MoRAG: Bringing Graph-Based Retrieval-Augmented Generation to Real Smartphones

Anonymous authors
Paper under double-blind review

Abstract

Graph-based RAG organizes fragmented text into explicit knowledge structures for retrieval through entities, relations, and article summaries. However, directly bringing graph-based RAG to smartphones creates a mismatch between cloud-oriented graph methods and mobile constraints: compact on-device models are more likely to miss entities, predict incorrect relations, or merge distinct entity mentions, while multi-level graph summaries require frequent, large-context LLM invocations. We build MoRAG, to the best of our knowledge, the first graph-based RAG framework end-to-end deployed on commercial smartphones. MoRAG combines MRGraph, a two-layer mobile graph that preserves entities, relations, and source paragraphs; EvoNER, which incrementally learns unseen entity types and locally updates the graph; and ATPlanner, which uses one LLM call to guide hybrid retrieval under a fixed budget. On real smartphones, MoRAG outperforms prior mobile RAG baselines by up to 22.4 points in correctness, constructs knowledge graphs $13.77\times$ faster, and consumes only 1.36% of the energy compared with the latest lightweight methods. **Code & Demo:** `https://anonymous.4open.science/r/MoRAG-31B4/`.

1 Introduction

Retrieval-Augmented Generation (RAG) enables large language models (LLMs) to access external knowledge bases and updated information (Lewis et al., 2020; Gao et al., 2023; Izacard et al., 2023). By retrieving task-relevant evidence, RAG can improve factuality, support knowledge-intensive question answering, and reduce the need to encode all information into model parameters (Guu et al., 2020; Asai et al., 2024; Karpukhin et al., 2020). This paradigm is particularly attractive for smartphones, where private documents, personal histories, and local contextual knowledge often need to remain on device (Wang & Chau, 2024).

However, most existing mobile RAG systems rely primarily on vector retrieval, where queries are matched to document chunks according to semantic similarity (Park et al., 2025; Kang et al., 2026; Seemakhupt et al., 2024). While this design is efficient and easy to deploy, it lacks an explicit mechanism for connecting dispersed evidence across entities, relations, and documents, making complex multi-hop retrieval difficult (Yang et al., 2018). For example, answering a question such as *"Which hotel did I book after receiving the invitation from the conference organized by Professor X?"* may require connecting an email invitation, a calendar event, and a booking confirmation. The system must identify bridge entities, follow cross-document relations, and assemble evidence from multiple locations (Liu et al., 2025; Peng et al., 2025). Graph-based RAG instead organizes fragmented text into explicit knowledge structures for retrieval through entities, relations, and article summaries (Edge et al., 2024).

Unfortunately, directly bringing graph-based RAG to smartphones creates a mismatch between cloud-oriented graph methods and mobile constraints because these methods rely on three capabilities that compact on-device LLMs cannot reliably provide. First, graph-based retrieval relies on extracted entities and relations to connect relevant evidence across passages (Edge et al., 2024; Fan et al., 2025). However, compact on-device models can miss entities, predict incorrect relations, or merge distinct entity mentions. These errors can break or misdirect retrieval paths, making relevant passages harder to retrieve through the graph even when the original text is retained. Second, cloud-oriented methods requires to repeatedly invoke LLM to cluster and summarize extracted knowledge into multi-level

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

graphs (Guo et al., 2025; Li et al., 2024; Sun et al., 2024). Each higher-level node summarizes connected lower-level nodes to locate a semantic region, but on-device LLMs have limited usable context and weaker abstraction capabilities. Repeated summarization can introduce descriptions unsupported by source documents while incurring prohibitive latency and energy overhead due to frequent invocations. A natural question is whether educing the number of LLM invocations by limiting GraphRAG to two hierarchy levels can support mobile deployment. Our experiments on a subset of MultiHopQA show that this two-level GraphRAG variant only achieves an F1-score of **6.79%** and requires about $37\,182\,\mathrm{s}$ for on-device graph construction. This is substantially lower than original GraphRAG with an F1-score of 25.26%. Thus, reducing the number of levels alone can not addresses either inaccurate on-device knowledge extraction or LLM-based summary costs. Third, everyday smartphone use continuously generates new personal information. Therefore, graph structures need to be updated frequently, which is difficult to achieve on GraphRAG because hierarchical structures require cascading updates.

Based on these observations, we argue that on-device graph-based RAG systems should: (1) avoid relying on powerful models for entity and relation extraction; (2) use lightweight graph structures that preserve raw-text information under mobile deployment; and (3) support incremental updates and accurate online answering.

Following this insight, we propose MRGraph, a lightweight graph architecture containing only two layers dedicated for mobile deployment. Its base layer, the entity-paragraph graph (EP-graph) adds paragraph nodes to an entity-centric graph: entity connections support multi-hop reasoning, while paragraph nodes preserve source evidence when entities or relations are missing or incorrect. The higher layer is a routing index above the EP-graph connects each node to a cluster of semantically similar EP-graph nodes, enabling retrieval without an LLM-constructed multi-layer graph. In addition, the lightweight graph structure enables efficient incremental updates without rebuilding the entire graph, while an LLM-guided search retrieves relevant entities and paragraphs and avoids the unacceptable delays caused by frequent invocations to LLM in traditional methods.

Our contributions are summarized as follows:

- We identify three assumptions that hinder smartphone deployment of graph-based RAG: accurate compression of source information into entity–relation triplets, reliable construction of multi-level graph summaries, and incremental updating on graph. To deal with these problems, we introduce MRGraph, a two-layer mobile graph that combines an EP-graph for preserving entities, relations, and source paragraphs with a routing index for efficient retrieval.
- We build MoRAG, the first graph-based RAG framework which is end-to-end deployed on commercial smartphones based on MRGraph and introduce two modules for efficient graph updating and retrieval. **EvoNER** incrementally learns unseen entity types, amortizes LLM supervision over clusters of uncertain entities, and locally updates the EP-graph. **ATPlanner** converts a query into graph-search guidance in one call; lightweight modules then execute the search and assemble source-grounded information for answering.
- Real-smartphone experiments show that MoRAG demonstrates superior performance: up to **22.4** points higher QA correctness than the strongest baseline with the same LLM backbone, constructs graph **13.77**$\times$ faster, and reduces energy consumption to merely **1.36%** compared with MiniRAG.

2 PRELIMINARIES

2.1 BASIC CONCEPTS

Entity: An entity $e \in \mathcal{E}$ is a real-world object, concept, or named thing mentioned in text. Each entity node stores a canonical name, surface forms, and a type label from taxonomy $\mathcal{C}$. Entities serve as retrieval anchors linking textual mentions to relations, summaries, and source paragraphs, and are extracted by NER models, LLM-based extractors, or both (Yu et al., 2020; Edge et al., 2024).

Relation: A relation $r = (e_i, \tau, e_j)$ is a typed, directed semantic connection between entities $e_i, e_j \in \mathcal{E}$, where $\tau \in \mathcal{T}$; the set of relation edges is $\mathcal{R}$.

Evidence: Retrieved text chunks provide the grounding material for answer generation. A top-k embedding search selects candidates, and an optional cross-encoder re-ranks them before they enter the LLM context.

2.2 Previous RAG Methods

Vector-based RAG encodes document chunks into dense embeddings and retrieves the top-k most similar chunks for each query (Karpukhin et al., 2020). This design scales well, but each chunk is retrieved independently which has no explicit mechanism to chain evidence across documents.

Graph-based RAG addresses this by extracting entities and relations from documents and organizing them into entity-relation graphs that make cross-document connections explicit (Edge et al., 2024; Peng et al., 2025). The core intuition is that a graph exposes the *topology* of knowledge, who is connected to whom through what relationship, enabling retrieval to follow structured paths rather than relying solely on semantic similarity.

GraphRAG first splits documents into chunks and invokes an LLM on each chunk to extract entities and relations. Extracted elements are assembled into a graph $\mathcal{G} = (\mathcal{E}, \mathcal{R})$. A defining characteristic is the subsequent construction of a multi-level community hierarchy via Leiden clustering: for each community at each level, the LLM generates a textual summary from the entities and relations within that community, producing multi-granularity views of the same source material. At query time, GraphRAG provides two modes: **global search** applies map-reduce over pre-computed community summaries to synthesize dataset-level answers, while **local search** retrieves entities via embedding similarity and assembles a context window from their associated text units, relations, and summaries in a single pass. This design is powerful for query-focused summarization but expensive: community construction, multi-level summarization, and summary regeneration after graph updates all require frequent, large-context LLM invocations, which become prohibitive under mobile constraints.

RAG augments language models with non-parametric evidence retrieved at inference time (Lewis et al., 2020). Recent work improves this pipeline through adaptive retrieval and self-reflection (Asai et al., 2024), rationale-based evidence denoising (Wei et al., 2025), domain-specific retriever optimization with synthetic data (Shen et al., 2026), permutation-robust generation (Zhang et al., 2026), and extreme context compression (Cheng et al., 2024). Additional related work is discussed in Section F.

3 Method

As shown in Figure 1, MoRAG first constructs MRGraph from documents with entities, paragraph provenance, and relations, then retrieves grounded evidence at runtime through a structured plan. We next introduce MRGraph (Section 3.1), its incremental update mechanism EvoNER (Section 3.2), and the retrieval procedure ATPlanner (Section 3.3).

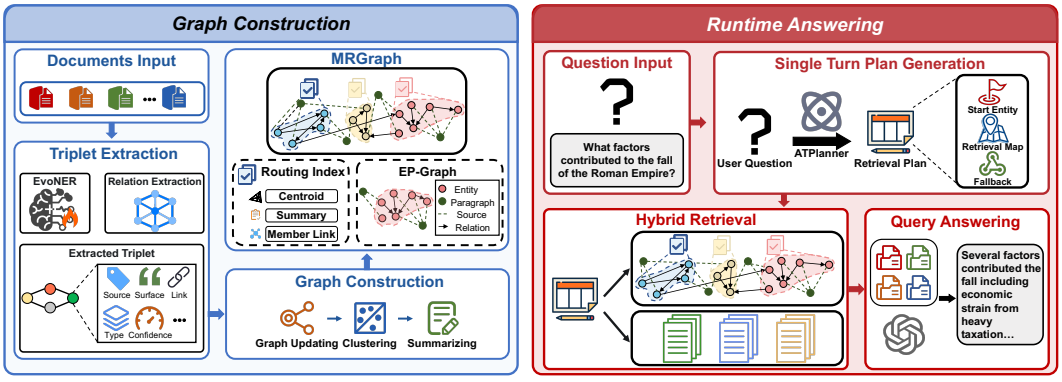


Figure 1: Overview of MoRAG's pipeline.

3.1 MRGraph

Graph is a crucial structure bridging knowledge extraction and runtime generation. To reduce LLM invocations during extraction while retaining source paragraphs as evidence, MRGraph uses two layers Figure 1: an entity-paragraph (EP) graph as the persistent evidence store and a routing index for fast access.

Base EP graph. The EP graph is a heterogeneous graph with two node types and two edge types:

$$\mathcal{G}_{\text{EP}} = (\mathcal{E}, \mathcal{P}, \mathcal{L}, \mathcal{R}), \tag{1}$$

where $\mathcal{E}$ is the set of entity nodes, $\mathcal{P}$ is the set of paragraph nodes, $\mathcal{L} \subseteq \mathcal{E} \times \mathcal{P}$ is the set of entity-paragraph inclusion links, and $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{T} \times \mathcal{E}$ is the set of directed relation edges with labels from taxonomy $\mathcal{T}$. An inclusion link $(e, p) \in \mathcal{L}$ records that entity e is mentioned in paragraph p; a relation edge $(e_i, \tau, e_j) \in \mathcal{R}$ records a typed connection τ from e_i to e_j.

Node and edge structures. Each entity node is a tuple

$$e = \big(\, n,\ H(n),\ \text{type} \in \mathcal{C},\ \mathbf{ve}_e \in \mathbb{R}^d,\ \text{cluster},\ \mathcal{P}_e \subseteq \mathcal{P} \,\big), \tag{2}$$

where n is the canonical name of entity; $H(n)$ is hash of n, which enables $O(1)$ deduplication; *type* constrains multi-hop expansion via type pruning; $\mathbf{ve}_e$ is a dense embedding produced by a lightweight on-device encoder, used for centroid-query similarity during routing; *cluster* points to the routing-index node containing e; and $\mathcal{P}_e = \{p \mid (e, p) \in \mathcal{L}\}$ are the back-links to all paragraphs mentioning e, enabling entity-to-evidence navigation directly.

Each paragraph node is a tuple

$$p = \big(\, \text{content},\ \text{symbolic},\ \mathbf{vp}_p \in \mathbb{R}^d,\ \text{seq} \in \mathbb{N},\ \mathcal{E}_p \subseteq \mathcal{E} \,\big), \tag{3}$$

where *content* is the source paragraph; *symbolic* replaces entity mentions with typed pointers for entity-span alignment and token efficiency; $\mathbf{vp}_p$ is a dense embedding enabling semantic paragraph retrieval via the paragraph vector index; *seq* records the ordinal position within the source document, used for runtime context expansion to neighboring paragraphs; and $\mathcal{E}_p = \{e \mid (e, p) \in \mathcal{L}\}$ are the back-links to all entities mentioned in p, enabling entity-centric expansion from a retrieved paragraph during multi-hop search.

Each relation edge is a tuple $r = \big(\, e_{\text{subj}} \in \mathcal{E},\ \tau \in \mathcal{T},\ e_{\text{obj}} \in \mathcal{E},\ w \in [0, 1] \,\big)$, where τ is matched against the planned relation phrase during plan-guided expansion; w is the confidence score from the relation extraction model, used as a soft filter during traversal.

Graph assembly and update locality. The EP graph is assembled incrementally. For each chunk, EvoNER detects entities, canonicalizes them via looking-up $H(n)$, creates a paragraph node, writes EP links, and inserts relation edges for pairs whose predicted confidence exceeds a threshold. This procedure is strictly local: it touches only the nodes and edges directly involved in the new document. The update is therefore append-and-merge.

Routing index. A flat EP graph would require $O(|\mathcal{E}| \cdot \bar{d}^H)$ operations for H-hop search with average degree $\bar{d}$. MRGraph therefore maintains a routing layer of **summary clusters** $\mathcal{S} = \{c_1, \ldots, c_K\}$, where each cluster groups entities proximal in embedding space and sharing the same domain. Each cluster c_i stores an L2-normalized centroid $\mathbf{vc}_i$ for dot-product lookup against a query embedding, a short summary text for text-based region matching, and back-links to its member entities. Clusters are formed and refreshed through embedding-based clustering, deferrable to background scheduling windows. Summaries for multiple clusters are generated in a single turn to minimize the total number of LLM invocations.

3.2 EvoNER

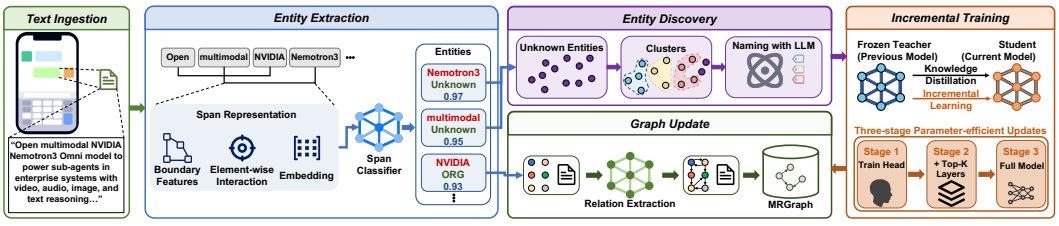


Figure 2: Schematic diagram of the graph-update mechanism.

Continuously updating MRGraph requires an entity extraction module that is both efficient and capable of incremental updates. Invoking an LLM on every text chunk incurs prohibitive latency

and energy consumption on smartphones. A lightweight named entity recognizer is cheaper, but is restricted to a fixed label space and cannot reliably process newly emerging entity types (Zhang et al., 2024). Consequently, new documents may be only partially written into the graph, leaving their entities and relations unavailable to subsequent retrieval.

EvoNER processes known types, discovers unseen types through amortized LLM supervision, backfills deferred facts, and adapts the recognizer. It has four steps (Figure 2): entity extraction, known-type graph updating, unknown-type discovery, and incremental model training.

Entity extraction: For chunk x, EvoNER scores candidate spans up to width W over deployed types $\mathcal{C}$ and `UNKNOWN`, producing $\mathcal{M}_x = \{(s_i, e_i, \ell_i, t_i, c_i)\}$ with boundaries, surface form, predicted type, and confidence. Mentions with confident boundaries and types enter the known-type branch; those with confident boundaries but uncertain types enter an unknown-entity buffer, preventing graph contamination while retaining them for recovery.

Known-type graph updating: EvoNER canonicalizes mentions, extracts relations, and updates the graph locally. Within a context window, normalized surface forms are candidates when one is a subset of the other; the longer is retained (e.g., *Jobs* and *Steve Jobs*), while mentions across windows remain separate. For each ordered canonical pair (e_i, e_j) in x, OpenNRE (Han et al., 2019) implements $\Phi_{\text{rel}}(x, e_i, e_j) = (\tau, w)$. Edges are retained when $w \geq \theta_{\text{rel}}$ and otherwise assigned $\tau = \texttt{NONE}$. EvoNER creates or retrieves paragraph p_x, merges entities by $H(n)$, writes $\mathcal{L}_x = \{(e, p_x) \mid e \in \mathcal{E}_x^{\text{known}}\}$, and inserts $\mathcal{R}_x = \{(e_i, \tau, e_j, w) \mid \tau \neq \texttt{NONE}, w \geq \theta_{\text{rel}}\}$. Entity–paragraph links are preserved regardless of relation confidence.

Unknown-type discovery: EvoNER separates entity detection from type induction by buffering mentions outside $\mathcal{C}$. Once distinct mentions exceed N, it clusters their embeddings with KMeans++. The top 40 mentions and contexts from each cluster are sent to the on-device LLM, which names coherent clusters and rejects inconsistent ones. Accepted labels propagate to cluster members; rejected clusters remain buffered for more evidence. For an accepted type $\hat{c}$, EvoNER retrieves stored source chunks, promotes affected mentions from `UNKNOWN`, combines them with known entities, and replays canonicalization, relation extraction for pairs involving a newly typed mention, and insertion of entity nodes, paragraph links, and confident relation edges.

Incremental model training: The LLM-assigned types and source contexts update EvoNER while the graph is backfilled, allowing future occurrences to enter the known-type branch without repeated LLM supervision. To prevent forgetting, EvoNER uses three stages:

$$\Theta_1 = \Theta_{\text{head}}, \qquad \Theta_2 = \Theta_{\text{head}} \cup \Theta_{\text{topK}}, \qquad \Theta_3 = \Theta, \tag{4}$$

where Θ_{head}, Θ_{topK}, and Θ denote the classification head, top-K Transformer blocks, and complete model. Stages 1–3 respectively train these parameter sets for S_1, S_2, and S_3 steps, increasing trainable capacity only if an earlier stage is insufficient.

The incremental objective is

$$\mathcal{L}_{\text{inc}} = \mathcal{L}_{\text{task}} + \lambda_{\text{KD}} \mathcal{L}_{\text{KD}}, \tag{5}$$

where λ_{KD} balances new-type adaptation and existing-type retention. For candidate spans $\Omega(x)$,

$$\mathcal{L}_{\text{KD}} = \frac{T_{\text{kd}}^2}{\sum_{r \in \Omega(x)} m_r + \epsilon} \sum_{r \in \Omega(x)} m_r \, \text{KL}\big(p_r^T \,\|\, p_r^S\big), \tag{6}$$

where p_r^T and p_r^S are teacher and student distributions at temperature T_{kd}, and m_r activates only when the frozen teacher is confident on an existing type. Thus, distillation preserves reliable deployed-type knowledge without forcing new entities into the old label space.

Each stage provides a valid stopping point, allowing training to be interrupted under battery, thermal, or scheduling constraints. Graph backfilling and model updates are executed asynchronously when the device has sufficient battery and foreground workloads are idle. After training, the student replaces the teacher for the next incremental round.

3.3 ATPLANNER

Mobile retrieval bottleneck: Cloud-oriented systems can support retrieval on graph with repeated LLM invocations or multi-level graph summaries. Both options are impractical under mobile

constraints. Repeated LLM navigation is expensive, while compact on-device models are less reliable at making a sequence of dependent search decisions. Because MRGraph uses a shallow routing index, unguided traversal can follow connected but irrelevant nodes, propagating seed-selection or relation-matching errors.

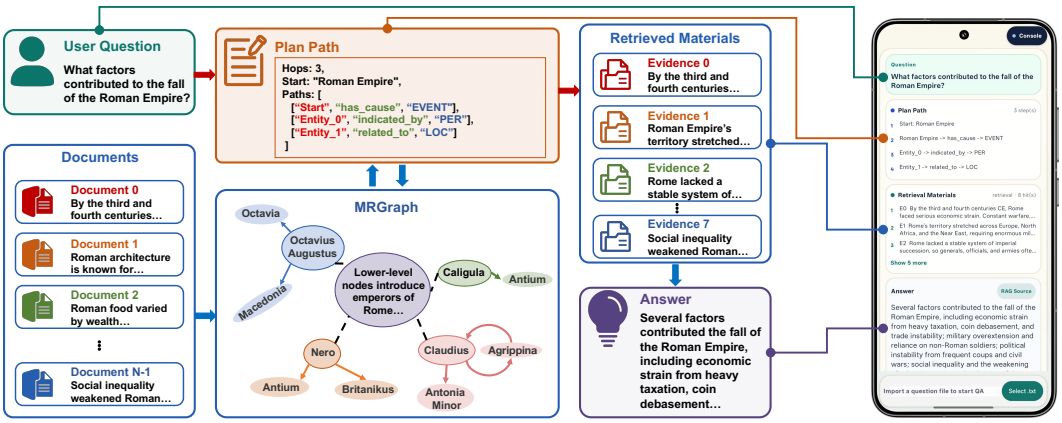


Figure 3: Workflow of plan-guided question answering.

Based on these problems, we proposed ATPlanner. As shown in Figure 3, ATPlanner invokes a compact planner once to predict a multi-hop retrieval structure; lightweight retrieval modules then execute the resulting plan under a fixed budget. The planner shares the answerer's backbone and adds only planner-specific LoRA parameters. After the plan is produced, graph traversal, text retrieval, fallback, and evidence assembly proceed without further LLM navigation. The training details of ATPlanner are provided in Section B.

Plan generation: Given a query q, ATPlanner generates a retrieval plan $\mathcal{P} = (\mathcal{E}_P, \mathcal{R}_P, H, B)$, where $\mathcal{E}_P$ is a set of entity slots, $\mathcal{R}_P$ forms a directed acyclic graph of relation steps, H is the longest path from a start slot to a final retrieval slot, and B is the token budget for the assembled evidence. Each entity slot $e_i = (s_i, t_i, o_i)$ contains a possibly empty surface phrase s_i, an expected entity type t_i, and a role $o_i \in \{\text{START}, \text{INTERMEDIATE}, \text{FINAL}\}$. The surface phrase is required for start slots and optional otherwise. Each relation step $(e_i, \rho_{ij}, e_j) \in \mathcal{R}_P$ specifies a relation phrase ρ_{ij} from e_i to e_j. Let $d(e_i)$ denote the depth of slot e_i and $\mathcal{A}_h = \{(e_i, \rho_{ij}, e_j) \in \mathcal{R}_P \mid d(e_i) = h - 1\}$ denote the steps executed at hop h. This representation supports multiple seeds and branching paths, which are useful for comparison and intersection questions. It also constrains each transition by the expected type of its target slot, making a best use of EvoNER's predicted entity type.

Plan execution: MoRAG first resolves the plan's start slots to seed entities through the routing index and descends to the corresponding regions of the EP-graph. At hop h, it expands the current frontier along relation edges that match the planned phrase ρ_{ij}. Candidate edges are ranked using predicate–plan similarity, relation confidence, and the query similarity of the neighboring entity. Neighbors that conflict with the target type t_j are pruned, and only a bounded frontier is retained. The traversed relations provide explicit bridge paths, while paragraph back-links attached to the selected entities recover their source evidence.

Fallback and evidence assembly: ATPlanner uses hybrid retrieval strategy against planning and retrieval errors. First, if the planner fails to produce a valid structured plan, MoRAG falls back to query-level text retrieval. Second, even with a valid plan, a global text branch retrieves additional paragraphs using lexical and dense similarity while excluding sources already covered by the planned search. Unlike local text repair, this branch does not update the multi-hop frontier; it contributes passages directly to the evidence pool. This hybrid retrieval method can therefore recover from an incorrect seed, a missing bridge relation, or an incomplete local neighborhood without another LLM invocation. Runtime grounding restores a bounded neighborhood around each selected paragraph from the same source document, reducing context loss caused by paragraph segmentation.

Asnwering: Finally, MoRAG combines the grounded graph evidence, plan-guided paragraph evidence, and global fallback evidence. It removes duplicate passages, ranks candidates by relevance and evidence utility, and packs them under budget B while reserving space for complementary graph and

text evidence. The answering model receives the original query together with this compact evidence set. The generated output records the supporting passage identifiers and whether retrieval contributed to the answer.

4 EXPERIMENTS

4.1 SETUP

We use compact LLM backbones from the Qwen3 (Qwen Team, 2025) and Ministral3 (Liu et al., 2026a) families, and compare against two groups of baselines: (1) **LLM-only** models without retrieval, and (2) **RAG** methods including Naïve RAG (Lewis et al., 2020), GraphRAG (Edge et al., 2024), InstructRAG (Wei et al., 2025), StableRAG (Zhang et al., 2026), TruthfulRAG (Liu et al., 2026b), xRAG (Cheng et al., 2024), and MiniRAG (Fan et al., 2025), for fair comparison.

MoRAG and NaïveRAG are executed entirely on real smartphones. Due to unacceptable graph construction time cost by LLM invocations other RAG methods are simulated in a cloud environment with same experiment settings. To demonstrate the on-device feasibility of MoRAG, we compared it with MiniRAG's graph construction process on smartphones. Details are provided in Appendix A.

We test MoRAG on two smartphones and two LLM scales, and the specific results are shown in Appendix E. Unless otherwise specified, OnePlus 15 and Qwen3-1.7B (S0) are used in subsequent experiments.

We conduct experiments on four QA benchmarks: TriviaQA (Joshi et al., 2017), Natural Questions (Kwiatkowski et al., 2019), HotpotQA (Yang et al., 2018), and MultiHopQA (Ho et al., 2020). We report **Correctness (Cr)** using an LLM-as-Judge protocol and **F1-score** for token-level overlap.

4.2 MAIN RESULTS: MORAG IMPROVES COMPACT BACKBONE QA UNDER MOBILE CONSTRAINTS

MoRAG is most effective on knowledge-intensive benchmarks: As shown in Table 1, MoRAG consistently achieves the strongest results on NQ, HotpotQA, and MultiHopQA across both backbone families, while its advantage on TriviaQA is less uniform. The gains are largest when answers require retrieving and composing dispersed evidence, whereas direct factoid questions can often be handled by powerful LLMs or simpler retrieval strategies. Compared with dense, graph-structured, instruction-guided, conflict-aware, order-robust, and context-compression baselines, MoRAG's two-layer graph exposes entity-level transitions while preserving source paragraphs for grounded answer synthesis. Meanwhile, its retrieval strategy decreases invocations of on-device LLM, which reduces the potentially misleading caused by hallucinations.

4.3 ANALYSIS FOR ON-DEVICE DEPLOYMENT

In this sections, we analyzed the operating characteristics of MoRAG on smartphones, including its memory and energy consumption during the graph construction and runtime generation. The graph construction study compares MoRAG with MiniRAG under the same configuration, since MiniRAG is the graph-based RAG baseline in our survey that is closest to mobile deployment.

Graph construction: Graph construction is the first practical bottleneck because a mobile graph is only useful if it can be constructed within a reasonable background-processing time. As shown in Table 2, under the S0 configuration MoRAG completes graph construction in $225.78\,\mathrm{s}$, whereas MiniRAG requires $3110.01\,\mathrm{s}$, a $13.77\times$ gap.

For the same document, MiniRAG requires $13.6\times$ more chunk-level ingestion time than MoRAG. Here, **chunk-level ingestion** refers to the full ingestion pipeline. Therefore, the runtime gap mainly reflects the cost of repeatedly applying chunk-level extraction and graph-writing operations under mobile constraints. Compared with MiniRAG, MoRAG updates MRGraph through EvoNER to entity, relation edges, and paragraphs, leading to substantially lower construction latency and energy consumption. This observation supports our design claim: under mobile constraints, graph-based RAG should avoid relying on repeatedly invocations of LLM for knowledge extraction. We note that chunk-level ingestion dominates total construction time for both systems, while graph updates

Table 1: Correctness (Cr, %) and F1 (%) on four QA benchmarks. Within each base-model block, the best result in each column is shown in **bold** and the second-best in underlined.

Base Model	Method	TriviaQA		NQ		HotpotQA		MultiHopQA	
		Cr	F1	Cr	F1	Cr	F1	Cr	F1
Qwen3									
LLM-only (no retrieval)									
	Qwen3-1.7B	22.10	21.52	9.10	10.17	14.15	13.00	22.00	19.18
	Qwen3-4B	28.70	37.66	23.15	19.95	24.00	22.29	30.05	29.29
	Qwen3-32B	**51.45**	42.26	40.15	35.92	33.25	30.56	31.54	32.69
Retrieval-augmented (1.7B)									
	Naïve RAG	47.00	36.31	62.24	37.23	41.55	37.80	45.66	34.58
	GraphRAG	32.90	32.31	39.85	35.73	30.25	26.46	30.05	25.26
	InstructRAG	43.15	22.67	41.30	34.99	33.90	26.15	13.90	12.94
	StableRAG	45.16	21.89	17.30	11.14	41.15	26.11	5.05	0.75
	TruthfulRAG	47.10	22.86	45.10	26.38	40.71	25.51	19.95	12.64
	xRAG	35.57	19.43	33.47	21.33	35.21	24.29	14.97	13.36
	MiniRAG	45.51	38.56	44.59	36.34	40.17	37.45	24.63	33.58
	MoRAG (ours)	50.00	**48.68**	**66.68**	**54.84**	**63.93**	**51.27**	**50.17**	**41.86**
Ministral3									
LLM-only (no retrieval)									
	Ministral3-3B	35.84	28.27	24.43	18.81	16.98	19.19	17.33	26.56
	Ministral3-14B	**60.70**	**60.87**	37.80	34.55	22.40	25.34	20.12	31.45
Retrieval-augmented (3B)									
	Naïve RAG	41.71	33.02	42.99	32.02	36.96	27.08	15.95	14.81
	GraphRAG	50.70	46.57	57.60	44.24	41.25	29.63	24.95	23.74
	InstructRAG	50.25	28.74	50.60	45.39	42.65	37.41	28.75	26.27
	StableRAG	15.95	10.86	27.35	18.11	44.03	32.94	13.71	9.07
	TruthfulRAG	47.86	25.52	53.00	31.31	47.86	35.38	43.20	33.67
	xRAG	33.32	22.16	35.67	25.57	39.78	30.31	21.93	11.54
	MiniRAG	47.51	29.51	43.79	34.13	35.83	33.89	35.96	20.47
	MoRAG (ours)	59.74	36.76	**63.86**	**54.07**	**66.74**	**46.08**	**57.60**	**43.37**

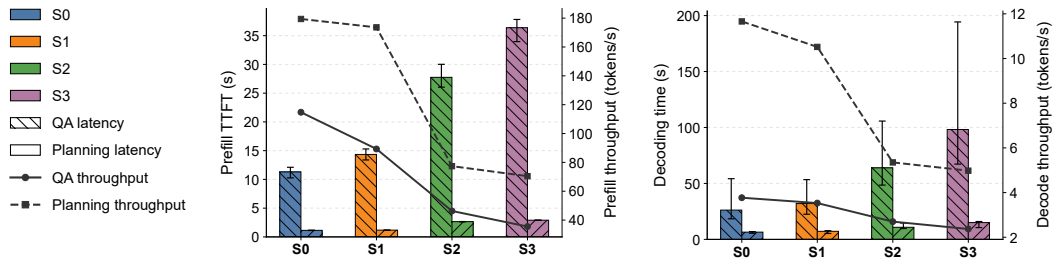
Table 2: Mobile graph construction comparison on S0 using the same articles. Runtime breakdown reports the percentage of total runtime.

Method	Total Runtime	Chunk Ingest.	Graph Update	Others	Energy Consumption
MoRAG	225.78 s	93.1%	6.0%	0.9%	0.1508 Wh
MiniRAG	3110.01 s	91.9%	6.0%	2.1%	11.12 Wh

account for only 6%. Although graph construction is not the dominant percentage of total runtime, the graph structure is responsible for the retrieval-quality gains observed in Table 1 and Figure 5.

Runtime answering: We next examine how MoRAG's query-time behavior changes across device and model size. In general, the measurements reveal a clear usability boundary. While both backbones can be executed on smartphones, they occupy very different latency regimes once the full retrieval pipeline is taken into account. Median QA prefill latency is 11.3 s for S0 and 14.3 s for S1, but rises to 27.8 s for S2 and 36.4 s for S3. The 1.7B model therefore remains the more deployable option under current mobile budgets. More broadly, the measured power remains relatively low because we prioritize stable long-term execution, allowing MoRAG to run as a persistent on-device system.

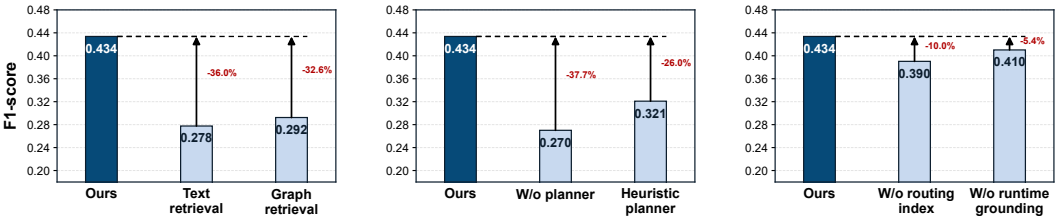
Figure 4 reveals a clear separation between prefilling cost and decoding cost. The planning step stays lightweight across all four settings and consistently runs at higher throughput than the final QA pass. The use of ATPlanner has not imposed an excessive burden on the system. Instead, the dominant source of first-token delay is retrieval-conditioned QA prefilling, where the model must absorb a much longer and more heterogeneous prompt assembled from retrieved evidence.



(a) Prefill stage across S0–S3.

(b) Decode stage across S0–S3.

Figure 4: Latency and throughput for MoRAG under configurations S0–S3. (a) Prefill latency and throughput. (b) Decode latency and throughput.



(a) Impact of retrieval branches.

(b) Impact of planner design.

(c) Impact of MRGraph components.

Figure 5: Ablation results.

Further analysis of configuration combinations involving various smartphones and LLM backbones, please refer to Appendix E.

4.4 Ablation Study

To isolate the contirbution of each MoRAG component, we investigate the following research questions on MultiHopQA. We ablate the retrieval strategy, ATPlanner, MRGraph components, and EvoNER on MultiHopQA with Ministral3-3B (Figure 5).

As shown in Figure 5a, replacing hybrid retrieval with either branch reduces F1 from 0.4337 to 0.2776 (text only) or 0.2924 (graph only), showing their complementarity. Text retrieval lacks entity-transition structure, whereas graph retrieval lacks sufficient semantic grounding and can add noisy neighbors; together they provide bridge evidence and grounded context. Figure 5b compares no planner, a heuristic planner, and ATPlanner. Removing ATPlanner reduces the F1-score to 0.2701 (37.7% degradation); the heuristic reaches 0.3209, still 26.0% below the full model. Thus, unguided retrieval struggles with multi-hop structure and rigid rules cannot capture complex semantic bridges. ATPlanner instead uses one LLM call to guide hybrid retrieval under a fixed budget, supplying the high-level guidance required by MRGraph. Figure 5c shows that removing the routing index lowers F1 from 0.4337 to 0.3903, showing that coarse semantic localization improves retrieval before EP-graph search. Removing runtime grounding lowers F1 to 0.4102 (5.4%); its raw-text neighborhoods recover context lost through paragraph segmentation or extraction, indicating that boundary-induced context loss remains a failure case. Appendix D evaluates continual entity learning. EvoNER starts nearly category-agnostic and acquires domain-specific categories as buffered unknown entities are clustered and sparsely labeled. Across three checkpoints, gains in entity-surface overlap generally accompany downstream F1 gains, supporting the claim that MoRAG can learn unknown information without retraining the mobile LLM.

5 Conclusion

We presented MoRAG, to the best of our knowledge, the first graph-based RAG framework end-to-end evaluated on commercial smartphones. With our proposed MRGraph, and EvoNER, ATPlanner based on it, MoRAG improves accuracy while cutting graph-construction latency and energy use.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code implementation, generate supervision for retrieval-planner training, and assess the correctness of answers produced by the evaluated RAG systems. Specifically, Claude Opus 4.6 generated planner-training supervision, which underwent rule-based filtering and manual verification, and Claude Haiku 4.5 served as an LLM judge under the protocol described in Appendix C. We did not use generative AI tools to conduct the preliminary literature review, formulate the proposed methodology, or design the experiments. These activities, together with the execution of experimental tests, were carried out by the human authors. Additionally, we used generative AI tools to edit the manuscript for language and readability. The authors reviewed the AI-assisted manuscript edits and code, tested the implementation, and manually checked a sample of the LLM-generated judgments. We take responsibility for the final content of this work, including the text, code, and evaluation results produced with AI assistance.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, et al. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of the International Conference on Learning Representations*, 2024.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme context compression for retrieval-augmented generation with one token. In *Advances in Neural Information Processing Systems*, volume 37, pp. 109487–109516, 2024. doi: 10.52202/079017-3476.

Darren Edge, Ha Trinh, Newman Cheng, et al. From local to global: A graph RAG approach to query-focused summarization, 2024.

Tianyu Fan, Jingyuan Wang, Xubin Ren, et al. Minirag: Towards extremely simple retrieval-augmented generation, 2025.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, et al. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *Proceedings of the International Conference on Learning Representations*, 2023.

Jinlan Fu, Xuanjing Huang, and Pengfei Liu. SpanNER: Named entity re-/recognition as span prediction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, pp. 7183–7195, 2021.

Yunfan Gao, Yun Xiong, Xinyu Gao, et al. Retrieval-augmented generation for large language models: A survey, 2023.

Georgi Gerganov. llama.cpp. `https://github.com/ggerganov/llama.cpp`, 2023.

Zirui Guo, Lianghao Xia, Yanhua Yu, et al. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=hkujvAPVsg`.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3929–3938. PMLR, 2020. URL `https://proceedings.mlr.press/v119/guu20a.html`.

Haoyu Han, Yu Wang, Harry Shomer, et al. Retrieval-augmented generation with graphs (GraphRAG), 2025.

Xu Han, Tianyu Gao, Yuan Yao, et al. OpenNRE: An open and extensible toolkit for neural relation extraction. In *Proceedings of EMNLP-IJCNLP: System Demonstrations*, pp. 169–174, 2019. doi: 10.18653/v1/D19-3029.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015. URL https://arxiv.org/abs/1503.02531.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625. International Committee on Computational Linguistics, 2020. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

Gautier Izacard, Patrick Lewis, Maria Lomeli, et al. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

Mandar Joshi, Eunsol Choi, Daniel S Weld, et al. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1601–1611, 2017.

Dong Ho Kang, Hyunjoon Lee, Hyeonjeong Cha, et al. Pocket RAG: On-device RAG for first aid guidance in offline mobile environment, 2026.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, et al. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 6769–6781, 2020.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, et al. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7: 452–466, 2019.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Shilong Li, Yancheng He, Hangyu Guo, et al. GraphReader: Building graph-based agent to enhance long-context abilities of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, et al. Ministral 3, 2026a.

Hao Liu, Zhengren Wang, Xi Chen, et al. HopRAG: Multi-hop reasoning for logic-aware retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1897–1913, 2025. doi: 10.18653/v1/2025.findings-acl.97.

Shuyi Liu, Yuming Shang, and Xi Zhang. TruthfulRAG: Resolving factual-level conflicts in retrieval-augmented generation with knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026b.

Natawut Monaikul, Giuseppe Castellucci, Simone Filice, et al. Continual learning for named entity recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 13570–13577, 2021.

Taehwan Park, Geonho Lee, and Min-Soo Kim. MobileRAG: A fast, memory-efficient, and energy-efficient method for on-device RAG, 2025.

Boci Peng, Yun Zhu, Yongchao Liu, et al. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*, 2025.

Ofir Press, Muru Zhang, Sewon Min, et al. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Qwen Team. Qwen3 technical report, 2025.

Korakit Seemakhupt, Sihang Liu, and Samira Khan. EdgeRAG: Online-indexed RAG for edge devices, 2024.

Haiyang Shen, Hang Yan, Zhongshi Xing, Mugeng Liu, Yue Li, Zhiyang Chen, Yuxiang Wang, Jiuzheng Wang, and Yun Ma. DRAGON: Domain-specific robust automatic data generation for RAG optimization. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 1065–1078. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-eacl.56. URL https://aclanthology.org/2026.findings-eacl.56/.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, et al. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *Proceedings of the International Conference on Learning Representations*, 2024.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, et al. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 10014–10037, 2023.

Zijie J. Wang and Duen Horng Chau. MeMemo: On-device retrieval augmentation for private and personalized text generation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2765–2770, 2024. doi: 10.1145/3626772.3657662. URL https://doi.org/10.1145/3626772.3657662.

Zhepei Wei, Wei-Lin Chen, and Yu Meng. InstructRAG: Instructing retrieval-augmented generation via self-synthesized rationales. In *Proceedings of the International Conference on Learning Representations*, 2025.

Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device language models: A comprehensive review. arXiv preprint arXiv:2409.00088, 2024. URL https://arxiv.org/abs/2409.00088.

Zhilin Yang, Peng Qi, Saizheng Zhang, et al. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018.

Juntao Yu, Bernd Bohnet, and Massimo Poesio. Named entity recognition as dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6470–6476, 2020.

Qianchi Zhang, Hainan Zhang, Liang Pang, et al. Stable-RAG: Mitigating retrieval-permutation-induced hallucinations in retrieval-augmented generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2026.

Zhen Zhang, Yuhua Zhao, Hang Gao, and Mengting Hu. LinkNER: Linking local named entity recognition models to large language models using uncertainty. In *Proceedings of the ACM on Web Conference 2024*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020.

A Baseline implementation details and deployment notes

We evaluate baselines under a compact-backbone regime rather than attempting to re-engineer every released system into a mobile-native application. MoRAG and Naïve RAG are executed on real smartphones for end-to-end mobile experiments. For MiniRAG, the memory-construction experiment is also executed on a real smartphone, because this directly tests whether a lightweight graph-based RAG pipeline can be populated under mobile latency and energy constraints. MiniRAG's QA accuracy experiment, however, is run in the cloud: constructing memory over the full benchmark corpora on a phone would be prohibitively slow and would obscure the intended accuracy comparison. The other RAG baselines are evaluated as cloud-based compact-backbone adaptations with the same benchmark data and comparable retrieval/generation budgets.

Table 3: Execution settings for baseline comparisons.

Method	QA accuracy	Mobile profiling
MoRAG	Real smartphone	Real smartphone
Naïve RAG	Real smartphone	Real smartphone
MiniRAG (Fan et al., 2025)	Cloud adaptation	Real smartphone
GraphRAG (Edge et al., 2024)	Cloud adaptation	Not phone-profiled
InstructRAG (Wei et al., 2025)	Cloud adaptation	Not phone-profiled
StableRAG (Zhang et al., 2026)	Cloud adaptation	Not phone-profiled
TruthfulRAG (Liu et al., 2026b)	Cloud adaptation	Not phone-profiled
xRAG (Cheng et al., 2024)	Cloud adaptation	Not phone-profiled

Baseline scope. Naïve RAG is the direct dense-retrieval baseline. GraphRAG and MiniRAG represent graph-based RAG pipelines; MiniRAG is the closest graph baseline to mobile deployment, so we additionally profile its memory construction on a phone. InstructRAG adds instruction-guided denoising, StableRAG studies retrieval-order stability, TruthfulRAG adds KG-based conflict handling, and xRAG compresses retrieved context. These systems target different operating points, so we interpret the comparison narrowly: it measures how each strategy behaves with compact backbones and matched benchmark inputs, not the maximum performance of the original full-scale systems.

B Experimental Details

Backbone models and quantization. Unless otherwise specified, all on-device models are quantized to Q6_K format using `llama.cpp` (Gerganov, 2023). We consider Qwen3-1.7B and Ministral3-3B as default compact backbones, with Qwen3-4B as a larger variant that still fits on device. For capacity reference, we additionally report results using Qwen3-32B and Ministral3-14B on server-grade hardware.

Offline preparation. All one-time offline steps are performed on a cloud server with two NVIDIA RTX A6000 GPUs, including model quantization, LoRA training, and dataset preprocessing. These steps are executed only once prior to deployment and are not part of the online inference cost.

ATPlanner training. For multi-hop retrieval planning, we trained ATPlanner using a strictly separated data protocol. Training data is constructed exclusively from the **training splits** of MultiHopQA and HotpotQA: for each question, a strong proprietary LLM (Claude Opus 4.6) generates decomposition targets consisting of the seed entity, hop relations, target entity types, and a retrieval budget, followed by rule-based filtering and manual verification. All evaluation is conducted on the **test splits** of the corresponding benchmarks, with zero overlap between training and test samples. Since MultiHopQA lacks an officially constructed set of standard answers, we used the work of previous researchers(Gutiérrez et al., 2024) as the ground truth. This ensures that ATPlanner has never seen any test question or its decomposition during training. The planner is implemented as a LoRA-adapted model using the same compact backbone as the answering model. It is used exclusively to generate structured retrieval plans; final answers are always produced by the quantized backbone conditioned on retrieved evidence.

Mobile deployment. All online stages of MoRAG, including memory ingestion, memory update, retrieval, and answer generation, are executed entirely on device. Experiments are conducted on two commercial smartphones: OnePlus 13 (Snapdragon 8 Elite, 16 GB RAM) and OnePlus 15 (Snapdragon 8 Elite Gen 5, 16 GB RAM). We report latency, memory, and energy measurements collected from real-device execution. The Android implementation records wall-clock stage latency, process PSS memory, battery current/voltage-derived power, and thermal status during NER ingestion and QA runs.

Evaluation protocol: We report two metrics. **Correctness (Cr)** is measured using an LLM-as-Judge protocol (Zheng et al., 2023) under a fixed evaluation rubric. **F1-score** is computed based on token-level overlap after normalization and matching. The full judge prompt, normalization rules, and implementation details are provided in Section C.

Prompt for LLM-as-Judge

You are a strict and impartial QA evaluation expert.
Your task is to assess whether the predicted answer *strictly* and *correctly* answers the question, rather than merely checking if it mentions a *portion* of the gold answer.
Scoring Criteria:
- **Correctness:** 0–100 points. 100 = exactly correct, containing no additional erroneous answers or factual errors.
Strict Grading Requirements:
- A score of 95–100 should be assigned *only* if the predicted answer is semantically equivalent to the gold answer and introduces no additional candidate answers, incorrect entities, incorrect relationships, incorrect dates/locations, or obviously irrelevant elaborations.
- If the predicted answer *includes* the gold answer but simultaneously introduces additional names, locations, years, categories, list items, or descriptions that are either too broad or too narrow in scope, a significant deduction must be applied.
- If the question expects a single entity or phrase, but the prediction provides multiple possible answers, an enumerated list, or an entire explanatory paragraph, full marks should not be awarded.
- If the predicted answer is only partially correct, too broad in scope, too narrow in scope, or contains supplementary information that is unsupported by the context, a score of 40–90 should be assigned, depending on the severity of the error.
- If the predicted answer is empty or clearly inconsistent with the question/gold answer, a score of 0–20 should be assigned.
- You may refer to the provided *context* to determine whether any additional information in the prediction is supported; however, the *gold_answer* remains the primary anchor for assessing correctness.
Notes:
- Aliases, abbreviations/full names, and Chinese/English equivalents may be considered correct provided they do not introduce any additional erroneous information.
- The *reasoning* should be provided as a single line of text, not exceeding 20 characters.

Figure 6: LLM-as-Judge prompt.

C LLM-AS-JUDGE PROTOCOL

Given a question, a reference answer or set of acceptable answers, and the model's prediction, a judge LLM (Claude Haiku 4.5) determines whether the prediction is factually correct under a strict rubric. The judge outputs a correctness score ranged from 0 to 100, label for each sample; we report the mean across samples as **Correctness (Cr, %)**. F1-score is computed as standard token-level precision–recall against reference answers. Retrieval-augmented methods use the same retrieval cutoff when applicable, and compact-backbone runs use the same Q6_K quantization setting. We present LLM-as-Judge prompts used in this paper in Figure 6.

Human-guided judge calibration: For each dataset, we used 25 questions from outside the test set to calibrate the prompt for the LLM-as-judge and reviewed outputs from MoRAG and GraphRAG with Qwen3-1.7B and Ministral3-3B, using the same questions across configurations. Three Ph.D. students examined each answer together with reference answers, source evidence, and the LLM judge's score and rationale, with method names hidden. Disagreements were discussed and scores revised until all reviewers approved the assessment and each score was within two points of their mean on the 0–100 scale. We used this mean to calibrate the judge prompt, revising the prompt and regenerating scores until the reviewed LLM scores differed from the human means by less than five points. The final prompt was then applied to re-score all outputs across all methods, backbones, and benchmarks; the reported Cr uses these LLM scores.

D EXPERIMENT: FROM ZERO TO INFINITY

We hope to enhance EvoNER's recognition ability for new entity categories through incremental training. This ability helps EvoNER continuously provide specific entity categories, enhancing the pruning efficiency and accuracy of ATPlanner searches on graphs. To examine whether our framework has this capability, we design a continual-learning experiment that evaluates whether

the EvoNER can gradually learn previously unseen entity categories and translate this learning into downstream question-answering improvements.

Table 4: Continual entity learning performance across four domains.

Domain	Metric	Training Progress		
		0%	**50%**	**100%**
AI	NER_{surf}	0.322	0.351	**0.401**
	F1-score	0.535	0.547	**0.557**
Book	NER_{surf}	**0.463**	0.449	0.449
	F1-score	0.740	**0.748**	**0.748**
Politics	NER_{surf}	0.578	0.609	**0.661**
	F1-score	0.546	0.554	**0.574**
Music	NER_{surf}	0.419	0.437	**0.448**
	F1-score	0.653	0.654	**0.662**

We initialize EvoNER in a nearly category-agnostic state: it has no predefined domain-specific entity types and can only make surface-level judgments about whether a span is likely to be an entity. During simulated ingestion, unknown entities are buffered and clustered; when a cluster becomes sufficiently coherent, the LLM assigns a type label and then EvoNER starts its continuous learning process. To verify the effectiveness of incremental learning, we evaluate the framework at three checkpoints corresponding to 0%, 50%, and 100% of the training corpus. At each checkpoint, training is paused, the current model state is frozen, and evaluation is conducted on 100 QA questions selected from MultiHopQA.

Table 4 reports the performance across four representative domains. NER_{surf} is the overlap rate between entities identified by NER at the current stage and gold entities; F1-score measures token-level overlap between golden answer and generated answer during QA. Overall, MoRAG demonstrates a consistent ability to absorb new knowledge over time in this streaming setup. As training progresses, the AI, Politics, and Music domains exhibit a steady upward trend in both entity recognition and downstream question answering. For instance, in the Politics domain, EvoNER improves its surface recognition rate from 0.578 to 0.661, helping ATPlanner construct better retrieval trajectories and lifting the F1-score from 0.546 to 0.574. While most domains show clear continuous learning gains, the **Book** domain exhibits a different pattern: its NER_{surf} slightly degrades after the initial state and plateaus, while its F1-score saturates early with minimal improvement. We attribute this behavior to two interacting factors. The first is a *data scale bottleneck*: the Book corpus is the smallest among the four. Because MoRAG relies on accumulating sufficient semantic evidence to trigger robust clustering and reliable LLM labeling, a small data stream provides too few repeated mentions of new entities to activate cluster-level type expansion reliably. The second factor is a high initial baseline: the zero-shot F1-score for Book is already exceptionally high. This suggests that book-related entities are already well-covered by the base NER capabilities and the backbone LLM's parametric memory. When the initial RAG pipeline is already strong, the marginal benefit of discovering new domain-specific concepts diminishes.

E Sensitivity Analysis

We evaluate MoRAG on two commodity Android smartphones, OnePlus 13 and OnePlus 15, the specification are list in Table 5, to measure whether structured RAG can run under realistic mobile hardware constraints. Both devices are equipped with recent flagship Snapdragon mobile SoCs and 16 GB-class LPDDR5X-family RAM. These devices provide a practical upper-bound setting for current consumer smartphones while still preserving the thermal, memory, and energy constraints of handheld deployment. The OnePlus 13 is powered by Snapdragon 8 Elite, while the OnePlus 15 uses the newer Snapdragon 8 Elite Gen 5 platform. This pairing allows us to evaluate MoRAG across two generations of mobile compute capability under the same RAM capacity and operating-system family. Unless otherwise specified, all latency, memory, and power measurements reported in the paper are collected from real-device execution on these phones.

Table 5: Evaluation device specifications.

	OnePlus 13	**OnePlus 15**
SoC	Snapdragon 8 Elite	Snapdragon 8 Elite Gen 5
RAM	16 GB LPDDR5X	16 GB LPDDR5X Ultra+
Storage	256 GB UFS 4.0	256 GB UFS 4.1
OS	ColorOS 16	ColorOS 16

Table 6: Configuration.

	Qwen3-1.7B	**Qwen3-4B**
OnePlus 15	S0	S2
OnePlus 13	S1	S3

The online profiling study then measures the QA path of MoRAG across four smartphones–LLM configurations, as defined in Table 6.

Table 7: Average PSS RAM and power consumption.

Config	**Graph Construction**		**Runtime Answering**	
	PSS RAM(MB)	**Power(W)**	**PSS RAM(MB)**	**Power(W)**
S0	5527.31	2.40	4149.72	5.26
S1	5099.07	2.93	4153.38	3.75
S2	6717.55	2.74	6803.68	5.38
S3	6643.01	2.37	6512.62	3.95

We first examine MoRAG's own resource usage during graph construction and runtime answering. As summarized in Table 7, the 4B backbone consistently incurs a substantially heavier memory footprint than 1.7B on both smartphones. This gap appears in both graph construction and end-to-end QA, where the 1.7B settings remain in the 4.1–5.5 GB range, while the 4B settings rise to roughly 6.5–6.8 GB.

The full 4B question-answering path crosses a practical usability threshold on mobile device: once the full RAG pipeline is included, answer latency increases enough to shift the interaction mode from responsive assistance to prolonged waiting. The cross-device gap is real but secondary. Across both phones, backbone size dominates the latency profile, indicating that the main deployment constraint is the cost of long-context QA itself rather than retrieval planning or device generation. These measurements justify our choice to use the 1.7B backbone in the main accuracy experiments.

F Related Work

On-device LLMs. Recent advances in quantization (Frantar et al., 2023) and efficient inference engines (Gerganov, 2023) have made it feasible to run billion-parameter models on mobile hardware (Xu et al., 2024). Models such as Qwen3 (Qwen Team, 2025) and Ministral3 (Liu et al., 2026a) provide compact backbones that can be deployed on edge devices. These efforts mainly improve model execution; MoRAG instead studies how a mobile assistant can maintain structured memory and update it continuously on the device.

Retrieval-augmented generation. RAG (Lewis et al., 2020) augments LLMs with external knowledge at inference time, commonly using dense passage retrieval (Karpukhin et al., 2020). Recent methods improve this pipeline through self-reflection (Asai et al., 2024), context compression (Cheng et al., 2024), or multi-step retrieval and reasoning (Trivedi et al., 2023; Press et al., 2023). Mobile RAG systems usually favor vector retrieval because it is simple and efficient (Park et al., 2025; Kang et al., 2026; Seemakhupt et al., 2024). MoRAG targets the complementary problem of making graph-structured retrieval feasible under the same mobile constraints.

Named entity recognition and continual learning. Span-based NER (Yu et al., 2020; Fu et al., 2021) formulates entity detection as span classification over enumerated candidates. Continual NER (Monaikul et al., 2021) studies how to absorb new entity types while preserving old ones, commonly using distillation (Hinton et al., 2015) or regularization (Kirkpatrick et al., 2017). EvoNER adapts this idea to mobile graph writing: uncertain but entity-like spans are buffered,

clustered by contextual embeddings, sparsely labeled by the LLM at the cluster level, and then incorporated through confidence-gated distillation.

Graph-based RAG systems. Graph-based RAG systems organize text into graph structures for multi-hop evidence composition (Edge et al., 2024; Han et al., 2025). Representative systems use community summaries, lightweight graph construction, or iterative graph navigation (Guo et al., 2025; Sun et al., 2024; Li et al., 2024; Fan et al., 2025). These designs are effective in server-side settings, but their deep LLM-generated semantic levels are difficult to construct, update, and traverse on smartphones. MoRAG therefore uses a shallow graph memory that keeps the abstraction needed for search and the provenance needed for grounded reasoning.