Cost-Efficient RAG for Entity Matching with LLMs: A Blocking-based Exploration

[Experiment, Analysis & Benchmark]

Chuangtao Ma*
Aalborg University
Aalborg, Denmark
chuma@cs.aau.dk

Zeyu Zhang*
University of Amsterdam
Amsterdam, the Netherlands
z.zhang2@uva.nl

Arijit Khan
Bowling Green State University, USA
Aalborg University, Denmark
arijitk@bgsu.edu

Sebastian Schelter
BIFOLD & TU Berlin
Berlin, Germany
schelter@tu-berlin.de

Paul Groth
University of Amsterdam
Amsterdam, the Netherlands
p.t.groth@uva.nl

Abstract

Retrieval-augmented generation (RAG) enhances LLM reasoning in knowledge-intensive tasks, but existing RAG pipelines incur substantial retrieval and generation overhead when applied to large-scale entity matching. To address this limitation, we introduce *CE-RAG4EM*, a cost-efficient RAG architecture that reduces computation through blocking-based batch retrieval and generation. We also present a unified framework for analyzing and evaluating RAG systems for entity matching, focusing on blocking-aware optimizations and retrieval granularity. Extensive experiments suggest that *CE-RAG4EM* can achieve comparable or improved matching quality while substantially reducing end-to-end runtime relative to strong baselines. Our analysis further reveals that key configuration parameters introduce an inherent trade-off between performance and overhead, offering practical guidance for designing efficient and scalable RAG systems for entity matching and data integration.

PVLDB Reference Format:
Chuangtao Ma, Zeyu Zhang, Arijit Khan, Sebastian Schelter, and Paul Groth. Cost-Efficient RAG for Entity Matching with LLMs: A Blocking-based Exploration. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/machuangtao/CE-RAG4EM.

1 Introduction

Entity matching (EM) is a fundamental data integration task that determines whether two records refer to the same real-world entity. Prior work spans rule-based systems [39, 41, 53], correlation-based techniques [7, 16], machine learning and deep learning models [4, 34, 58], and active learning approaches [20, 32]. EM remains challenging along both efficiency and effectiveness dimensions: comparing all m records in one table with n records in another incurs a quadratic $O(mn)$ cost, making scalability a central concern. Blocking is therefore essential, as it groups records into candidate sets and restricts comparisons to plausible pairs, substantially reducing the search space and enabling large-scale EM [42]. On the effectiveness side, noisy, heterogeneous, and context-dependent attributes necessitate advanced similarity modeling, feature learning, and multi-step refinement.

Recent work explores transformer-based pre-trained language models (PLMs), which reduce but do not completely eliminate the need for labeled data since many methods still rely on costly fine-tuning or knowledge distillation [10, 26, 27, 60]. Moreover, PLMs continue to struggle in complex scenarios requiring deeper contextual reasoning or structured knowledge. Large language models (LLMs) such as GPT-4 demonstrate strong generalization in low-resource and cross-domain data integration, driving recent advances in schema matching, entity resolution, and entity matching [13]. Their ability to interpret context and generate structured outputs has made LLM-based EM an emerging direction for large-scale integration tasks [46, 64, 70], supported by techniques such as prompt engineering [1, 21, 36, 64], fine-tuning [35, 45, 50, 54, 67], and in-context learning [14, 45, 49]. However, LLM-based methods face substantial challenges in real-world, large-scale settings: massive tables with heterogeneous and limited attribute information reduce accuracy; highly imbalanced match distributions, sparse supervision, and model biases increase false negatives; limited reasoning depth and hallucination tendencies further undermine reliability. As a result, LLM-based EM often experiences significant performance degradation and high computational cost in enterprise-scale data integration pipelines [5].

Retrieval-augmented generation (RAG) [23, 25] enhances the trustworthiness and explainability of LLM outputs by integrating retrieved factual knowledge with instruction-based prompting, enabling more reliable reasoning for knowledge-intensive tasks. RAG has proven effective across data management applications, e.g., natural language querying [59], tabular QA [51, 56, 72], and schema matching [28, 52]. Recent work shows that external knowledge substantially reduces hallucinations in real-world heterogeneous integration scenarios [29]. However, *RAG-based EM remains largely*

*Equal contribution.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

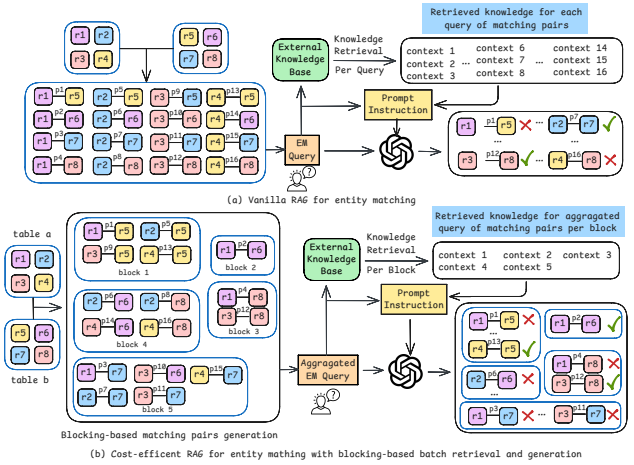


Figure 1: Vanilla RAG vs. CE-RAG4EM for entity matching: (a) per-query retrieval/generation; (b) blocking-based batch retrieval/generation. Matched records share the same color.

unexplored. Moreover, even when hallucinations are mitigated, existing RAG systems incur significant computational overhead due to per-query context retrieval and the high cost of vector embedding and nearest-neighbor search over large knowledge bases [18, 22]. These inefficiencies are amplified in EM, where many queries are repetitive or near-duplicated, causing retrieval modules to repeatedly fetch overlapping context and resulting in substantial, unnecessary latency and cost.

Overall, the research gaps underlying these challenges can be summarized as follows. **(1)** *Need of cost-efficient retrieval for RAG-based EM.* Vanilla RAG systems retrieve context independently for each query (Figure 1 (a)), which becomes prohibitively expensive over large knowledge bases. In EM, many queries are highly similar, yet mechanisms such as integrating blocking with RAG for cost-efficient batch retrieval (Figure 1 (b)) remain unexplored. **(2)** *Lack of a unified RAG framework for EM.* There is no unified framework for EM that supports principled comparision of RAG variants (e.g., RAG, GraphRAG [19], KG-RAG [71]) across heterogeneous knowledge sources. In particular, existing approaches differ significantly in retrieval pipelines, graph-traversal strategies, and blocking methods required for effienct batch retrieval, making fair and systematic comparison difficult. **(3)** *Absence of a systematic evaluation of RAG on EM.* The effectiveness of RAG-based EM depends jointly on retrieval quality and LLM capabilities, yet key design factors and trade-offs between accuracy and computational cost remain insufficinetly characterized. Existing evaluations [6, 18] primarily focus on general QA tasks, leaving RAG for EM without a dedicated assessment.

Contributions. The contributions of this work can be summarized as follows:

- We introduce *CE-RAG4EM*, a blocking-guided, cost-efficient RAG system for large-scale entity matching (§ 3).
- We present a unified framework for analyzing and evaluating RAG for entity matching, covering blocking-based retrieval /generation, different granularity context, and both vector search and graph traversal in RAG and KG-RAG settings (§3).
- We systematically evaluate *CE-RAG4EM* to validate its design choices and characterize the performance–overhead trade-offs, by studying batch retrieval/generation, retrieval granularity, and graph traversal, and benchmarking against strong LLM and PLM-based baselines. We also analyze alternative blocking methods and backbone LLMs, and perform sensitivity analysis of key parameters (max block size, Top-k context) (§4).
- We summarize empirical insights on key design choices in *CE-RAG4EM* and discuss their implications for efficient and scalable RAG-based entity matching (§5).

Despite blocking having been studied for reducing comparison costs in entity matching [58], our work is the first to introduce blocking to RAG and KG-RAG for LLM-based EM with batch retrieval and inference.

2 Preliminaries

We present the preliminaries behind LLM-based Entity Matching (EM) and provide background on RAG and KG-RAG for EM.

2.1 LLM-based Entity Matching

Definition 1 (Entity Matching Problem). *Let T_s and T_t be the source and target tables, and let $\mathcal{R}$ denote the record space. Given two records $r_1 \in T_s$ and $r_2 \in T_t$, the entity matching (EM) problem is to decide whether r_1 and r_2 refer to the same real-world entity.*

Accordingly, an EM system learns a function

$$f : T_s \times T_t \rightarrow \{0, 1\},$$

where $f(r_1, r_2) = 1$ indicates a match and $f(r_1, r_2) = 0$ otherwise.

Definition 2 (LLM-based Entity Matching). *LLM-based EM replaces task-specific classifier function f with a generative model that directly reasons over serialized record pairs.*

$$f(r_1, r_2) = \mathcal{F}_{LLM}(r_1, r_2) \rightarrow \{yes, no\}.$$

Instead of learning a task-specific decision boundary, LLMs are prompted with textual descriptions of record pairs and asked to produce a binary decision (e.g., yes or no).

2.2 RAG and KG-RAG for EM

Beyond the structured records, we assume a factual knowledge graph (e.g., Wikidata [61]) providing entity-centric facts and relationships, e.g., product taxonomies, canonical identifiers.

Definition 3 (Knowledge Graph). *A knowledge graph (KG) is denoted as $\mathcal{G} = (V, P, E)$, where V is a finite set of nodes (entities or concepts), P is a finite set of predicate (relation) types, and $E \subseteq V \times P \times V$ denotes a finite set of directed, typed edges.*

We write $\mathcal{T} = V \times P \times V$ for the universe of possible triples and view $E \subseteq \mathcal{T}$ as the set of triples present in the KG.

RAG-based EM augments LLM by incorporating the serialized pairs with the retrieved knowledge, i.e., unstructured knowledge (entity or predicate) in RAG and structured knowledge (triples) in KG-RAG.

Given a record pair (r_1, r_2), we aim to extract contextual knowledge from KG $\mathcal{G}$ that is *relevant* to this pair and *complements or enriches* the information contained in the records.

Definition 4 (Knowledge Retriever and Triple Search). *A knowledge retriever and triple search is a function, which maps a pair of records (r_1, r_2) to a finite set $\mathsf{Retr}(r_1, r_2) \subseteq E$, consisting of the set of graph nodes that correspond to record $r \in \mathcal{R}$, and expanded triples from the knowledge graph $\mathcal{G}$, which are relevant to (r_1, r_2).*

Intuitively, $\mathsf{Retr}(r_1, r_2)$ may return contextual knowledge at different granularities: (1) entity-level context (entities relevant to r_1 or r_2), (2) predicate-level context (relations/predicates connecting relevant entities), and (3) triple-level context obtained by expanding from relevant entities or predicates.

We now describe how a record pair and the retrieved knowledge are represented as input to the entity matching model. Let Σ be a finite alphabet (e.g., characters or tokens), and let Σ^* denote the set of all finite strings over Σ.

Definition 5 (Graph-Aware Serializer). *A graph-aware serializer is a function that maps $(r_1, r_2, \mathsf{Retr}(r_1, r_2))$ to a textual sequence*

$$x(r_1, r_2) \;=\; \mathsf{Serializer}\big(r_1, r_2, \mathsf{Retr}(r_1, r_2)\big) \in \Sigma^*.$$

The serializer is responsible for constructing a prompt or input string that exposes (1) the original attributes of r_1 and r_2, and (2) the extracted knowledge $\mathsf{Retr}(r_1, r_2)$, to the matching model in a structured way (for example, by listing the record attributes followed by a formatted list of relevant triples).

Definition 6 (RAG and KG-based Entity Matching). *RAG- and KG-RAG-based EM augment LLM-based EM by incorporating retrieved contextual knowledge into the LLM input. In RAG4EM, the context is textual entity/predicate descriptions; in KG-RAG4EM, it is KG triples:*

$$\mathcal{F}_{LLM}(x(r_1, r_2)) \to \{yes, no\}.$$

RAG4EM allows the LLM to leverage auxiliary background knowledge beyond the input records, while KG-RAG4EM provides structured relational evidence via retrieved triples, which can improve decisions on ambiguous record pairs.

2.3 RAG4EM with Batch Input and Inference

We formalize a batch variant of RAG4EM, where the generative model processes multiple record pairs in a single request.

Definition 7 (RAG with Batch Input and Inference). *Given a batch of record pairs*

$$\big((r_1{}^{(1)}, r_2{}^{(1)}), \ldots, (r_1{}^{(B)}, r_2{}^{(B)})\big) \in (\mathcal{R} \times \mathcal{R})^B,$$

we construct a single *serialized input sequence*

$$bx = \mathsf{Serializer}\big(\{(r_1^{(i)}, r_2^{(i)})\}_{i=1}^{B},\ \{\mathsf{Retr}(r_1^{(i)}, r_2^{(i)})\}_{i=1}^{B}\big).$$

A generative matching model M takes x as input and produces an output text, which is parsed into a batch of match decisions

$$\mathcal{F}_{LLM}(bx) \in \{yes, no\}^B.$$

A single model call thus returns B match decisions for the B input pairs.

Remark. The matching model is typically instantiated as an LLM that has been pre-trained and may already internalize substantial knowledge that overlaps with the knowledge encoded in $\mathcal{G}$. Consequently, the extracted KG $\mathsf{Retr}(r_1, r_2)$ can be:

- *beneficial*, when it provides factual, precise, or more up-to-date knowledge than internal knowledge of model; or
- *redundant* (or even detrimental), when it merely repeats what the model already "knows" or introduces noise.

Our formalization allows both: when $\mathsf{Retr}(r_1, r_2) = \emptyset$, or when M effectively ignores the triples in $\mathsf{Retr}(r_1, r_2)$, the system reduces to an LLM-based EM without external knowledge.

3 Methodology

We present *CE-RAG4EM*–a blocking-guided design for cost-efficient RAG pipeline in entity matching.

3.1 Overview

CE-RAG4EM provides a cost-efficient RAG pipeline for EM, summarized in Figure 2. The system begins with blocking on the raw entity records from the source table and the target table to create a blocking-based matching pair by grouping similar records across tables into the same block and deduplicating the redundant matching pairs across blocks (§ 3.2). For each block, the entity matching queries are created for each matching pair, and the queries are then concatenated within the block for an aggregated query that is further vectorized and sent to the vector database of KG for contextual knowledge retrieval (§ 3.3). Top-ranked retrieved items then seed triple search and graph expansion, enabling the system to gather relevant triples and subgraphs from the external KG via breadth-first search and neighborhood expansion within the triple-search module (§3.4). Retrieved entities and predicates–along with those discovered during triple search–are enriched with textual statements and refined using similarity-based ranking and instruction-tuned filtering in the knowledge enrichment and refinement stage (§3.5). Finally, *CE-RAG4EM* employs tailored prompting strategies for both per-query generation and block-level batch generation, enabling the LLM to effectively leverage ranked Top-k knowledge to produce accurate EM decisions (§3.6).

3.2 Blocking-based Matching Pair Generation

To mitigate the quadratic complexity of per-query contextual knowledge retrieval and generation, we propose a blocking-based matching-pair batch construction strategy that enables efficient batch retrieval and querying for RAG-based entity matching. Blocking groups similar records into the same batch, reducing redundant retrieval across highly related queries. Formally, let T_s and T_t denote the source and target tables, each containing multiple records. The blocking-based batch construction process for EM consists of three phases.

Records Preprocessing and Block Generation. To group similar records from T_s and T_t into the same block, we first construct a unified search space by forming the global record set $\mathcal{R} = T_s \cup T_t$. We then apply suitable blocking functions [42] to $\mathcal{R}$, which partition the records into a set of similarity-based blocks $\mathcal{B} = \{B_1, B_2, \ldots, B_\alpha\}$.

Candidate Pair Generation within a Block. For each block B, we construct a set of candidate matching pairs P_B by taking the Cartesian product of the source-table and target-table records contained within that block: $P_B = \{(r_i, r_j) \mid r_i \in (B \cap T_s),\ r_j \in (B \cap T_t)\}$. Each element of P_B is a candidate pair formed from a source record r_i and a target record r_j that co-occur in the same block.

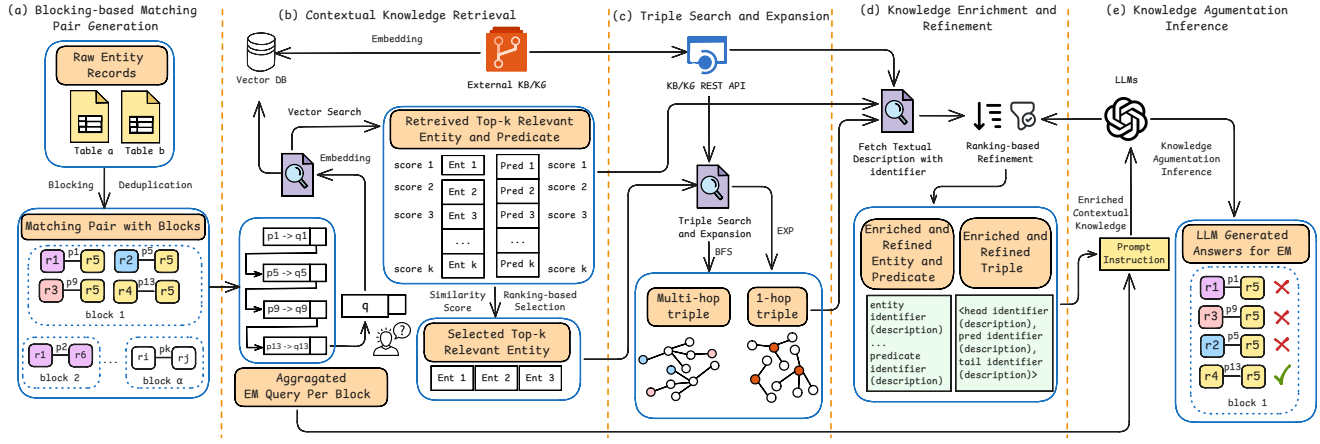


Figure 2: Oveview of Our *CE-RAG4EM*. The framework is composed of five phases (a)-(e), which are detailed in §3. The check mark indicates that LLM responds with `Yes` for the given EM query, while a cross mark indicates that LLM responds with `No`.

Deduplication. Since a record may satisfy multiple blocking keys or similarity thresholds, it can appear in several blocks, which naturally leads to redundant matching pairs. To prevent this overlap, we apply a deduplication rule that retains only the first occurrence of any matching pair in the earliest block where it appears, discarding all subsequent duplicates in later blocks. This ensures that each matching pair is assigned to exactly one block and avoids repeated processing. After this blocking-based batch construction and deduplication, each block contains a coherent group of similar matching pairs that is ready for downstream processing, including batch retrieval and batch querying.

3.3 Contextual Knowledge Retrieval

The contextual knowledge retrieval component relies on vector-based retrieval, using vector similarity to identify and return the Top-k entities and predicates from the external knowledge graph that are most relevant to a given entity matching query.

Vector-based Entities and Predicates Retrieval. To retrieve the Top-k relevant contextual entities V_k and predicates P_k from the external knowledge graph $\mathcal{G}$, we employ dense vector retrieval and similarity-based ranking. Given a specific entity matching query q from a matching pair, the dense retrieval aims to retrieve the relevant entities and predicates from $\mathcal{G}$. A pre-trained encoder model (i.e., Jina Embeddings V3 [55]) is utilized to map the EM query q as well as the entities and predicates in $\mathcal{G}$ into a d-dimensional embedding space. The relevance score between the given query q and each entity or predicate is quantified via cosine similarity between the embeddings in the vector space. Based on the ranking of the vector similarity in descending order, the Top-k relevant entities V_k and predicates P_k are obtained, which provide inputs for the subsequent triple search and expansion.

Blocking-based Batch Retrieval. To reduce the high retrieval latency and computational overhead of per-query retrieval in vanilla RAG, we introduce a blocking-based batch retrieval mechanism that operates over the matching pairs generated within each block in §3.2.

(a) Threshold-based Block Decomposition. Blocking can sometimes produce very large blocks when many similar queries accumulate in the same candidate set, a known consequence of blocking biases and limitations in entity matching [33]. To prevent such oversized blocks, we apply a threshold-based decomposition strategy: whenever a block contains more matching pairs than the allowed maximum *max_bs*, it is partitioned into several smaller, non-overlapping sub-blocks, each capped at *max_bs* pairs. This ensures that large blocks are divided into manageable units, keeping retrieval and generation efficient and well-balanced across all sub-blocks.

(b) Blocking-based Query Aggregartion and Retreival. We aggregate all matching-pair queries within the same block into a single unified query, allowing contextual knowledge to be retrieved once per block rather than individually for each pair. Building on the matching-pair construction (§3.2), we concatenate the queries associated with all pairs in a block—and any sub-blocks derived from it—into one combined retrieval query. This unified query is then used to obtain the Top-k relevant entities and predicates through dense search and ranking. Since the matching pairs grouped into a block share similar attributes and keywords, the aggregated query preserves their semantic coherence, enabling the retrieved contextual knowledge to be more comprehensive and better aligned with the need of each individual pair.

3.4 Triple Search and Expansion

Following the vector-based retrieval phase, we introduce a triple search and expansion procedure to leverage the explicit structural knowledge associated with the retrieved items by exploring and expanding them into triples over $\mathcal{G}$. This process operates on the Top-k retrieved entities V_k and applies two complementary techniques– breadth-first search and neighborhood-based expansion –to gather structurally relevant triples from the external knowledge graph.

Breadth-First Search (BFS). We extract structurally relevant subgraphs by performing a BFS-based triple search over the external knowledge graph $\mathcal{G}$, using the retrieved Top-k entities V_k as starting points. From these entities, we form all possible source-destination pairs and, for each pair, run a BFS traversal to identify triples that connect them through multi-hop paths in the KG. This traversal explores sequences of linked triples that reveal meaningful structural

relationships between the entities. To control computational cost, the search is bounded by a maximum depth $D_{\max}$; once this limit is reached, the traversal stops and no further triples are explored. This depth constraint ensures that the triple search remains efficient while still capturing informative structural connections.

Neighborhood-based Expansion (EXP). The EXP strategy complements BFS by focusing on one-hop structural context rather than multi-hop paths. For each entity in the Top-k set V_k, EXP identifies its directly connected neighboring entities and the associated predicates in the knowledge graph, forming a one-hop neighborhood triple that reflects the entity's immediate structural surroundings. This expansion captures locally relevant contextual knowledge for each query, ensuring that the most directly connected information is incorporated into downstream processing [6].

3.5 Knowledge Enrichment and Refinement

Knowledge enrichment aims to enrich the retrieved entity and predicate identifiers with the corresponding textual description that is inherent in the external KG, while the ranking-based knowledge refinement aims to select and feed the Top-k retrieved contextual knowledge for better knowledge augmentation.

Knowledge Enrichment with Entity and Predicate Identifier. The entities and predicates retrieved through vector-based retrieval are essentially symbolic identifiers, and the subsequent triple search and expansion therefore produce subgraphs that also consist only of identifiers. On their own, these identifiers provide limited contextual value for LLM-based reasoning or fact-checking, as they lack the descriptive information needed for meaningful interpretation. To address this gap, we introduce a knowledge-enrichment step that augments each retrieved entity and predicate with its corresponding textual description from external KG. For every retrieved entity and predicate identified in § 3.3, the enrichment module iteratively fetches the associated textual description via identifier. All retrieved entities and predicates are then represented in the form `identifier (description)`, and triples are expressed as `<head identifier (description), predicate identifier (description), tail identifier (description)>`. This enrichment transforms abstract identifiers into interpretable text, enabling LLMs to leverage richer contextual knowledge for reasoning.

Knowledge Refinement. Although vector similarity retrieval identifies the Top-k relevant knowledge, it may still return lower-scoring results that introduce noise when passed directly to an LLM. This issue is amplified by the BFS and EXP expansion steps, which can surface broad contextual information that is not always relevant to a specific query. To mitigate noise and preserve answer quality, we introduce ranking-based knowledge-refinement modules that filter and retain only the most relevant contextual knowledge for augmentation. The refinement process consists of two components.

(a) Vector-similarity Ranking based Refinement. The Top-k entities and predicates are refined directly according to their vector-similarity scores in descending order. EXP-generated triples are ranked by the dense similarity of the corresponding seed entity to the query and their sequence order from the initial retrieval stage, to select the Top-k most relevant triples.

This ensures that the triple derived from the high-relevance seed entity is prioritized as a relevant contextual knowledge for knowledge augmentation, because the sequence of the generated triple is the same as the sequence of the seed entities for EXP-based triple search.

The Top-k BFS triples are selected according to their order of appearance in the triple list, which mirrors the ranking of the Top-k seed entities used for triple search. This ordering is preserved because the BFS queue is constructed sequentially from source-destination pairs formed according to the ranking of the seed entities, ensuring that the triple sequence remains consistent with the initial retrieval order.

Prompt for Per-Query Inference:
You are an expert in entity matching, who is to determine whether these two given entity representations refer to the same entity. You are also provided with additional information retrieved from Wikidata, which might be helpful for your reasoning.
Input: Entity 1: { } Entity 2: { } Additional Information (You can use this in your reasoning if available): { }
Instruction: 1. Analyse each entity's semantics independently: consider key terms, roles, and context. 2. Rank the relevance of each entry in the additional information, and only use it if it helps make the decision. 3. Perform a step-by-step logical comparison of the two entities.
Output Format: Match Decision: [Yes / No]

(b) Instruction tuning based Knowledge Refinement. We apply instruction tuning based knowledge refinement to remove irrelevant or unhelpful contextual information before LLM inference. By leveraging the model's in-context reference capabilities, we embed an instruction in the prompt that directs the LLM to assess the relevance of the provided knowledge and use it only when it contributes meaningfully to the decision. This ensures that only highly relevant contextual information is incorporated into LLM's reasoning.

3.6 Knowledge Augmentation Inference

After retrieving and refining the Top-k relevant knowledge from external sources, we design prompt instructions that guide the LLM to effectively use this knowledge for inference augmentation. The instruction design supports two modes of knowledge augmentation: per-query generation and blocking-based batch generation.

Prompt Instruction for Per-Query Inference. To incorporate the refined contextual knowledge retrieved from the external KG, we design a prompt instruction that guides the LLM's inference on a per-block basis. For each query, the system first retrieves the ranked contextual knowledge associated with its corresponding `block_id` and then integrates it into the prompt. The per-query instruction follows a *filter-then-reasoning* logic, enabling the LLM to discard misleading or noisy knowledge before inference. In addition, the prompt encourages step-by-step comparison, allowing the model to fully leverage the provided context for more accurate reasoning. By combining enriched knowledge with instruction-guided prompting, this approach leverages both knowledge-based reasoning and LLM inference to improve the reliability of entity matching.

Prompt for Blocking-Based Batch Inference:
The preamble is identical to the per-query inference prompt.
Input: Entity Pairs in a Batch: [Pair 1 - Entity 1: { } Entity 2: { } ... Pair N - Entity 1: { } Entity 2: { }] Additional Information (shared; you may use this in your reasoning if available): { }
Instruction: 1. Process each entity pair sequentially, and treat each pair independently. *The remaining instructions are the same as the instructions (1-3) for per-query inference.*
Output format: Match Decisions: [Yes / No]

Prompt Instruction for Blocking-based Batch Inference. To further reduce computation overhead and token usage in *CE-RAG4EM*, we extend the block-level design from the retrieval phase to the inference phase through a block-based batch generation strategy. Rather than invoking the LLM separately for each entity matching query, we combine the aggregated queries within a block with their corresponding retrieved contextual knowledge and a shared instruction prompt, enabling the LLM to generate answers for the entire batch in a single call. The batch of queries is processed sequentially within the same prompt, allowing the model to perform structured, block-level reasoning guided by the retrieved knowledge. This approach enables the model to effectively and efficiently reason over multiple queries in one consolidated inference pass, while avoiding repeated inclusion of the instruction template, thereby substantially reducing input token consumption.

3.7 Representative CE-RAG4EM Solutions

The design space of *CE-RAG4EM* is structured around two core dimensions: blocking-based optimization and retrieval granularity.

- **Blocking-based Optimization (BO).** The blocking-based optimization consists of block-level batch retrieval and block-level batch generation, and either strategy can be applied independently or jointly within the *CE-RAG4EM* framework.
- **Retrieval Granularity (RG).** The retrieval granularity depends on the granularity level of retrieved contextual knowledge, which contains entity & predicate-based textual knowledge and triple-based structured knowledge.

Given these multiple configuration options in *CE-RAG4EM*, we identify a set of representative solutions in Table 1. Though these six variants differ in whether batch retrieval and batch generation are enabled and in the type of contextual knowledge provided to the LLM, they share the same underlying vector-based retrieval pipeline along with knowledge-enrichment and refinement modules.

Table 1: Summary of the Design Solutions of *CE-RAG4EM*.

Design Solution		Retrieval Granularity (RG)	
		Entity & Predicate	**Triple**
BO	**Batch Retrieval (BR)**	CE-RAG4EM-BR	CE-KG-RAG4EM-BR
	Batch Generation (BG)	CE-RAG4EM-BG	CE-KG-RAG4EM-BG
	BR & BG	CE-RAG4EM-BR-BG	CE-KG-RAG4EM-BR-BG

4 Experiments

We present a comprehensive empirical evaluation of *CE-RAG4EM*, a cost-efficient retrieval-augmented generation (RAG) framework for entity matching (EM). Our evaluation has two goals: (i) assess the matching effectiveness of *CE-RAG4EM* against representative PLM- and LLM-based EM approaches, and (ii) systematically analyze how key design choices in RAG-based EM govern accuracy and efficiency. We further use controlled ablations to attribute observed gains and overheads to individual components. **The data, code, and a full version are available on GitHub.**[1]

Table 2: Nine datasets [11, 48], grouped by domain, with summary statistics. Attribute types: T (text), N (numeric), C (categorical), D (date), M (mixed text+numeric); counts in parentheses. #Pos/#Neg denote matched/unmatched pairs.

	Dataset	Domain	#Attr.	#Attr. Type	#Pos.	#Neg.
ABT	Abt-Buy	web product	3	T(1), N(1), M(1)	1,028	8,547
AMGO	Amazon-Google	software	3	M(1), T(1), N(1),	1,167	10,293
BEER	Beer	drink	4	T(3), N(1)	68	382
DBAC	DBLP-ACM	citation	4	T(2), C(1), D(1)	2,220	10,143
DBGO	DBLP-Google	citation	4	T(2), C(1), D(1)	5,347	23,360
FOZA	Fodors-Zagats	restaurant	6	T(3), C(1), N(2)	110	836
ITAM	iTunes-Amazon	music	8	T(4), N(2), C(1) D(1)	132	407
WAAM	Walmart-Amazon	electronics	5	T(1), C(2), N(1), M(1)	962	9,280
WDC	Web Data Commons	web product	5	C(2), M(1), T(1), N (1)	2,250	7,992

Our experiments explore the design space of *CE-RAG4EM* along two orthogonal dimensions. The first dimension is *retrieval granularity*, comparing entity- and predicate-level retrieval with knowledge graph (KG)-based triple retrieval, to understand how evidence granularity affects relevance and downstream generation. The second dimension focuses on *blocking-based optimization*, which amortizes overhead by enabling batch retrieval and batch generation.

We conduct extensive experiments to evaluate both matching quality and efficiency, reporting retrieval overhead and end-to-end latency as primary efficiency metrics. Since latency is closely tied to practical inference cost (e.g., API usage) under a fixed serving setup, these results also serve as a proxy for computational cost. All experiments are repeated with *three random seeds*, and we report averaged results. Together, the experiments clarify when and why *CE-RAG4EM* is effective and quantify the trade-offs introduced by its design choices.

4.1 Experimental Setup

This section describes the datasets, baselines, evaluation metrics, and key implementation details used in our empirical study. Unless stated otherwise, all experiments are run on a Ubuntu 22.04 server with 60 CPU cores (Intel Xeon Ice Lake, 2.8GHz), two NVIDIA A100 GPUs (40GB each), and 100GB RAM.

Datasets. We evaluate *CE-RAG4EM* on nine widely used entity matching benchmarks from the Magellan [11] and Web Data Commons (WDC) [48] collections. These datasets cover diverse domains and vary in schema complexity, attribute types, and class imbalance. Table 2 summarizes the key statistics of all datasets.

Knowledge Base & Knowledge Retrieval. We use Wikidata [9, 61] as the external knowledge base for retrieval and augmentation. Our goal in selecting Wikidata is to provide a *domain-agnostic* and *public* knowledge source that can be applied uniformly across all nine benchmarks, enabling a more universal and reproducible comparison of retrieval strategies. Though domain-specific knowledge graphs (e.g., product or bibliographic KGs) could offer higher coverage and yield stronger results in particular domains, they are not consistently available across datasets and would introduce confounding factors tied to domain engineering. By using Wikidata,

[1] https://github.com/machuangtao/CE-RAG4EM

we can isolate the impact of *CE-RAG4EM*'s design choices under commonsense knowledge, rather than attributing performance differences to the availability or quality of a domain-specific KG. For retrieval, we use the public vector index released by the Wikidata Embedding Project.[2] Retrieval is issued in *natural language*: for each candidate entity pair, we construct a textual query from their attribute descriptions and submit it to the embedding service, which returns the nearest Wikidata entities and/or predicates in the embedding space (depending on the retrieval setting). We further use the Wikidata REST API[3] to resolve the retrieved entity/predicate identifiers to labels and descriptions, and to fetch the connected items and relations to construct triples for CE-KG-RAG4EM.

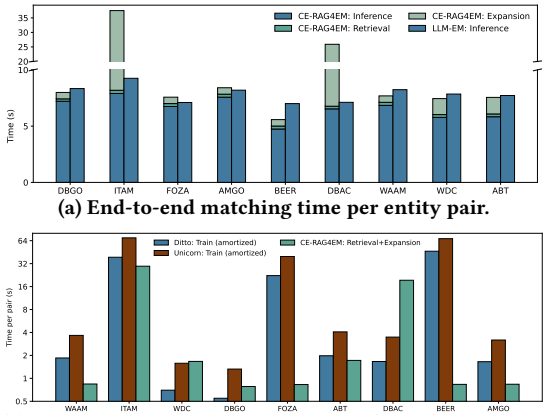
Baselines. We compare *CE-RAG4EM* against three categories of baselines. (i) *PLM-based entity matching*: Ditto [26] and Unicorn [60]. (ii) *LLM-based entity matching (LLM-EM)*: direct prompting of an LLM using only the input record pair, without external retrieval. (iii) *Vanilla RAG4EM*: a standard RAG pipeline for EM that performs retrieval and generation independently for each query (no batching), and does not include KG traversal or triple augmentation. Depending on the retrieval granularity, it retrieves either Wikidata entities or predicates ranked by vector similarity.

Evaluation Metrics. We evaluate *CE-RAG4EM* along three axes: **(i) matching quality**, reported primarily with **F1** and accompanied by **Precision/Recall** to make trade-offs explicit under class imbalance; **(ii) efficiency**, measured by **wall-clock latency** (reported per entity pair) and, where informative, separated into retrieval, enrichment, and generation contributions; and **(iii) cost-related indicators**, where applicable, captured via retrieval workload (e.g., number of retrieval calls) to reflect blocking-induced overhead and amortization. Token counts are used internally to estimate LLM inference time, but are not reported separately.

Blocking Methods. We implement blocking using `pyJedAI` [38], a widely used open-source EM toolkit, to ensure a reproducible and standardized implementation. We primarily adopt *Q-Gram blocking* because it is a common and robust blocking technique for noisy textual attributes (e.g., typos and lexical variation), which are prevalent in EM benchmarks [43]. To assess robustness to the choice of blocking strategy, we further consider two widely used alternatives: *Standard Blocking (StdBlck)* and *Extended Q-Gram blocking (XQGram)* [43**?**]. We quantify the impact of blocking choices in Exp-4.

Backbone LLMs. We evaluate both commercial and open-source LLM backbones. Since *CE-RAG4EM* targets cost-aware EM, we focus on lightweight models that are explicitly positioned as cost-efficient and low-latency in their respective ecosystems. For commercial LLMs, we consider GPT-4o-mini and Gemini 2.0 Flash-Lite, which are designed for cost-efficient inference and fast response times; these models are accessed via the OpenAI API and Google's Gemini AI APIs. For open-source LLMs, we evaluate Qwen3-4B and Qwen3-8B from the Qwen3 family [57], deployed locally using vLLM [24] for efficient inference. Unless otherwise stated, GPT-4o-mini is used as the default backbone throughout the experiments.

Hyperparameters. We use standard decoding controls for all LLMs. Temperature and nucleus sampling (top-p) regulate randomness in generation, while k_{decode} limits sampling to the k most probable next tokens; together they control the determinism and diversity trade-off of model outputs. We cap the maximum generation length at 1024 tokens for all LLMs to bound cost and latency. For commercial API models (OpenAI and Gemini), we tune decoding hyperparameters on AMGO under the LLM-EM setting and reuse the selected configuration across all datasets and experiments (temperature = 0.5, top-p = 0.8, k_{decode} = 20). We intentionally avoid per-dataset tuning to prevent overfitting to individual benchmarks and to ensure fair, comparable evaluation across datasets; AMGO is commonly reported as a challenging dataset [46, 70], so tuning on AMGO provides a conservative configuration that transfers to other domains. For Qwen models, we follow the official recommended decoding setup (temperature = 0.7, top-p = 0.8, k_{dec0de} = 20). We use `max_bs` = 6 as the default maximum block size, and explore alternative block sizes in Exp-5 to map the quality and latency/cost tradeoff induced by this parameter.



(a) End-to-end matching time per entity pair.

(b) Amortized training time (PLMs) vs. retrieval+expansion time (CE-RAG4EM).

Figure 3: Efficiency comparison of CE-RAG4EM against (a) LLM-EM and (b) PLM baselines.

4.2 Exp-1: Overall Effectiveness

Research Question. *How does CE-RAG4EM compare to (i) LLM-only prompting and (ii) supervised PLM-based entity matching in terms of matching performance, under realistic labeling assumptions?*

Evaluation Protocol. We compare *CE-RAG4EM* against two baseline families: (i) *LLM-EM*, which prompts the same backbone LLM using only the input record pair without extra context; and (ii) *PLM-based EM*, including *Ditto* [26] and *Unicorn* [60]. For *CE-RAG4EM*, we evaluate different retrieval granularities among Entity, Predicate, and Triple. Since our study uses a single predefined test partition, we report *CE-RAG4EM (best-of)* in this experiment: for each dataset, we report the best-performing variant among these configurations as an upper envelope of achievable effectiveness. We use *best-of* only for this summary experiment; the remainder of the paper reports fixed configurations and per-factor analyses without post-hoc selection. For *Ditto* and *Unicorn*, we use the official implementations and adopt the *leave-one-dataset-out* protocol of [70]: for each target dataset, the PLM is trained on the remaining datasets and evaluated on the held-out dataset, using default training hyperparameters. We use this cross-domain setting because the standard in-dataset

[2] https://www.wikidata.org/wiki/Wikidata:Embedding_Project
[3] https://www.wikidata.org/wiki/Wikidata:REST_API

supervised protocol can yield strong PLM performance but assumes labeled data are available for every new domain; in contrast, our target scenario is LLM/RAG-style EM, where labels may be missing or costly to obtain. We report effectiveness primarily using F1 (with Precision/Recall for completeness). For efficiency, we report (a) end-to-end latency per entity pair for *CE-RAG4EM* vs. *LLM-EM*, and (b) amortized per-pair PLM training time versus the retrieval and enrichment overhead of *CE-RAG4EM*.

Table 3: F1/Precision/Recall of CE-RAG4EM vs. LLM-EM.

Dataset	F1 (%)		Precision (%)		Recall (%)	
	CE-RAG4EM	LLM-EM	CE-RAG4EM	LLM-EM	CE-RAG4EM	LLM-EM
DBGO	80.77 (+24.22)	65.02	92.45 (-0.73)	93.13	71.71 (+43.47)	49.98
ITAM	72.61 (+12.70)	64.43	97.62 (+0.18)	97.44	58.03 (+20.52)	48.15
FOZA	83.11 (+10.34)	75.32	100.00 (0.00)	100.00	71.21 (+17.50)	60.61
AMGO	55.47 (+14.02)	48.65	51.40 (-17.60)	62.37	60.26 (+55.00)	38.89
BEER	73.49 (+8.73)	67.59	96.67 (+9.32)	88.43	59.52 (+8.70)	54.76
DBAC	81.87 (+7.12)	76.43	95.14 (-1.05)	96.15	71.85 (+13.26)	63.44
WAAM	74.85 (+5.84)	70.72	84.59 (-1.13)	85.56	67.18 (+14.08)	58.89
WDC	73.55 (+5.42)	69.77	81.74 (-2.07)	83.47	66.53 (+11.01)	59.93
ABT	78.21 (+2.57)	76.25	91.27 (-2.35)	93.47	69.26 (+7.21)	64.60

Table 4: F1/Precision/Recall of CE-RAG4EM vs. PLMs (Ditto, Unicorn). Datasets are sorted by CE-RAG4EM advantage; rows where CE-RAG4EM beats both PLMs appear first.

Dataset	F1 (%)			Precision (%)			Recall (%)		
	CE-RAG4EM	Ditto	Unicorn	CE-RAG4EM	Ditto	Unicorn	CE-RAG4EM	Ditto	Unicorn
WAAM	74.85	56.50	61.47	84.59	64.32	70.99	67.18	50.37	68.01
ITAM	72.61	64.33	67.45	97.62	67.32	68.12	58.03	61.59	66.79
WDC	73.35	45.16	70.03	81.74	49.89	70.24	66.53	41.26	69.83
DBGO	80.77	77.62	78.06	92.45	81.23	81.71	71.71	74.31	74.72
FOZA	83.11	69.64	82.61	100.00	89.16	90.13	71.21	57.13	76.25
ABT	78.72	67.50	78.72	91.27	64.71	89.41	69.26	59.43	70.32
DBAC	81.87	82.96	88.72	95.14	87.65	92.12	71.85	78.76	85.57
BEER	73.49	84.19	82.20	96.67	87.33	84.39	59.52	81.27	80.13
AMGO	55.47	53.86	68.86	51.40	56.70	67.97	60.26	51.29	69.77

Results and Analysis. We compare and analyze the results of *CE-RAG4EM* between LLM-EM and PLM-EM, respectively.

CE-RAG4EM vs. LLM-EM. Table 3 shows that *CE-RAG4EM* outperforms *LLM-EM* on all nine datasets in terms of F1. The gains are primarily recall-driven: across datasets, *CE-RAG4EM* consistently increases recall, indicating that retrieved external evidence helps the LLM identify additional true matches that are missed under direct prompting. On several datasets (e.g., BEER, FOZA, and ITAM), *CE-RAG4EM* improves F1 while maintaining very high precision, suggesting that retrieval provides complementary, relevant signals without introducing many false positives. In contrast, on datasets such as AMGO, WAAM, and WDC, we observe a drop in precision relative to *LLM-EM* even though F1 increases overall. A plausible explanation is that retrieved context can occasionally include noisy or weakly related evidence; when record pairs are ambiguous, especially in datasets with mixed textual and numeric attributes, such evidence may bias the LLM toward predicting matches, increasing false positives. Overall, these results highlight a precision and recall trade-off inherent to RAG-based EM: retrieval can substantially improve coverage (recall), but its benefits depend on the relevance of the retrieved context, motivating the later analysis of retrieval granularity and context construction.

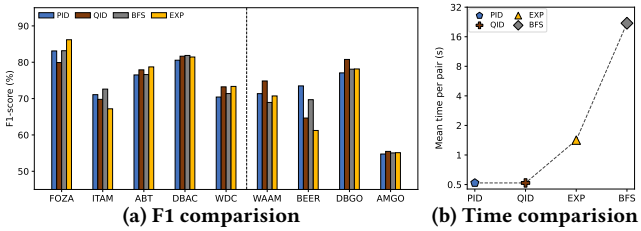
Figure 3a shows that *CE-RAG4EM* often reduces the dominant inference cost relative to *LLM-EM*. Although retrieval and enrichment add extra steps, the augmented context typically enables the LLM to produce shorter outputs (i.e., fewer generated tokens), which reduces generation time, cost, and can lead to a net end-to-end speedup. In several datasets (e.g., DBGO and BEER), this inference-time reduction is large enough to offset the additional overhead of retrieval and enrichment, yielding lower overall latency per pair. Across datasets, retrieval itself consistently accounts for only a small fraction of the end-to-end time, suggesting that vector search is not the primary bottleneck under our setup. In contrast, enrichment time can become substantial on some datasets and may dominate the pipeline, indicating that the current implementation of enrichment (e.g., resolving identifiers and expanding neighborhood information) is a key target for further optimization in later design-space analysis of context construction, as it directly affects overhead.

CE-RAG4EM vs. PLM-EM. Table 4 compares *CE-RAG4EM* with supervised PLM baselines (*Ditto* and *Unicorn*) under the leave-one-dataset-out protocol. Overall, *CE-RAG4EM* remains competitive without requiring target-domain labels: it outperforms PLMs on several datasets (e.g., WAAM, ITAM, WDC) and is comparable on others (e.g., DBGO, FOZA, ABT). This suggests that external knowledge retrieval can partially substitute for target-domain supervision in specific cases. PLMs are stronger on DBAC, BEER, and AMGO, which clarifies when supervised transfer is advantageous. On DBAC, PLMs likely benefit from transferable schema-level cues that generalize well across citation-style datasets. On BEER, *CE-RAG4EM* achieves very high precision but lower recall, suggesting that retrieval provides strong evidence for only a subset of true matches, leading to conservative match decisions and missed positives when evidence is absent or not retrieved. On AMGO, *CE-RAG4EM* shows lower precision than *Unicorn*, consistent with the hypothesis that ambiguous records and mixed attribute types increase the chance of weakly relevant retrieved context, which can bias the LLM toward over-matching and introduce additional false positives. Figure 3b compares amortized per-pair PLM training cost (*Ditto* / *Unicorn*) with the per-pair retrieval+enrichment overhead of *CE-RAG4EM*. Since PLM inference (local) is not directly comparable to closed-source API inference, we focus on the per-dataset "setup" overhead: PLMs pay an upfront training cost (amortized over test pairs), whereas *CE-RAG4EM* pays a per-pair augmentation cost. A clear trend is that on smaller datasets, the amortized PLM training overhead is large, while *CE-RAG4EM*'s overhead is comparatively stable and is dominated by enrichment (i.e., expansion of retrieved knowledge). Moreover, on six out of nine datasets, such as WAAM and ITAM, the per-pair retrieval+enrichment overhead of *CE-RAG4EM* is lower than the amortized training overhead of *Ditto*/*Unicorn*, implying that retrieval-based augmentation can incur less per-dataset overhead than cross-dataset PLM training for label-scarce workloads.

4.3 Exp-2: Retrieval Granularity

Research Question. *How does retrieval granularity affect the effectiveness and overhead of CE-RAG4EM: node-level retrieval in CE-RAG4EM-BR versus KG triple in CE-KG-RAG4EM-BR?*

Evaluation Protocol. We study retrieval granularity by comparing: (i) node-level retrieval in *CE-RAG4EM-BR*, where the retrieved context consists of either Wikidata predicates (PID) or entities (QID);



(a) F1 comparision (b) Time comparision

Figure 4: Exp-2 (Retrieval Granularity). PID/QID: node-level retrieval in *CE-RAG4EM-BR*. EXP/BFS: KG-triple context construction in *CE-KG-RAG4EM-BR* via expansion or BFS. (a) F1 by dataset (sorted by KG-variant advantage). (b) Mean context-construction time per entity pair.

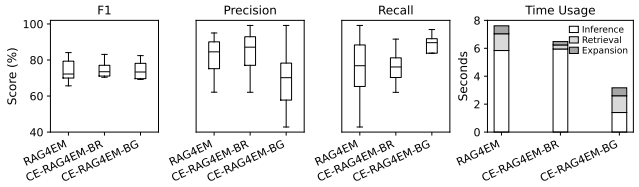


Figure 5: Exp-3 (Batching vs. per-query). F1/Prec./Rec. and end-to-end time per pair for *RAG4EM*, *CE-RAG4EM-BR* (retrieval by blocks), and *CE-RAG4EM-BG* (generation by blocks). Batched costs uniformly amortized per (sub-)block.

and (ii) triple-level retrieval in *CE-KG-RAG4EM-BR*, where the context is constructed as a small set of Wikidata triples generated via either expansion-based traversal (EXP) or breadth-first search (BFS) starting from the retrieved nodes. Across all variants, we control the retrieval budget by using the Top-k = 2 retrieved nodes to construct the context. We compare effectiveness using F1 across datasets. For efficiency, since all variants perform the same Top-k retrieval and have similar token budgets in our setup, we focus on the additional triple construction/enrichment overhead to isolate the cost introduced by KG-based context.

Results and Analysis. Figure 4a compares node-level retrieval (*CE-RAG4EM-BR* with PID/QID) and triple-level retrieval (*CE-KG-RAG4EM-BR* with EXP/BFS). Overall, KG-based triple construction is most beneficial on FOZA, ITAM, ABT, DBAC, and WDC, while node-level retrieval is competitive or better on the remaining datasets. A plausible explanation is that these "KG-friendly" datasets contain sparse or ambiguous attribute descriptions (e.g., short names, missing identifiers, mixed fields), where a small set of relational triples provides disambiguating context (e.g., type, brand/artist/venue, location), whereas isolated entities/predicates may be insufficient to resolve ambiguity. Across datasets, EXP is frequently among the top-performing strategies and is the best choice on several datasets. This can be attribute to EXP typically constructs a smaller, more focused triple context around the retrieved nodes compared to BFS, which can improve relevance (and thus reduce noise) when the initial retrieval is accurate; however, because it relies more heavily on the quality of the starting nodes and a limited expansion budget, its gains can vary across datasets.

Figure 4b shows the corresponding overhead trends. PID and QID incur small extra cost, while KG-based methods introduce additional triple-construction time, with BFS being the most expensive. This cost difference matches the construction behavior: EXP limits expansion to a small, focused neighborhood, whereas BFS explores more broadly and thus incurs higher API and processing overhead. In return, BFS can sometimes deliver the largest F1 gains, suggesting a quality and cost trade-off: broader traversal improves coverage but risks higher latency (and potentially more noise), while EXP provides a more stable middle ground.

4.4 Exp-3: Batch vs. Per-Query Execution

Research Question. *How do batching optimizations in CE-RAG4EM compare to per-query execution in RAG4EM in terms of effectiveness and end-to-end time per pair?*

Evaluation Protocol. We compare three RAG-based EM pipelines that differ only in how retrieval and generation are executed: (i) *RAG4EM*, which performs retrieval and generation independently for each entity pair (per-query); (ii) *CE-RAG4EM-BR*, which performs *batch retrieval* once per block and reuses the retrieved context for all pairs in the block, while generation is still executed per pair; and (iii) *CE-RAG4EM-BG*, which performs *batch generation* by sending all pairs in a block as one LLM request, while retrieval is still executed per pair. For all variants, we keep the retrieval configuration fixed and use the same context budget (Top-k = 2 retrieved Wikidata predicates, PID). We report effectiveness using F1 (with Precision/Recall for completeness). For efficiency, we report end-to-end *time per pair*. For the batched variants, we attribute shared costs (e.g., a single block-level retrieval in *CE-RAG4EM-BR* or a single block-level generation in *CE-RAG4EM-BG*) back to individual pairs by dividing the block-level time by the number of pairs within the block. We further decompose time into retrieval, expansion, and generation to identify which stage benefits most from batching.

Results and Analysis. Figure 5 compares per-query *RAG4EM* with two batching optimizations: *CE-RAG4EM-BR* (batch retrieval) and *CE-RAG4EM-BG* (batch generation). Overall, the three variants achieve similar F1 on average, indicating that batching primarily changes the *precision–recall balance* and the *system overhead* rather than shifting accuracy uniformly. In terms of effectiveness, *CE-RAG4EM-BG* tends to increase recall but can reduce precision, whereas *CE-RAG4EM-BR* preserves a more stable balance. A plausible explanation is that batch generation presents multiple pairs together in a single prompt, which can introduce *cross-pair coupling*: evidence or patterns from some pairs may influence the model's decisions on others. This can make the model more willing to predict matches, improving recall (fewer missed positives) but also increasing false positives when weakly related context is inadvertently shared or when the model adopts a more "match-biased" decision rule for consistency within the batch. In contrast, *CE-RAG4EM-BR* reuses retrieval within a block but still generates decisions per pair, which reduces LLM coupling across pairs and helps maintain precision while still benefiting from retrieval reuse. From an efficiency perspective, *CE-RAG4EM-BG* substantially reduces inference time per pair, making it highly competitive despite its weaker precision stability. This reduction is expected because batch generation amortizes fixed LLM invocation overhead across multiple pairs and can shorten total decoding by producing a compact batched output format. These results suggest that batch generation is a promising system optimization, but it requires careful prompt and output

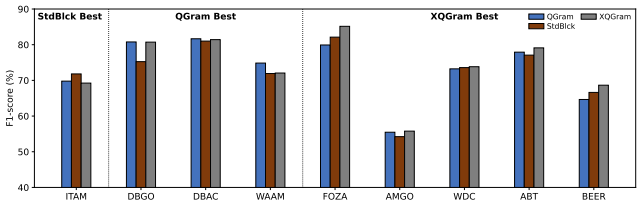


Figure 6: Exp-4 (Blocking Strategy Robustness). F1 comparison of *CE-RAG4EM-BR* under three blocking methods. Datasets are grouped according to the blocking method that achieves the best performance (separated by dotted lines).

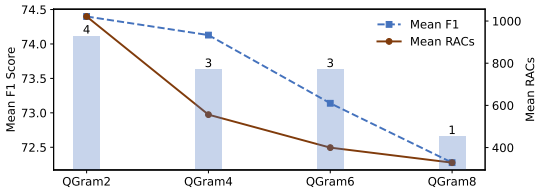


Figure 7: Exp-5 (Block Size Sensitivity). Effectiveness–efficiency trade-off under `QGram` with the max block size varies in $\{2, 4, 6, 8\}$. Solid line: mean F1 (left y-axis). Dashed line: mean retrieval API calls (`RACs`, right y-axis). Numbers above bars: datasets with best F1 at each block size.

design (e.g., stronger per-pair isolation or calibration) to avoid precision degradation while retaining its large runtime advantages.

4.5 Exp-4: Blocking Strategy Robustness

Research Question. *How sensitive is CE-RAG4EM to the choice of blocking method under batch retrieval?*

Evaluation Protocol. We evaluate robustness to blocking under the batch-retrieval setting. For each dataset, we generate blocks using three unsupervised blocking methods: `StdBlck` (standard token-based blocking), `QGram` (character q-gram blocking), and `XQGram` (extended q-gram blocking). To isolate the impact of blocking, we fix the remainder of the pipeline: all configurations use GPT-4o-mini and *CE-RAG4EM-BR(QID)* with Top-k = 2 retrieved Wikidata items per block, using the same aggregated query construction, refinement step, and prompting template.

Results and Analysis. Figure 6 reports the `F1` of *CE-RAG4EM-BR(QID)* under the three blocking methods. Overall, performance is stable across blocking choices, and q-gram based methods (`QGram`/`XQGram`) outperform `StdBlck` on most datasets. A plausible explanation is that many benchmarks contain noisy textual fields and formatting variation (Table 2); exact token-based blocking can fragment near-duplicates into different blocks, while q-gram signatures tolerate typos and lexical variation and thus produce more coherent batches for block-level retrieval. Among the q-gram methods, `XQGram` is often best or close to best, while `QGram` remains competitive and is therefore used as our default in the main experiments. Although blocking time is generally small compared to retrieval and generation, we consistently observe that `QGram` is more efficient than `XQGram` in our implementation, making it a better overall operating point when factoring in both effectiveness and runtime. Since *CE-RAG4EM-BR* retrieves context once per block, the blocking method must balance block purity and coverage: overly broad blocks can mix heterogeneous pairs and introduce less relevant evidence, while overly fragmented blocks reduce the chance that related pairs share a block and benefit from the retrieved context. In this trade-off, `XQGram`'s more discriminative signatures can improve purity on some datasets, whereas `QGram` provides a simpler, faster, and robust default that performs well across diverse domains, as consistently observed throughout our experiments.

4.6 Exp-5: Block Size Sensitivity

Research Question. *How does block size impact the effectiveness–efficiency trade-off in CE-RAG4EM?*

Evaluation Protocol We evaluate the sensitivity of *CE-RAG4EM-BR* to the maximum block size parameter (*max_bs*), which controls the upper bound on the number of candidate pairs processed per block and thus directly affects the granularity of batch retrieval. For each dataset, we run *CE-RAG4EM-BR* with `QGram` blocking while varying *max_bs* $\in \{2, 4, 6, 8\}$, with the retrieved context consisting of the Top-2 Wikidata items (`QID`) with their textual descriptions. Notably, when a raw block exceeds *max_bs*, we apply the same threshold-based decomposition strategy in §3.3 to split it into non-overlapping sub-blocks, each containing at most *max_bs* pairs. We quantify the effectiveness–efficiency trade-off using (i) matching quality measured by `F1`, and (ii) retrieval cost measured by the number of retrieval API calls (`RACs`), which captures the degree of retrieval reuse achieved by batch retrieval at each block size.

Results Figure 7 summarizes the effectiveness–efficiency trade-off as we vary the maximum block size *max_bs* under `QGram` blocking. Overall, increasing *max_bs* reduces retrieval overhead monotonically, as larger blocks enable more reuse of a single retrieval result across multiple pairs (fewer `RACs`). In contrast, matching quality degrades more gradually as *max_bs* grows. This reflects a clear trade-off: larger blocks improve efficiency by reducing retrieval calls, but they can dilute query specificity and yield less targeted context for some pairs, which may slightly hurt effectiveness. Among the tested configurations, *max_bs* $\in \{4, 6\}$ provides the best balance: it maintains near-peak average `F1` while substantially reducing `RACs`.

4.7 Exp-6: KG-RAG Design Choices

Research Question. *Which graph traversal strategy and triple budget are more effective for KG-RAG-based entity matching?*

Evaluation Protocol. We evaluate KG-RAG design choices by instantiating *CE-KG-RAG4EM-BR* under the same batch-retrieval setting and varying only the triple construction procedure. For each block, we retrieve the top ranked Wikidata items via vector search and use them as seeds to construct candidate triples using either expansion-based traversal (EXP) or breadth-first search (BFS). To control prompt length and reduce noise, we apply the same ranking-based subgraph refinement for both traversal strategies. We further vary the retained triple budget, keeping the Top-k refined triples with $k \in \{1, 2\}$, resulting in four configurations: EXP Top-1, EXP Top-2, BFS Top-1, and BFS Top-2.

Results and Analysis. Figure 8 reports dataset-level paired `F1` differences for two KG-RAG design axes: (i) traversal strategy (EXP vs. BFS) under a fixed triple budget, and (ii) triple budget (Top-2 vs. Top-1) under a fixed traversal strategy. Each point denotes a dataset-level paired difference and the diamond indicates the mean.

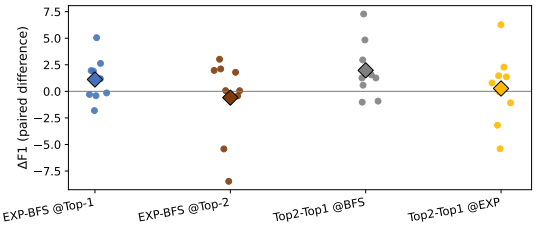


Figure 8: Exp-6 (KG-RAG design choices). Paired ΔF1 across datasets for traversal strategy (EXP−BFS at Top-1/Top-2) and triple budget (Top-2−Top-1 within BFS/EXP). Each dot is one dataset; diamonds show mean differences.

Traversal strategy (EXP vs. BFS). At a fixed triple budget, EXP and BFS exhibit mixed wins across datasets, and neither traversal dominates consistently. A plausible explanation is that EXP constructs a concise 1-hop neighborhood around the retrieved seed QIDs, which is particularly effective when matching relies on strong, near-exact identifiers (e.g., addresses, phone numbers, prices) and when the retrieved seeds are accurate, since it provides direct contextual grounding without introducing intermediate entities that may add noise. In contrast, BFS can incorporate multi-hop relations that are useful when records are mix-heavy with text and numeric attributes and semantically ambiguous, where additional relational context helps the LLM resolve implicit connections via deeper reasoning; however, this broader traversal also increases the risk of off-topic relations, making its benefit more dataset-dependent.

Triple budget (Top-2 vs. Top-1). Increasing the triple budget from Top-1 to Top-2 tends to help BFS more than EXP. We hypothesize that BFS benefits from retaining multiple complementary triples along different paths: with only a single triple, BFS may miss a critical relation, whereas a second refined triple improves evidence coverage for ambiguous cases. EXP, by design, prioritizes the most salient one-hop relations early, so adding a second triple often yields diminishing returns unless the additional triple provides a genuinely complementary attribute cue. Overall, these trends suggest a quality–noise trade-off: larger triple budgets can improve coverage and recall, but they also require effective refinement to avoid introducing irrelevant knowledge.

Finally, although increasing the BFS triple budget can make BFS competitive with or even better than EXP on many datasets, this improvement comes at a substantial expansion cost: constructing and refining a larger multi-hop neighborhood significantly increases overhead. This trend over expansion time is consistent with Exp-2 (as shown in Figure 3b), where BFS incurs markedly higher expansion/enrichment time than lightweight expansion EXP.

4.8 Exp-7: Backbone LLM Generalization

Research Question. *Do the benefits of CE-RAG4EM persist across backbone LLMs with different sizes and architectures, and what factors explain variation in the gains?*

Evaluation Protocol. We evaluate backbone generalization by instantiating the same *CE-RAG4EM-BR* pipeline with (Top-1 Wikidata item, `QID` on four backbone LLMs: GPT-4o-mini, Gemini-2.0-flash-lite, Qwen3-4B, and Qwen3-8B. For each backbone, we measure effectiveness for the *base* (LLM-only) setting and the corresponding

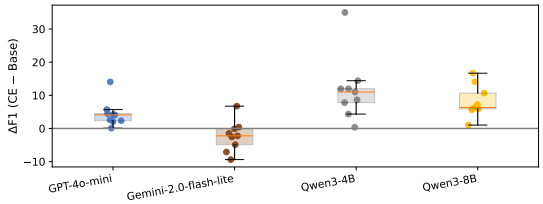


Figure 9: Exp-7 (Backbone LLM generalization): ΔF1 distribution by backbone, where ΔF1=F1(*CE-RAG4EM*)−F1(*Base*). Dots are datasets; boxplots show median and interquartile range. ΔF1> 0 indicates improvement.

CE-RAG4EM setting, and summarize the effect of retrieval using paired differences ΔF1 = F1(*CE-RAG4EM*) − F1(*Base*) (Figure 9).

Results and Analysis. Figure 9 shows that *CE-RAG4EM-BR* yields predominantly positive ΔF1 across datasets for GPT-4o-mini and both Qwen3 backbones, indicating that lightweight grounding transfers across architectures. A key reason is that entity matching often requires resolving ambiguities that are not fully determined by the record pair alone (e.g., aliasing, incomplete descriptions, or domain-specific identifiers). Injecting a relevant entity description provides an external "anchor" that reduces uncertainty and helps the model align attributes more consistently, which benefits both API models (GPT-4o-mini) and open-source models (Qwen3).

The gains are more pronounced and less stable for smaller backbones (notably Qwen3-4B). This pattern is consistent with a *capacity/knowledge* hypothesis: smaller models have less parametric knowledge and weaker long-context reasoning, so they benefit more from explicit, structured evidence that narrows the hypothesis space. As model capacity increases (e.g., Qwen3-8B and GPT-4o-mini), the base setting is already stronger, so retrieval yields smaller but still generally positive improvements—suggesting diminishing returns when the backbone can already infer many matches from surface cues. Gemini-2.0-flash-lite exhibits higher variance, with ΔF1 values closer to zero and occasional degradations. One plausible explanation is *evidence utilization*: different model families may differ in how they prioritize external context relative to the input record pair under the same prompt and budget. When the retrieved entity description is highly relevant, it helps; when it is weakly related (e.g., due to ambiguous mentions or noisy attributes), some models may over-weight the added context and drift toward incorrect matches, producing negative ΔF1 on a subset of datasets. This observation aligns with our earlier findings that retrieval quality and context relevance are key determinants of RAG-based EM performance, and suggests that backbone-specific prompting or evidence filtering could further stabilize gains. Overall, Exp-7 demonstrates that *CE-RAG4EM* generalizes across heterogeneous backbones, while also revealing systematic variation: smaller open-source models benefit more from external grounding, whereas some lightweight commercial backbones show greater sensitivity to context quality under a fixed retrieval budget.

5 Discussion and Recommendation

Drawing on extensive experiments across datasets with diverse attribute types, blocking-based optimization settings, and retrieval granularities, we distill key empirical findings and outline strategic design recommendations for future RAG4EM development.

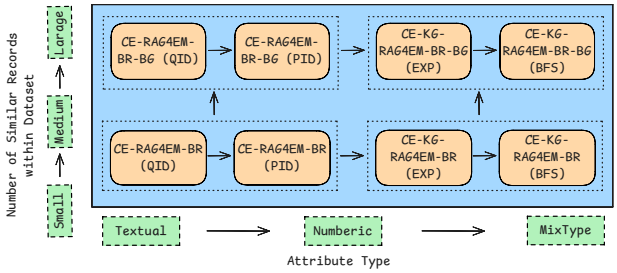


Figure 10: Recommended Design Choices of *CE-RAG4EM*.

5.1 Summary of Empirical Findings

F1: Blocking-based Batch Optimization Trade-off: Blocking Size *vs.* Performance. Blocking-based optimization reduces retrieval and inference costs in *CE-RAG4EM* by sharing prompt instructions and retrieved knowledge across batch queries. However, block size is a key hyperparameter: F1 drops once it exceeds a threshold (e.g., 6), even though retrieval and inference costs continue to decline. This F1 loss stems from noisy shared context and from input-length limits in both the embedding model used for vector search and the backbone LLMs' context windows.

F2: Blocking Strategy Robustness. Because *CE-RAG4EM*'s F1 gains rely on the purity and coverage of the blocks produced by the blocking, robust blocking methods are essential for generating high-quality blocks that enable effective batch retrieval and inference.

F3: Retrieval Granularity Trade-off: Node *vs* Triple. Our comparison of node-level retrieval and triple-based retrieval (BFS/EXP) shows a clear F1-cost trade-off. *CE-KG-RAG4EM* achieves higher recall on ambiguous pairs–particularly those with numeric or mixed-type attributes–by grounding LLMs in the latent relationships encoded in triples, though this introduces additional triple-search and traversal overhead. Consequently, the optimal retrieval granularity depends on attribute diversity, with *KG-RAG4EM* delivering stronger F1 on records containing mixed numeric and textual attributes (e.g., identifiers) than node-level RAG4EM.

F4: Triple Search and Traversal Trade-off. In *KG-RAG4EM*, deeper searches such as BFS offer useful multi-hop context for uncovering implicit connections but also introduce more irrelevant noise than local neighborhood expansion. Although greater depth improves coverage, it substantially increases triple-search and enrichment time. Balancing implicit knowledge against noise, therefore, requires an appropriate trade-off between search strategy and depth to maintain both F1 and computational efficiency.

F5: Language model-level trade-offs: Peformance *vs.* Model Size. *RAG4EM* delivers consistent F1 gains across model families and sizes. It provides the largest F1 improvements for smaller open-source models (e.g., Qwen3-4B) and mid-sized commercial models (e.g., GPT-4o-mini) with low inference cost. Medium-sized models (e.g., Gemini-2.0-flash-lite, Qwen3-8B) achieve even higher but more variable F1 gains with higher inference cost. These highlight a clear F1 gains and cost trade-off, showing that RAG4EM is well-suited for lightweight models with computational constraints.

5.2 Recommendation for Design Choices

Based on our empirical analysis, we outline the following design recommendations for building RAG-based entity matching systems.

R1: Optimize Block Size to Balance Performance and Cost. We recommend tuning the maximum block size dynamically based on the attribute types present in the records and the context-window limits of the backbone models. Empirically, a block-size range of 4–6 provides strong retrieval reuse and reduces retrieval and inference cost without compromising F1.

R2: Prioritize Batch Retrieval and Inference with Robust Blocking. Because *CE-RAG4EM*'s performance gains depend heavily on the quality and coverage of matching pairs produced by the blocking strategy, prioritizing a robust blocking approach is essential for effective batch retrieval and generation. Moreover, batch retrieval should serve as the default configuration in *RAG4EM*, as it lowers retrieval costs while leveraging shared contextual knowledge and preserving inference independence.

R3: Adopt a Context-adaptive Retrieval Granularity. To balance F1 with retrieval and triple-search cost, KG-RAG configurations using EXP or BFS should not be applied uniformly across datasets. A context-adaptive granularity strategy is preferable: use node-level retrieval for high-confidence blocks with textual or date attributes, and reserve triple-level traversal (BFS) for ambiguous cases involving numeric, categorical, or mixed attributes.

R4: Follow Filter-then-Reasoning Pipeline for KG-RAG. Introducing a knowledge-refinement mechanism is essential for filtering noise, as graph traversals can surface broad contextual information that may introduce semantic distractions and mislead inference. We recommend a hybrid filtering strategy: combine ranking-based triple refinement with instruction-guided prompt filtering to remove noisy knowledge at multiple stages before LLM inference.

R5: Prioritize Small Then Medium Models. Lightweight backbone models are prioritized in *RAG4EM*, as they deliver strong performance gains when grounded with high-value contextual knowledge, while maintaining an F1-cost balance. Medium or large models are best reserved for highly complex matching cases that require intensive reasoning based on the LLMs' internal knowledge.

We summarize our empirical findings in a decision matrix that recommends representative *CE-RAG4EM* design choices along two dimensions: attribute type (textual vs. numeric/mixed) and the volume of similar records (small vs. large). As illustrated in Figure 10, retrieval granularity should shift from lightweight node-level (QID/PID) to deeper triple-level contexts (EXP/BFS) as attribute diversity increases and matching becomes more ambiguous. For datasets with many similar records, we further recommend moving from batch retrieval alone to combined batch retrieval and batch generation to reduce overall retrieval and inference cost.

6 Related Work

This work focuses on designing a blocking-based cost-efficient RAG4EM. Thereby, the related work is summarized as follows.

6.1 Entity Matching

PLM and LLM-based Entity Matching. In this approach, the entity matching task is generally modeled as a binary sequence-pair classification task, while the trained language model aims to capture complex contextual understandings of matching records and generate the answer for pairwise matching pairs.

(1) *PLM-based Entity Matching*: Transformer-based PLMs such as RoBERTa [10, 60], BERT [26, 40, 44, 60], DistilBERT [8], and

GPT-2 [69] have been applied to entity matching. GraLMatch [8] uses transitivity and graph-based context to reduce false positives and fine-tunes DistilBERT with optimizations on limited labeled data for group matching across sources. To enhance contextual understanding, strategies such as knowledge distillation [10], knowledge injection [26], and fine-tuning [44, 60] have been introduced. SETEM [10] combines self-network mixing, knowledge transfer, and self-ensembling training to improve PLM efficiency with limited labeled data. However, fine-tuning and training PLMs for entity matching still demand large amounts of labeled data, particularly for task-specific and domain-specific applications.

(2) *LLM-based Entity Matching*: Several LLM techniques, including prompt engineering, fine-tuning, and in-context learning, are studied for entity matching. SerializeEM [66] introduces random walk-based entity serialization with a graph structure to capture deeper semantic context. To address the tendency of LLMs to generate negative answers, COMEM [64] evaluates multiple prompting strategies–LLM as matcher, comparator, and selector–and integrates them with a ranking-based filter for robust and cost-efficient matching. Mistral4SelectEM [50] further improves performance by structuring selective entity matching into a Siamese network and fine-tuning it with contrastive margin ranking loss to better distinguish true positives from similar negatives.

Blocking for Entity Matching. Blocking is used to remove likely non-matching pairs when creating candidate pairs from raw tables [26, 60] or to select subsets of pairs through combined strategies [8], which reduces pairwise comparisons and complexity in entity matching. Blocking methods include heuristic rule-based, traditional, clustering-based, and machine learning or deep learning approaches [33, 58, 62]. Rule-based blocking requires expert knowledge to define rules, while machine learning-based blocking needs labeled data and expensive computation. Traditional blocking is simpler and efficient, using blocking functions to create blocking key values (BKV) and grouping entities with equal or similar BKVs into blocks, such as standard blocking, sorted neighborhood, Q-gram, and suffix blocking [3]. However, traditional blocking scales poorly, works best on small datasets [33], and often groups the same entities into multiple blocks because it fails to capture attribute semantics [3]. Although blocking is widely studied for reducing comparisons in entity matching [38], its use for batch retrieval in RAG and KG-RAG has not been explored.

6.2 RAG, GraphRAG, and KG-RAG

RAG aims to guide LLMs to generate the correct answer by retrieving the relevant contexts from large textual documents and incorporating them with the original query, while GraphRAG and KG-RAG enhance LLMs by integrating structured graph-based knowledge rather than relying on textual chunks, thereby improving their performance in complex and knowledge-intensive tasks.

RAG. RAG [25] system usually retrieves the relevant context from the textual document and chunks based on vector search and ranking, incorporates the retrieved context based on prompt-instruction, and then feed to LLM to mitigate the hallucinations of LLLMs. In view of RAC excels in a superior capability in grounding LLM for reasoning, RAG approaches have been investigated in knowledge-intensive tasks, such as natural language understanding [15], question answering [25, 31], etc. To address the issue that the retrieved context may mislead LLMs to generate icorrect answer, a SELF-RAG [2] is proposed by training as a language model that adaptively retrieves the context from external knowledge bases only if it is necessary. However, the performance gains of naive RAG approaches remain limited, as LLMs often exhibit weaker reasoning capabilities when incorporating the retrieved unstructured text compared to structured knowledge.

GraphRAG and KG-RAG. GraphRAG augments LLMs by enabling contextual and multi-hop reasoning through relevant subgraphs built from textual documents, where graphs are converted into hierarchical descriptions using prompting [19]. It is applied in knowledge-intensive tasks such as question answering [37, 65] and recommendations [47]. However, irrelevant graph context and noisy knowledge reduce LLM performance and may cause hallucination. To address this, ranking-based graph filtering methods are proposed to remove noisy and irrelevant context before integration [17, 68]. Yotu-GraphRAG [12] further enhances reasoning by adding agents for graph construction and retrieval. Despite these advances, constructing graph data from large textual documents remains time-consuming, as entity and relationship extraction requires significant computing resources and domain expertise. Instead of retrieving subgraphs from a graph that was built based on a textual document in GraphRAG, KG-RAG augments LLMs with relevant subgraphs or triples retrieved from curated factual knowledge graphs, enabling fact-grounded responses through KG reasoning. KG-RAG is widely studied in question answering [30], recommendations [63], and data management tasks such as schema matching [29]. Despite these advances, indexing and retrieving subgraphs from large-scale KGs with billions of triples remain computationally intensive [30], as vector search over large embedding spaces is costly. Moreover, both GraphRAG and KG-RAG require subgraph retrieval for each query, which increases retrieval costs.

7 Conclusion

In this paper, we addressed the bottleneck of computational inefficiency in adapting RAG for entity matching by introducing novel *CE-RAG4EM* with blocking-based batch retrieval and inference. We built a unified framework for analyzing and evaluating *CE-RAG4EM* with diverse retrieval granularity. Our extensive evaluations demonstrate that *CE-RAG4EM* not only significantly reduces the retrieval and inference cost, but also allows smaller open-source models to compete with larger LLMs. The empirical findings offer a practical guideline for building scalable, cost-efficient, and highly reliable RAG systems for data integration in real-world data engineering.

References

[1] Ioannis Arvanitis-Kasinikos and George Papadakis. 2025. Entity Matching with 7B LLMs: A Study on Prompting Strategies and Hardware Limitations. In *DOLAP (CEUR Workshop Proceedings)*, Vol. 3931. 31–38.

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *ICLR*.

[3] Fabio Azzalini, Songle Jin, Marco Renzi, and Letizia Tanca. 2021. Blocking Techniques for Entity Linkage: A Semantics-Based Approach. *Data Sci. Eng.* 6, 1 (2021), 20–38.

[4] Nils Barlaug and Jon Atle Gulla. 2021. Neural Networks for Entity Matching: A Survey. *ACM Trans. Knowl. Discov. Data* 15, 3 (2021), 1–37.

[5] Jan-Micha Bodensohn, Ulf Brackmann, Liane Vogel, Anupam Sanghi, and Carsten Binnig. 2025. Unveiling Challenges for LLMs in Enterprise Data Engineering. *Proc. VLDB Endow.* 19, 2 (2025), 196–209.

[6] Yukun Cao, Zengyi Gao, Zhiyang Li, Xike Xie, S. Kevin Zhou, and Jianliang Xu. 2025. LEGO-GraphRAG: Modularizing Graph-based Retrieval-Augmented Generation for Design Space Exploration. *Proc. VLDB Endow.* 18, 10 (2025), 3269–3283.

[7] António Correia, Diogo Guimarães, Dennis Paulino, Shoaib Jameel, Daniel Schneider, Benjamim Fonseca, and Hugo Paredes. 2021. AuthCrowd: Author Name Disambiguation and Entity Matching using Crowdsourcing. In *CSCWD*. IEEE, 150–155.

[8] Fernando de Meer Pardo, Claude Lehmann, Dennis Gehrig, Andrea Nagy, Stefano Nicoli, Branka Hadji Misheva, Martin Braschler, and Kurt Stockinger. 2025. GraLMatch: Matching Groups of Entities with Graphs and Language Models. In *EDBT*. 1–12.

[9] Djellel Difallah. 2025. WikiRAG: Revisiting Wikidata KGC Datasets with Community Updates and Retrieval-Augmented Generation. In *KDD*. ACM, 5391–5401.

[10] Huahua Ding, Chaofan Dai, Yahui Wu, Wubin Ma, and Haohao Zhou. 2024. SETEM: Self-ensemble training with Pre-trained Language Models for Entity Matching. *Knowl. Based Syst.* 293 (2024), 111708.

[11] AnHai Doan, Pradap Konda, Paul Suganthan GC, Yash Govind, Derek Paulsen, Kaushik Chandrasekhar, Philip Martinkus, and Matthew Christie. 2020. Magellan: toward building ecosystems of entity matching solutions. *Commun. ACM* 63, 8 (2020), 83–91.

[12] Junnan Dong, Siyu An, Yifei Yu, Qian-Wen Zhang, Linhao Luo, Xiao Huang, Yunsheng Wu, Di Yin, and Xing Sun. 2025. Youtu-GraphRAG: Vertically Unified Agents for Graph Retrieval-Augmented Complex Reasoning. *CoRR* abs/2508.19855 (2025).

[13] Juliana Freire, Grace Fan, Benjamin Feuer, Christos Koutras, Yurong Liu, Eduardo Peña, Aécio S. R. Santos, Cláudio T. Silva, and Eden Wu. 2025. Large Language Models for Data Discovery and Integration: Challenges and Opportunities. *IEEE Data Eng. Bull.* 49, 1 (2025), 3–31.

[14] Jiajie Fu, Haitong Tang, Arijit Khan, Sharad Mehrotra, Xiangyu Ke, and Yunjun Gao. 2025. In-context Clustering-based Entity Resolution with Large Language Models: A Design Space Exploration. *Proc. ACM Manag. Data* (2025), 1–28.

[15] Michael R. Glass, Gaetano Rossiello, Md. Faisal Mahbub Chowdhury, and Alfio Gliozzo. 2021. Robust Retrieval Augmented Generation for Zero-shot Slot Filling. In *EMNLP*. ACL, 1939–1949.

[16] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. 2014. Corleone: hands-off crowdsourcing for entity matching. *Proc. ACM Manag. Data* (2014), 601–612.

[17] Kai Guo, Harry Shomer, Shenglai Zeng, Haoyu Han, Yu Wang, and Jiliang Tang. 2025. Empowering GraphRAG with Knowledge Filtering and Integration. *CoRR* abs/2503.13804 (2025).

[18] Haoyu Han, Harry Shomer, Yu Wang, Yongjia Lei, Kai Guo, Zhigang Hua, Bo Long, Hui Liu, and Jiliang Tang. 2025. RAG vs. GraphRAG: A Systematic Evaluation and Key Insights. *CoRR* abs/2502.11371 (2025).

[19] Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2025. GRAG: Graph Retrieval-Augmented Generation. In *NAACL (Findings)*. ACL, 4145–4157.

[20] Jiacheng Huang, Wei Hu, Zhifeng Bao, Qijin Chen, and Yuzhong Qu. 2023. Deep entity matching with adversarial active learning. *VLDB J.* 32, 1 (2023), 229–255.

[21] K. M. Sajjadul Islam, Ayesha Siddika Nipu, Jiawei Wu, and Praveen Madiraju. 2025. LLM-Based Prompt Ensemble for Reliable Medical Entity Recognition from EHRs. In *IRI*. IEEE, 162–167.

[22] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2025. RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. *ACM Trans. Comput. Syst.* 44, 1, Article 2 (2025), 27 pages.

[23] Arijit Khan, Yuyu Luo, Wenjie Zhang, Minqi Zhou, and Xiaofang Zhou. 2026. Retrieval-augmented Generation (RAG): What is There for Data Management Researchers? A discussion on research from a panel at LLM+Vector Data Workshop @ IEEE ICDE 2025. *SIGMOD Rec.* 54, 4 (2026), 33–42.

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SIGOPS*.

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.

[26] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60.

[27] Yuliang Li, Jinfeng Li, Yoshi Suhara, AnHai Doan, and Wang-Chiew Tan. 2023. Effective entity matching with transformers. *VLDB J.* 32, 6 (2023), 1215–1235.

[28] Xuanqing Liu, Runhui Wang, Yang Song, and Luyang Kong. 2024. GRAM: Generative Retrieval Augmented Matching of Data Schemas in the Context of Data Security. In *KDD*. ACM, 5476–5486.

[29] Chuangtao Ma, Sriom Chakrabarti, Arijit Khan, and Bálint Molnár. 2025. Knowledge Graph-based Retrieval-Augmented Generation for Schema Matching. *CoRR* abs/2501.08686 (2025).

[30] Chuangtao Ma, Yongrui Chen, Tianxing Wu, Arijit Khan, and Haofen Wang. 2025. Large Language Models Meet Knowledge Graphs for Question Answering: Synthesis and Opportunities. In *EMNLP*. 24578–24597.

[31] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Generation-Augmented Retrieval for Open-Domain Question Answering. In *ACL/IJCNLP*. 4089–4100.

[32] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A Comprehensive Benchmark Framework for Active Learning Methods in Entity Matching. *Proc. ACM Manag. Data* (2020), 1133–1147.

[33] Mohammad Hossein Moslemi, Harini Balamurugan, and Mostafa Milani. 2024. Evaluating Blocking Biases in Entity Matching. In *IEEE Big Data*. IEEE, 64–73.

[34] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. *Proc. ACM Manag. Data* (2018), 19–34.

[35] John Bosco Mugeni, Steven J. Lynden, Toshiyuki Amagasa, and Akiyoshi Matono. 2025. AssistEM: Domain Instruction Tuning for Enhanced Entity Matching. In *PAKDD (5) (Lecture Notes in Computer Science)*, Vol. 15874. Springer, 115–127.

[36] Navapat Nananukul, Khanin Sisaengsuwanchai, and Mayank Kejriwal. 2024. Cost-efficient prompt engineering for unsupervised entity resolution in the product matching domain. *Discov. Artif. Intell.* 4, 1 (2024), 56.

[37] Tengjun Ni, Xin Yuan, Shenghong Li, Kai Wu, Ren Ping Liu, Wei Ni, and Wenjie Zhang. 2025. StepChain GraphRAG: Reasoning Over Knowledge Graphs for Multi-Hop Question Answering. *CoRR* abs/2510.02827 (2025).

[38] Konstantinos Nikoletos, George Papadakis, and Manolis Koubarakis. 2022. pyJedAI: a Lightsaber for Link Discovery. In *ISWC (Posters/Demos/Industry) (CEUR Workshop Proceedings)*, Vol. 3254. CEUR-WS.org.

[39] Matteo Paganelli, Paolo Sottovia, Francesco Guerra, and Yannis Velegrakis. 2019. TuneR: Fine Tuning of Rule-based Entity Matchers. In *CIKM*. ACM, 2945–2948.

[40] Matteo Paganelli, Donato Tiano, and Francesco Guerra. 2024. A multi-facet analysis of BERT-based entity matching models. *VLDB J.* 33, 4 (2024), 1039–1064.

[41] Fatemah Panahi, Wentao Wu, AnHai Doan, and Jeffrey F. Naughton. 2017. Towards Interactive Debugging of Rule-based Entity Matching. In *EDBT*. OpenProceedings.org, 354–365.

[42] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. 2014. Meta-Blocking: Taking Entity Resolutionto the Next Level. *IEEE Transactions on Knowledge and Data Engineering* 26, 8 (2014), 1946–1960.

[43] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Comput. Surv.* 53, 2, Article 31 (March 2020), 42 pages.

[44] Ralph Peeters and Christian Bizer. 2021. Dual-Objective Fine-Tuning of BERT for Entity Matching. *Proc. VLDB Endow.* 14, 10 (2021), 1913–1921.

[45] Ralph Peeters and Christian Bizer. 2023. Using ChatGPT for Entity Matching. In *ADBIS (Communications in Computer and Information Science)*, Vol. 1850. Springer, 221–230.

[46] Ralph Peeters, Aaron Steiner, and Christian Bizer. 2025. Entity Matching using Large Language Models. In *EDBT*. OpenProceedings.org, 529–541.

[47] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2025. Graph Retrieval-Augmented Generation: A Survey. *ACM Trans. Inf. Syst.* 44, 2 (2025), 1–52.

[48] Anna Primpeli, Ralph Peeters, and Christian Bizer. 2019. The WDC training dataset and gold standard for large-scale product matching. In *WWW Companion*. 381–386.

[49] Kun Qian, Yisi Sang, Farima Fatahi Bayat, Anton Belyi, Xianqi Chu, Yash Govind, Samira Khorshidi, Rahul Khot, Katherine Luna, Azadeh Nikfarjam, Xiaoguang Qi, Fei Wu, Xianhan Zhang, and Yunyao Li. 2024. APE: Active Learning-based Tooling for Finding Informative Few-shot Examples for LLM-based Entity Matching. In *DaSH@ACL*. ACL, 1–3.

[50] Qian Ruan, Dachuan Shi, and Thomas Bauernhansl. 2025. Fine-tuning large language models with contrastive margin ranking loss for selective entity matching in product data integration. *Adv. Eng. Informatics* 67 (2025), 103538.

[51] Juan Sequeda, Dean Allemang, and Bryon Jacob. 2024. Increasing Accuracy of LLM-powered Question Answering on SQL databases: Knowledge Graphs to the Rescue. *IEEE Data Eng. Bull.* 48, 4 (2024), 109–134.

[52] Eitam Sheetrit, Menachem Brief, Moshik Mishaeli, and Oren Elisha. 2024. ReMatch: Retrieval Enhanced Schema Matching with LLMs. *CoRR* abs/2403.01567 (2024).

[53] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing Entity Matching Rules by Examples. *Proc. VLDB Endow.* 11, 2 (2017), 189–202.

[54] Aaron Steiner, Ralph Peeters, and Christian Bizer. 2025. Fine-Tuning Large Language Models for Entity Matching. In *ICDEW*. IEEE, 9–17.

[55] Saba Sturua, Isabelle Mohr, Mohammad Kalim Akram, Michael Günther, Bo Wang, Markus Krimmel, Feng Wang, Georgios Mastrapas, Andreas Koukounas, Nan Wang, and Han Xiao. 2025. Jina Embeddings V3: Multilingual Text Encoder with Low-Rank Adaptations. In *ECIR*. 123–129.

[56] Nan Tang, Chenyu Yang, Zhengxuan Zhang, Yuyu Luo, Ju Fan, Lei Cao, Sam Madden, and Alon Y. Halevy. 2024. Symphony: Towards Trustworthy Question Answering and Verification using RAG over Multimodal Data Lakes. *IEEE Data Eng. Bull.* 48, 4 (2024), 135–146.

[57] Qwen Team. 2025. Qwen3 Technical Report. *CoRR* abs/2505.09388 (2025).

[58] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. 2021. Deep Learning for Blocking in Entity Matching: A Design Space Exploration. *Proc. VLDB Endow.* 14, 11 (2021), 2459–2472.

[59] Rishit Toteja, Arindam Sarkar, and Prakash Mandayam Comar. 2025. In-Context Reinforcement Learning with Retrieval-Augmented Generation for Text-to-SQL. In *COLING*. ACL, 10390–10397.

[60] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A Unified Multi-tasking Model for Supporting Matching Tasks in Data Integration. *Proc. ACM Manag. Data* 1, 1 (2023), 1–26.

[61] Denny Vrandecic, Lydia Pintscher, and Markus Krötzsch. 2023. Wikidata: The Making Of. In *WWW Companion*. ACM, 615–624.

[62] Runhui Wang and Yongfeng Zhang. 2024. Pre-trained Language Models for Entity Blocking: A Reproducibility Study. In *NAACL-HLT*. ACL, 8720–8730.

[63] Shijie Wang, Wenqi Fan, Yue Feng, Shanru Lin, Xinyu Ma, Shuaiqiang Wang, and Dawei Yin. 2025. Knowledge Graph Retrieval-Augmented Generation for LLM-based Recommendation. In *ACL (1)*. ACL, 27152–27168.

[64] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xuanang Chen, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025. Match, Compare, or Select? An Investigation of Large Language Models for Entity Matching. In *COLING*. ACL, 96–109.

[65] Junde Wu, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, Yueming Jin, and Vicente Grau. 2025. Medical Graph RAG: Evidence-based Medical Large Language Model via Graph Retrieval-Augmented Generation. In *ACL (1)*. ACL, 28443–28467.

[66] Haoteng Yin, Jinha Kim, Prashant Mathur, Krishanu Sarker, and Vidit Bansal. 2025. How to Talk to Language Models: Serialization Strategies for Structured Entity Matching. In *NAACL (Findings)*. ACL, 7836–7850.

[67] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Jellyfish: A Large Language Model for Data Preprocessing. *CoRR* abs/2312.01678 (2023).

[68] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. 2025. A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models. *CoRR* abs/2501.13958 (2025).

[69] Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. 2025. AnyMatch - Efficient Zero-Shot Entity Matching with a Small Language Model. In *GOOD DATA@AAAI*.

[70] Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. 2025. A Deep Dive Into Cross-Dataset Entity Matching with Large and Small Language Models. In *EDBT*. 922–934.

[71] Xiangrong Zhu, Yuexiang Xie, Yi Liu, Yaliang Li, and Wei Hu. 2025. Knowledge Graph-Guided Retrieval Augmented Generation. In *NAACL (Long Papers)*. ACL, 8912–8924.

[72] Jiaru Zou, Dongqi Fu, Sirui Chen, Xinrui He, Zihao Li, Yada Zhu, Jiawei Han, and Jingrui He. 2025. GTR: Graph-Table-RAG for Cross-Table Question Answering. *CoRR* abs/2504.01346 (2025).