs of many timesteps, while it may not be observed for some graphs and summary sizes.

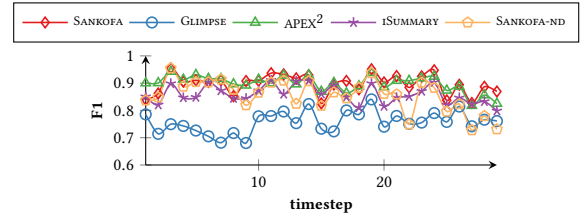


Figure 7: Effect of decay factor on present-time queries, IMDB, 0.005%-summary.

6.2.3 Increasing number of timesteps. We juxtapose the two most scalable algorithms, SANKOFA and iSUMMARY, over an extended time horizon on WIKI. As Figure 8 shows, both methods capture current updates and queries, but SANKOFA excels at retaining past query relevance. Because iSUMMARY’s runtime scales poorly with an expanding query log, it cannot maintain the full history during topic shifts, leading to inferior performance on past queries.

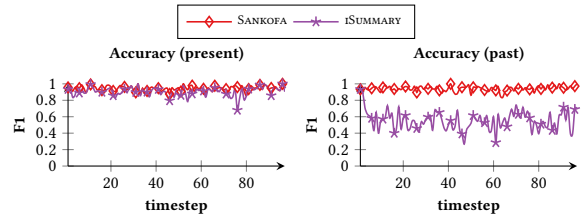


Figure 8: Long time horizon, WIKI-EXT, 0.5%.

6.2.4 Real queries. We evaluate how the methods behave on real-world WIKI queries. Since real graph exploration queries are scarce, we adapt organized exploratory sessions⁸ to a 16-query exploratory workload spanning June to August 2017 across 16 WIKI timesteps, using ‘Shifting Intent’ sessions (Category 6) from [4] ordered by date, to mimic a user workflow: starting with a query (Q_1), moving through related searches (Q_2 – Q_9) and incremental refinements (Q_{10} – Q_{13}), before shifting to a new topic (Q_{14} – Q_{16}). Q_3, Q_5, Q_6 lack topic entities, challenging the topic-to-object assumptions of existing baselines. To reflect local summary constraints, we apply result limits to the most prolific queries: 1 000 for Q_3, Q_5, Q_6 (originally 4.8M matches) and 20 for Q_{15}, Q_{16} (originally 400K). We evaluate present and past accuracy across the query log.

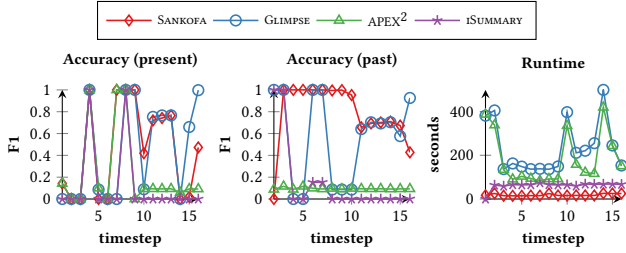


Figure 9: Accuracy on empirical queries WIKI, 5%.

Figure 9 shows that all methods struggle during early dynamic timesteps, yet only SANKOFA and GLIMPSE successfully predict entities during query refinements. APEX² and GLIMPSE exhibit runtime spikes as long intervals between queries accumulate updates on compressed sparse adjacency matrices. While SANKOFA attains superior performance on past queries, its quality declines slightly over time, since it does not dynamically adjust γ as the evaluation set grows; while γ can be learned by tracking query occurrences, we fixed it to 0.8 to ensure consistency across experiments.

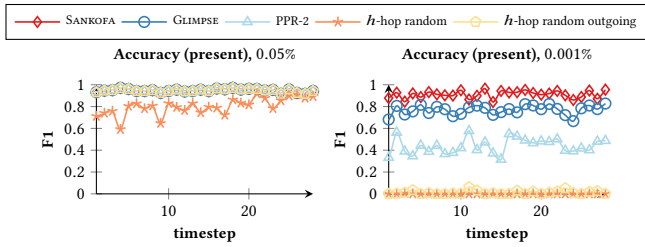


Figure 10: Choice of PPR, IMDB, 0.05% and 0.001%-summary.

6.2.5 Choice of PPR. We evaluate the choice of PPR against (1) *h-hop random*, which uniformly samples in and out neighbors; (2) *h-hop random outgoing*, which samples only out neighbors; and (3) PPR-2 [48]. As Figure 10 shows, *h-hop* methods fail on 0.001% summaries, and PPR-2 achieves only marginal success. SANKOFA delivers high quality summaries across regimes. As the dataset expands to a denser random-sampling pool over time, *h-hop* gains performance.

6.3 Hyperparameter Analysis

6.3.1 Decay factor γ . Sensitivity analysis indicates that SANKOFA consistently achieves high F1 scores with larger values of $\gamma > 0.6$. While lower decay factors $\gamma < 0.1$ can improve long term stability in

⁸<https://github.com/hartig/ExploratoryQueryingSessions>

settings with extended time horizons or restrictive summary sizes, the best rate depends on the setting. For the sake of consistency across experiments, we fix γ to 0.8.

6.3.2 Damping factor β . Table 5 reports the average F1 scores across 5 topics for a 0.001% summary on DBP with selected high-end values of the damping factor β . We observe that setting $\beta \leq 0.20$ leads to good performance, while ensuring stability on smaller summaries. In practice, for reducing runtime and allowing for a sufficient traversal length we set $\beta = 0.1$.

Table 5: Effect of damping factor β on DBP, 0.001% summary.

β	0.1	0.15	0.2	0.25	0.3
avg. F1	0.86	0.84	0.80	0.70	0.63

6.3.3 Number of top nodes k . Equation (11) sets the number of top nodes to be selected as $k = \frac{k_{\max}}{r}$. We empirically determined that setting $r = \frac{1}{10b}$ leads to a good balance between the budget b and the top- k selection. Using this value in the expression for k , we get $k = \frac{k_{\max}}{r} = 10b \cdot n^2/x$, where n is the number of nodes and x is the number of triples. For instance, a 0.1% summary has $b = 1000$, hence $r = 100$, while a 1% summary has $b = 100$, hence $r = 10$.

6.3.4 Number of walks R . Figure 11 shows that runtime grows with the number of walks on a 0.001% summary of DBP. Using 20 walks yields about a 1.4% accuracy improvement over 5-10 walks. It has thus sufficed to use 20 walks in our experiments.

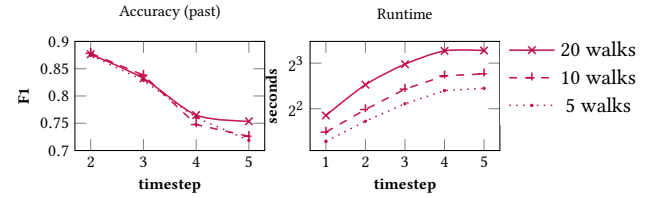


Figure 11: Effect of # of walks, 0.001% summary, 5 topics, DBP.

6.3.5 Node frequency c . Parameter c governs the number of times is to be counted to be ascertained in the top- k . As recommended in [8], we set $c = 2$. Our experiments indicate that this parameter does not affect results significantly.

7 CONCLUSION

We present SANKOFA, the first graph summarization method to incrementally maintain a summary over both dynamic graphs and shifting query workloads. By efficiently adapting a random walk PageRank approximation, SANKOFA achieves high F1 scores on future queries with a runtime under 25 seconds on the WIKI graph, which comprises 37 million triples. Incorporating global node weights further ensures robustness, even for extremely compact summaries containing only 0.0001% of the original triples. Future work will explore authenticating summaries [42], integrating semantic information and developing faster approximation schemes using random walk sketches.

ACKNOWLEDGMENTS

Ama Bembua Bainson and Kasper Overgaard Mortensen are supported by the Independent Research Fund Denmark (DFF).

REFERENCES

- [1] 2024. Information technology – Database languages – GQL. <https://www.iso.org/standard/76120.html>
- [2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *FOCS*. IEEE, 475–486.
- [3] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. 2017. Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR)* 50, 5 (2017), 1–40.
- [4] Marcelo Arenas, Enrico Franconi, Janik Hammerer, Olaf Hartig, Katja Hose, Laura Koesten, George Konstantinidis, Leonid Libkin, Wim Martens, Yuya Sasaki, et al. 2025. Exploring Exploratory Querying. *Proc. VLDB Endow.* 18, 13 (2025), 5731–5739.
- [5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A nucleus for a web of open data. In *ISWC*. Springer, 722–735.
- [6] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. 2007. Monte Carlo methods in PageRank computation: When one iteration is sufficient. *SINUM* 45, 2 (2007), 890–904.
- [7] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*. 635–644.
- [8] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *Proc. VLDB Endow.* 4 (2010), 173–184.
- [9] Caleb Belth, Xinyi Zheng, Jilles Vreeken, and Danai Koutra. 2020. What is normal, what is strange, and what is missing in a knowledge graph: Unified characterization via inductive summarization. In *WWW*. 1115–1126.
- [10] Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. 2019. Summarizing semantic graphs: a survey. *VLDB Journal* 28 (2019), 295–327.
- [11] Zhao Chang, Lei Zou, and Feifei Li. 2016. Privacy preserving subgraph matching on large graphs in cloud. In *SIGMOD*. 199–213.
- [12] Xi Chen, Yun Xiong, Siwei Zhang, Jiawei Zhang, Yao Zhang, Shiyang Zhou, Xixi Wu, Mingyang Zhang, Tengfei Liu, and Weiqiang Wang. 2024. Dtformer: A transformer-based method for discrete-time dynamic graph representation learning. In *CIKM*. 301–311.
- [13] Richard Cyganiak, David Wood, and Markus Lanthaler. 2014. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation. <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>
- [14] Alin Deutsch, Nadime Francis, Alastair Green, Keith W. Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoč, Mingxi Wu, and Fred Zemke. 2022. Graph Pattern Matching in GQL and SQL/PGQ. In *SIGMOD*. 2246–2258.
- [15] Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted random sampling with a reservoir. *Inf. Process. Lett.* 97, 5 (2006), 181–185.
- [16] Lukas Faber, Tara Safavi, Davide Mottin, Emmanuel Müller, and Danai Koutra. 2018. Adaptive personalized knowledge graph summarization. In *MLG Workshop*.
- [17] Philippe Fournier-Viger, Ganghuan He, Chao Cheng, Jiaxuan Li, Min Zhou, Jerry Chun-Wei Lin, and Unil Yun. 2020. A survey of pattern mining in dynamic graphs. *WIREs DMKD* 10, 6 (2020), e1372.
- [18] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *SIGMOD*. 1433–1445.
- [19] François Goasdoué, Paweł Guziewicz, and Ioana Manolescu. 2019. Incremental structural summarization of RDF graphs. In *EDBT*. 566–569.
- [20] Damien Graux, Diego Collarana, and Fabrizio Orlandi. 2021. Formal Concept Analysis for Semantic Compression of Knowledge Graph Versions. In *FCA4AI@IJCAL*.
- [21] Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 Query Language. W3C Recommendation. <https://www.w3.org/TR/sparql11-query/>.
- [22] Guanhao Hou, Qintian Guo, Fangyuan Zhang, Sibao Wang, and Zhewei Wei. 2023. Personalized pagerank on evolving graphs with an incremental index-update scheme. *PACMMOD* 1, 1 (2023), 1–26.
- [23] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *WSDM*. 105–113.
- [24] Richa Jalota, Daniel Vollmers, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo. 2021. LAUREN - Knowledge Graph Summarization for Question Answering. In *ICSC*. 221–226.
- [25] Hafiz Tayyeb Javed, Kifayat-Ullah Khan, Muhammad Faisal Cheema, Asaad Algarni, and Jeongmin Park. 2024. Instance-Based Lossless Summarization of Knowledge Graph With Optimized Triples and Corrections (IBA-OTC). *IEEE Access* 12 (2024), 5584–5604.
- [26] Zhang Junchao, Chen Junjie, Jiancheng Song, and Rong-Xiang Zhao. 2013. Monte Carlo based personalized PageRank on dynamic networks. *IJDSN* 9, 9 (2013), 829804.
- [27] Shinhan Kang, Kyuhan Lee, and Kijung Shin. 2022. Personalized graph summarization: formulation, scalable algorithms, and applications. In *IEEE ICDE*. IEEE, 2319–2332.
- [28] Sarang Kapoor, Dhish Kumar Saxena, and Matthijs van Leeuwen. 2021. Online summarization of dynamic graphs using subjective interestingness for sequential data. *DMKD* 35, 1 (2021), 88–126.
- [29] Abd Errahmane Kiouche, Julien Baste, Mohammed Haddad, and Hamida Seba. 2021. A Neighborhood-preserving Graph Summarization. *ArXiv abs/2101.11559* (2021).
- [30] Haridimos Kondylakis, Dimitris Kotzinos, and Ioana Manolescu. 2019. RDF graph summarization: principles, techniques and applications. In *EDBT*. 433–436.
- [31] Amy Nicole Langville and Carl Dean Meyer. 2006. *Google's PageRank and beyond - the science of search engine rankings*. Princeton University Press.
- [32] Zihao Li, Dongqi Fu, Mengting Ai, and Jingrui He. 2025. APEX²: Adaptive and Extreme Summarization for Personalized Knowledge Graphs. In *KDD*. ACM, 741–752.
- [33] Qun Liao, ShuangShuang Jiang, Min Yu, Yulu Yang, and Tao Li. 2017. Monte Carlo based incremental PageRank on evolving graphs. In *PAKDD*. 356–367.
- [34] Nelly Litvak, Werner RW Scheinhardt, and Yana Volkovich. 2007. In-degree and PageRank: why do they follow similar power laws? *Internet Mathematics* 4, 2-3 (2007), 175–198.
- [35] Jiaqi Liu, Qin Zhang, Luoyi Fu, Xinbing Wang, and Songwu Lu. 2019. Evolving knowledge graphs. In *IEEE INFOCOM*. IEEE, 2260–2268.
- [36] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *CSUR* 51, 3 (2018), 1–34.
- [37] Finlay MacLean. 2021. Knowledge graphs and their applications in drug discovery. *Expert Opin. Drug Discov.* 16, 9 (2021), 1057–1069.
- [38] Chifumi Nishioka and Ansgar Scherp. 2018. Analysing the evolution of knowledge graphs for the purpose of change verification. In *ICSC*. 25–32.
- [39] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. 2019. Industry-scale Knowledge Graphs: Lessons and Challenges. *CACM* 62, 8 (2019), 36–43.
- [40] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Stanford InfoLab. <http://ilpubs.stanford.edu:8090/422/>
- [41] Jeff Z. Pan, Simon Razniewski, Jan-Christoph Kalo, Sneha Singhania, Jiaoyan Chen, Stefan Dietze, Hajira Jabeen, Janna Omeljanenko, Wen Zhang, Matteo Lissandrini, Russa Biswas, Gerard de Melo, Angela Bonifati, Edlira Vakaj, Mauro Dragoni, and Damien Graux. 2023. Large Language Models and Knowledge Graphs: Opportunities and Challenges. *TGDK* 1, 1 (2023), 2:1–2:38.
- [42] Stavros Papadopoulos, Lixing Wang, Yin Yang, Dimitris Papadias, and Panagiotis Karras. 2011. Authenticated Multistep Nearest Neighbor Search. *IEEE Trans. Knowl. Data Eng.* 23, 5 (2011), 641–654.
- [43] Alexandros Pappas, Georgia Troullinou, Giannis Roussakis, Haridimos Kondylakis, and Dimitris Plexousakis. 2017. Exploring importance measures for summarizing RDF/S KBs. In *ESWC*. 387–403.
- [44] Carlos Eduardo Pires, Paulo Sousa, Zoubida Kedad, and Ana Carolina Salgado. 2010. Summarizing ontology-based schemas in PDMS. In *ICDEW*. 239–244.
- [45] Valentina Presutti, Lora Aroyo, Alessandro Adamou, Balthasar A. C. Schopman, Aldo Gangemi, and Guus Schreiber. 2011. Extracting Core Knowledge from Linked Data. In *2nd International Workshop on Consuming Linked Data (COLD)*.
- [46] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. 2021. Knowledge graph embedding for link prediction: A comparative analysis. *TKDD* 15, 2 (2021), 1–49.
- [47] Andrea Rossi, Donatella Firmani, Paolo Merialdo, and Tommaso Teofili. 2022. Explaining link prediction systems based on knowledge graph embeddings. In *SIGMOD*. 2062–2075.
- [48] Tara Safavi, Caleb Belth, Lukas Faber, Davide Mottin, Emmanuel Müller, and Danai Koutra. 2019. Personalized knowledge graph summarization: From the cloud to your pocket. In *ICDM*. 528–537.
- [49] Ansgar Scherp, David Richerby, Till Blume, Michael Cochez, and Jannik Rau. 2023. Structural Summarization of Semantic Graphs Using Quotients. *TGDK* 1, 1 (2023), 12:1–12:25.
- [50] Yuxuan Shi, Gong Cheng, and Evgeny Kharlamov. 2020. Keyword search over knowledge graphs via static and dynamic hub labelings. In *WWW*. 235–245.
- [51] Konstantinos Skitsas, Davide Mottin, and Panagiotis Karras. 2025. Pilos: Scalable Large-Subgraph Matching by Online Spectral Filtering. In *ICDE*. 1180–1193.
- [52] Qi Song, Yinghui Wu, Peng Lin, Xin Dong, and Hui Sun. 2018. Mining Summaries for Knowledge Graph Search. *IEEE Trans. Knowl. Data Eng.* 30, 10 (2018), 1887–1900.
- [53] Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. 2016. ABSTAT: Ontology-Driven Linked Data Summaries with Pattern Minimalization. In *ESWC Satellite Events*. 381–395.
- [54] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*. 697–706.
- [55] Georgia Iirini Trouli, Alexandros Pappas, Georgia Troullinou, Lefteris Koumakis, Nikos Papadakis, and Haridimos Kondylakis. 2022. Summer: Structural summarization for RDF/S kgs. *Algorithms* 16, 1 (2022), 18.

- [56] Giannis Vassiliou, Fanouris Alevizakis, Nikolaos Papadakis, and Haridimos Kondylakis. 2023. iSummary: Workload-based, personalized summaries for knowledge graphs. In *ESWC*. 192–208.
- [57] Giannis Vassiliou, Nikolaos Papadakis, and Haridimos Kondylakis. 2023. SummaryGPT: Leveraging ChatGPT for summarizing knowledge graphs. In *ESWC*. 164–168.
- [58] Giannis Vassiliou, Georgia Troullinou, Nikolaos Papadakis, and Haridimos Kondylakis. 2021. Wbsum: Workload-based summaries for RDF/S kbs. In *SSDBM*. 248–252.
- [59] Daniel Vial and Vijay G. Subramanian. 2019. On the Role of Clustering in Personalized PageRank Estimation. *ACM Trans. Model. Perform. Evaluation Comput. Syst.* 4, 4 (2019), 21:1–21:33.
- [60] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.
- [61] Xiaxia Wang and Gong Cheng. 2024. A survey on extractive knowledge graph summarization: applications, approaches, evaluation, and future directions. In *IJCAI*. 8290–8298.
- [62] Xiaxia Wang, Gong Cheng, and Evgeny Kharlamov. 2019. Towards Multi-Facet Snippets for Dataset Search. (2019), 1–6.
- [63] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. 2021. KEPLER: A Unified Model for Knowledge Embedding and Pre-trained Language Representation. *Trans. Assoc. Comput. Linguistics* 9 (2021), 176–194.
- [64] Gang Wu, Juanzi Li, Ling Feng, and Kehong Wang. 2008. Identifying potentially important concepts and relations in an ontology. In *ISWC*. 33–49.
- [65] Tianxing Wu, Arijit Khan, Melvin Yong, Guilin Qi, and Meng Wang. 2022. Efficiently embedding dynamic knowledge graphs. *Knowl. Based Syst.* 250 (2022), 109124.
- [66] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding. In *WWW*. 1271–1279.
- [67] Lyu Xu, Byron Choi, Yun Peng, Jianliang Xu, and Sourav S. Bhowmick. 2023. A Framework for Privacy Preserving Localized Graph Pattern Query Processing. *Proc. ACM Manag. Data* 1, 2 (2023), 129:1–129:27.
- [68] Renchi Yang, Jieming Shi, Xiaokui Xiao, Y. Yang, and Sourav S. Bhowmick. 2019. Homogeneous network embedding for massive graphs via reweighted personalized PageRank. *Proc. VLDB Endow.* 13 (2019), 670–683.
- [69] Yuhao Yang, Chao Huang, Lianghao Xia, and Chenliang Li. 2022. Knowledge graph contrastive learning for recommendation. In *SIGIR*. 1434–1443.
- [70] Jiasheng Zhang, Rex Ying, and Jie Shao. 2024. Online Detection of Anomalies in Temporal Knowledge Graphs with Interpretability. *Proc. ACM Manag. Data* 2, 6 (2024), 247:1–247:26.
- [71] Xiang Zhang, Gong Cheng, and Yuzhong Qu. 2007. Ontology summarization based on RDF sentence graph. In *WWW*. 707–716.
- [72] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *AAAI*.
- [73] Weiguo Zheng, Hong Cheng, Lei Zou, Jeffrey Xu Yu, and Kangfei Zhao. 2017. Natural language question/answering: Let users talk with the knowledge graph. In *CIKM*. 217–226.
- [74] Mussab Zneika, Claudio Lucchese, Dan Vodislav, and Dimitris Kotzinos. 2016. Summarizing Linked Data RDF Graphs Using Approximate Graph Pattern Mining. In *EDBT*. 684–685.