Generalized Entity Matching with Adaptivity via Large Language Models

XINGGUANG CHEN, National University of Singapore, Singapore
YIMIN SHI, National University of Singapore, Singapore
XIAOKUI XIAO*, National University of Singapore, Singapore

Entity matching is a fundamental task in data management, supporting applications such as product matching in e-commerce, linking scholarly records, unifying location data, and cross-source fact-checking. Its generalized form, known as generalized entity matching (GEM), extends the challenge to heterogeneous sources across structured, semi-structured, and unstructured data, where schema mismatch, structural variation, and domain drift often make supervision costly or infeasible. Existing approaches either rely heavily on labeled data or fail to generalize across domains, lacking the ability to use both structural and content information at scale. We present GLEAM, an end-to-end generalized entity matching framework that dynamically adapts to data structure and domain characteristics without requiring labeled pairs. GLEAM integrates (i) a structure- and content-aware blocking module that constructs a weighted heterogeneous graph encoding schema context and semantic similarity, guided by a lightweight LLM-based attribute importance estimator, (ii) an adaptive connector that models candidate score distributions via a two-component GMM and performs Bayesian updates using online feedback from matching, enabling entity-specific early stopping, and (iii) a hierarchical reasoning matching module that infers domain and attribute hierarchies from a few examples and performs comparative selection over candidate sets with domain-aware prompts. Across a diverse set of heterogeneous datasets, GLEAM consistently achieves strong end-to-end F1 while reducing unnecessary LLM calls. It outperforms leading unsupervised and LLM-based baselines and remains competitive with supervised methods, while achieving up to a 25.7% improvement in F1. We further provide probabilistic analyses of the connector's thresholding behavior and update rules, along with ablation studies on graph weighting, adaptivity, and hierarchical prompting.

CCS Concepts: • **Computing methodologies → Natural language processing**; *Probabilistic reasoning*; • **Information systems → Entity resolution**; *Mediators and data integration.*

Additional Key Words and Phrases: Generalized Entity Matching; Large Language Models; Heterogeneous Data

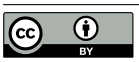
ACM Reference Format:
Xingguang Chen, Yimin Shi, and Xiaokui Xiao. 2026. Generalized Entity Matching with Adaptivity via Large Language Models. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 189 (June 2026), 25 pages.
https://doi.org/10.1145/3802066

1 Introduction

Entity matching aims to identify pairs of data entities that refer to the same real-world object. As a fundamental task in data management, entity matching supports applications such as product

*Corresponding author.

Authors' Contact Information: Xingguang Chen, National University of Singapore, Singapore, xgchen@nus.edu.sg; Yimin Shi, National University of Singapore, Singapore, yiminshi@comp.nus.edu.sg; Xiaokui Xiao, National University of Singapore, Singapore, xkxiao@nus.edu.sg.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 2836-6573/2026/6-ART189
https://doi.org/10.1145/3802066

matching in e-commerce, linking scholarly records, unifying location data, and cross-source fact-checking [2, 10, 23, 24]. However, many existing entity matching approaches assume homogeneous schemas, where entities share comparable attribute structures and semantics [12, 14, 15]. Recent schema-agnostic entity matching methods relax explicit schema alignment assumptions by relying on schema-free similarity modeling or implicit field correspondence [19]. These methods, however, typically operate on structured tables and do not explicitly address cross-structure or hierarchical heterogeneity. This limits their applicability in many practical settings. In practice, data are often heterogeneous, collected from diverse sources with varying formats and schema designs.

Consider a fact-checking scenario: when a rumor circulates online, fact-checking organizations aim to retrieve relevant debunks from multiple sources. Yet, the associated data may appear in diverse textual forms. The same rumor, along with its rephrased variants, may appear as text across heterogeneous data representations, such as a row in a relational table, a field in a JSON record, a fragment extracted from a web page, a summarized snippet on social media, or a user-generated message in instant messaging platforms. In essence, these heterogeneous textual records may encode the same underlying factual statement despite differences in structure, granularity, and schema organization. Traditional schema matching is infeasible in this setting because it requires costly manual alignment and fails on semi-structured or unstructured text. Likewise, conventional entity matching methods assume structural consistency between entities and thus struggle with cross-structure or cross-schema matching. These challenges call for a generalized framework capable of integrating structure, content, and reasoning across heterogeneous textual data sources.

Following recent studies [2, 23, 24], we focus on generalized entity matching (GEM), which identifies corresponding entities across heterogeneous representations, such as relational, semi-structured, and textual data types. In this work, we consider a fully *unsupervised* setting, where no labeled matching pairs are used at any stage of the pipeline. This setting differs from many methods evaluated under benchmarks such as Machamp [23] (e.g., PromptEM [24]), which rely on labeled seed pairs or other training signals for adaptation.

Limitations of Existing Solutions. Many existing solutions for GEM remain learning-centric and depend on some form of supervision for adaptation or calibration. This dependency becomes a major obstacle in low-resource or cross-domain settings. PromptEM [24] represents an important step toward low-resource GEM by formulating entity matching as a prompt-tuning task on pre-trained language models (LMs). By aligning LMs with the prompt-tuning objective, PromptEM bridges the adaptation gap and leverages the knowledge embedded in LMs. It also incorporates uncertainty-aware self-training and data pruning to improve pseudo-label quality and training efficiency. Despite these advantages, PromptEM remains a supervised method at its core, as it requires labeled seed pairs to guide prompt design and pseudo-label calibration. Its reliance on handcrafted prompts and domain-specific annotations limits its adaptability to unseen schemas and modalities. While it mitigates data scarcity to some extent, it still struggles in fully unsupervised or cross-domain scenarios, motivating the need for more adaptive and label-efficient GEM frameworks.

In contrast, TDmatch [2] takes an unsupervised approach to GEM by representing relational tuples, semi-structured records, and text documents within a unified heterogeneous graph. Through random walks, it learns joint embeddings that capture semantic relations between entities in different formats, enabling schema-free matching without labeled data. However, TDmatch has two key limitations. First, its graph construction lacks matching guidance, making performance unstable and sensitive to data distribution and construction heuristics. Second, the random-walk embedding process suffers from poor scalability, requiring large memory and long computation even on moderate-sized datasets. As a result, TDmatch serves as a valuable proof of concept but remains limited in both efficiency and robustness.

Meanwhile, ComEM [27] explores the potential of large language models (LLMs) for entity matching under homogeneous settings. It systematically examines several prompting strategies such as matching, comparing, and selecting, to capture relationships between records, showing that LLMs can achieve high accuracy and cost efficiency. However, ComEM assumes well-aligned schemas and structured tabular inputs, which restricts its ability to generalize to heterogeneous data. Nevertheless, its insight that LLMs can act as oracles for entity equivalence motivates us to extend this reasoning capability to more diverse GEM scenarios.

Challenges. To address the limitations of prior work and build an effective and efficient solution for GEM that operates directly on raw data, we identify three main challenges.

First, exhaustively comparing all entity pairs across datasets is computationally prohibitive, which makes blocking a necessary step in entity matching. Existing blocking techniques are mainly developed for homogeneous data and seldom consider schema heterogeneity or variation in data formats. Token based approaches such as Sparkly [17] and semantic approaches such as DeepBlocker [20] achieve strong results in traditional entity matching tasks but rely on the assumption of aligned schemas and consistent attribute meanings. At present, there is no blocking framework designed specifically for generalized entity matching, where jointly using structural and content information remains an open problem.

Second, recent advances in LLMs create new opportunities for unsupervised entity alignment, yet it remains difficult to guide LLMs to reason across heterogeneous schemas and hierarchical data. Designing prompts that help LLMs capture structural dependencies rather than shallow similarity requires further exploration.

Finally, many entity matching systems such as Magellan [10] treat blocking and matching as two independent stages, passing only candidate identifiers between them. This separation discards useful information such as similarity distributions from the blocking process, and forces practitioners to manually tune parameters like candidate counts or thresholds. A unified framework that transfers signals from blocking to matching and adapts termination decisions dynamically is essential for scalable and low-resource GEM.

Overview of Our Approach. To address these challenges, we present **G**enera**L**ized **E**ntity matching with **A**daptivity via large language **M**odels (GLEAM). Given two raw datasets, relational, semi-structured, or textual, with potentially heterogeneous schemas, GLEAM first applies a graph-based blocking strategy that jointly models structure and content under the guidance of an LLM. The LLM determines the exploration depth for each entity to generate proximity signals that represent potential candidate matches. An adaptive connector then transforms these proximity signals into entity-specific thresholds and decides, using feedback from the matching process, whether to continue or stop matching. This adaptive design reduces unnecessary computation while maintaining accuracy. Finally, hierarchical reasoning driven by an LLM performs detailed matching through a two-phase prompting procedure: triage, which organizes domain and attribute hierarchies, and selection, performing comparative reasoning among candidates.

Contributions. Our main contributions are as follows:

- We introduce GLEAM, an unsupervised, end-to-end framework for generalized entity matching that requires no labeled pairs, integrating graph-based blocking, adaptive control, and reasoning-driven LLM matching.
- We design an LLM-guided structural weighting scheme that incorporates priors on attribute importance into a heterogeneous graph to weight candidate entities during blocking.
- We propose an adaptive connector that maps proximity signals from blocking into entity-specific thresholds, with theoretical support for its update and convergence behavior in matching.

- We develop a two-stage hierarchical reasoning strategy for matching, triage (domain and attribute hierarchy) followed by selection, to effectively address schema heterogeneity.
- Extensive experiments on heterogeneous datasets demonstrate that GLEAM consistently surpasses representative unsupervised and LLM-based baselines, achieving up to 25.7% F1 improvement over the state-of-the-art supervised method, including comprehensive ablation and efficiency analyses.

2 Preliminaries

2.1 Problem Statement

Generalized Entity Matching Setting. We consider generalized entity matching, where the goal is to identify records referring to the same real-world entity across heterogeneous data sources. These sources exhibit heterogeneous structures and schemas.

To unify these heterogeneous inputs, we abstract each data source as a collection of entity records. Each record corresponds to an entity representation extracted from the source, such as a relational tuple, a JSON object, or a free-text passage. This canonical entity view serves as an intermediate abstraction for problem formulation.

Based on this abstraction, we adopt a two-table notation for formalization, where a "table" denotes a finite collection of entity records. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ denote two such collections extracted from two heterogeneous sources. The goal is to identify entity pairs (e_i, e'_j), where $e_i \in \mathcal{T}_1$ and $e'_j \in \mathcal{T}_2$, that refer to the same real-world entity. A typical entity matching pipeline [10] comprises two main steps: blocking and matching. The blocking step [20] reduces the quadratic search space from $n_1 \times n_2$ possible pairs by filtering out unlikely candidate pairs, thereby avoiding the computational cost of exhaustive pairwise comparisons. The matching step [15] then determines whether each remaining candidate pair is indeed a true match.

This generalized setting has been formalized in prior work such as Machamp [23]. It extends traditional entity matching beyond the scope of relational tables with identical schemas [10, 15, 20], to accommodate schema heterogeneity, enabling matching across structured, semi-structured, or unstructured tables.

Formally, let A be a set of d attributes representing the top-level keys in the schema. Each entity e in a table $\mathcal{T}$ is represented as a collection of key-value pairs, where each key belongs to A and each value is either a primitive type (e.g., a string or number) or a nested key-value structure [23]. Nested keys are not included in A and are handled recursively during downstream processing.

Based on the structure of values associated with keys in A, we categorize tables as follows:

- If all values are primitive and $d > 1$, then $\mathcal{T}$ is a structured (relational) table.
- If one or more values are nested, then $\mathcal{T}$ is a semi-structured table, such as in JSON-style data.
- If $d = 1$ and the single attribute contains unstructured free text, then $\mathcal{T}$ is an unstructured table.

While the *unsupervised* setting has been extensively studied in traditional entity matching [25, 27, 29], it remains underexplored in the context of generalized entity matching. We formally define the task as follows:

Unsupervised Generalized Entity Matching. Given two tables $\mathcal{T}_1$ and $\mathcal{T}_2$ with potentially different structures (structured, semi-structured, or unstructured) and schemas (homogeneous or heterogeneous), the goal is to identify all entity pairs (e_i, e'_j) such that $e_i \in \mathcal{T}_1$, $e'_j \in \mathcal{T}_2$, and e_i and e'_j refer to the same real-world entity without access to any labeled matching pairs.

Running Example. Fig. 1 illustrates the generalized entity matching task using the factual claim *"COVID-19 vaccines cause infertility"*, which serves as a real-world entity in this generalized setting. The same claim appears across heterogeneous representations, including a semi-structured JSON record, relational fact-checking tuples, and unstructured free-text records. In this example, we use

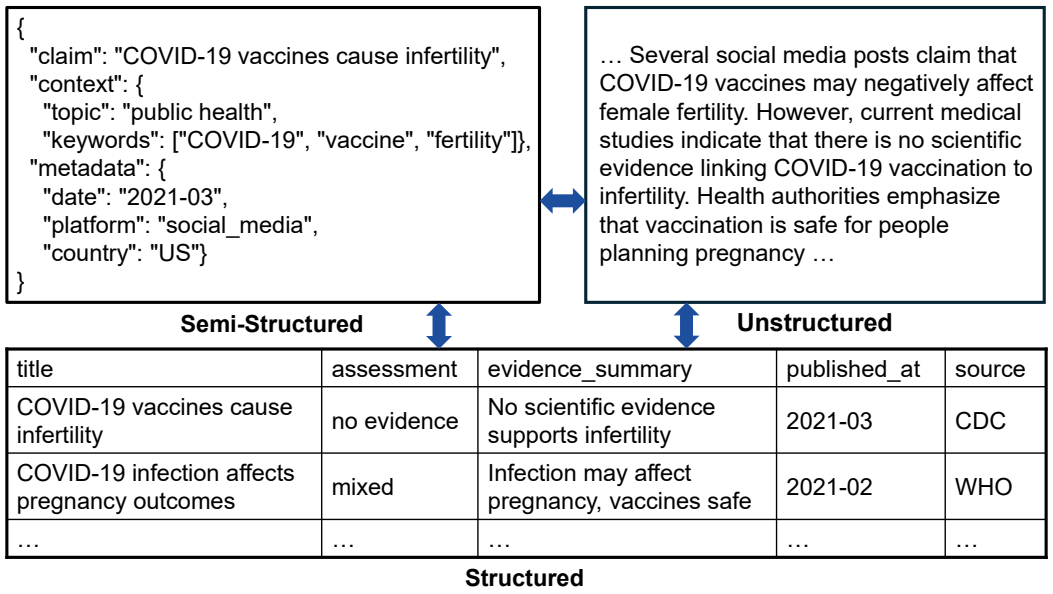


Fig. 1. An example of generalized entity matching.

the JSON record as one anchor entity for illustration, and the task is to identify all corresponding records across sources that refer to the same underlying claim. Despite referring to the same entity, these records differ substantially in structure and schema, making direct comparison non-trivial [2].

2.2 Existing Solutions

This subsection reviews existing solutions for generalized entity matching, with a particular focus on unsupervised approaches and their limitations.

Supervised Generalized Entity Matching. A wide range of entity matching methods originally developed for structured tables have been extended to the generalized setting [23]. These approaches typically featurize each entity using various models and then apply a binary classifier to predict whether a candidate pair represents a match or non-match. The models used for featurization and classification span a spectrum of techniques, including traditional machine learning algorithms such as support vector machines (SVMs) and random forests, deep neural networks such as bidirectional long short-term memory networks (BiLSTMs) [6] and recurrent neural networks (RNNs) [15].

Despite their success, these approaches exhibit several limitations when applied to generalized entity matching. Most rely on heuristic serialization techniques—such as linearizing entity fields or adding special delimiter tokens—to accommodate generalized inputs. Such methods fail to capture rich structural dependencies and contextual relationships inherent in complex or hierarchical data formats. Additionally, many methods assume schema homogeneity across tables and apply the same encoder architecture independently to both sides of a candidate pair. This design makes it difficult to reconcile semantic mismatches arising from heterogeneous schemas, particularly in the low-resource setting where labeled data is limited.

PromptEM [24] represents a recent supervised advance that reformulates the entity matching task as prompt-based masked language modeling. Each candidate pair is encoded into a prompt template, and a masked language model is trained to predict match-relevant tokens. While PromptEM shows

improved adaptability under schema variation, it remains inherently dependent on labeled training data. Collecting and annotating such data is expensive, especially when dealing with highly domain-specific or structurally diverse records that require expert knowledge to label. These challenges highlight the need for unsupervised alternatives that eliminate the reliance on manual supervision.

Unsupervised Entity Matching Solutions. The limitations of supervised approaches have prompted the development of several unsupervised solutions, each built upon distinct theoretical insights and methodological frameworks.

A plethora of work builds on the observation that match and non-match pairs exhibit different distributions in the feature space. These approaches vary in how they represent and separate these distributions. For instance, ZeroER [29] optionally applies blocking based on informative attributes (e.g., titles), generates feature vectors for candidate pairs using a set of handcrafted similarity functions, and models the resulting feature space with Gaussian Mixture Models (GMMs) to distinguish between matches and non-matches. Related ideas have also been explored in other settings, e.g., attribute recommendation [4]. Although effective for structured data, ZeroER's dependence on manually designed features and similarity functions limits its adaptability to heterogeneous or unstructured data. Sudowoodo [25] takes a self-supervised approach by employing contrastive learning to produce similarity-aware embeddings of entity records from both tables. Blocking is performed using approximate nearest neighbor search to retrieve top-k similar candidates, followed by automatic label generation: positive pairs are created through data augmentation, while negative pairs are sampled via clustering-based techniques. Despite its self-supervised nature, Sudowoodo's performance is sensitive to the quality of clustering and struggles in scenarios involving highly heterogeneous data where forming semantically meaningful clusters is nontrivial.

Another line of work formulates the matching problem as a graph-based neighborhood discovery task, often treating entities in one table as anchors and identifying corresponding matches in the other table through clustering or proximity. TDmatch [2], for example, constructs a heterogeneous graph where nodes represent either raw data values (e.g., strings, numbers) or metadata elements (e.g., structured attributes or unstructured entity identifiers). Edges connect metadata nodes to their respective data nodes, forming an undirected and unweighted graph. Random walks are conducted to capture contextual neighborhood information, and Word2Vec is used to derive embeddings for metadata nodes. Matching decisions are made by computing cosine similarity between embeddings and selecting the top-k most similar neighbors. While this method effectively incorporates structural signals, it suffers from scalability issues: random walks over large graphs are computationally expensive [22], and the unweighted graph design fails to capture the relative importance of connections, treating all relationships equally and leading to unstable performance.

More recently, ComEM [27] explores the use of large language models (LLMs) for unsupervised entity matching by systematically evaluating three prompting strategies: matching, comparing, and selecting. Among these, the selecting strategy—where an LLM is prompted to choose the correct match for a given record from a candidate set—stands out for its effectiveness and efficiency. While ComEM showcases the potential of LLMs in entity matching, it treats entity matching as a generic task and does not explicitly account for the unique challenges of generalized entity matching, such as schema heterogeneity, structural variation, and domain-specific contextual nuances. As a result, its applicability in more complex generalized scenarios remains limited, overlooking the rich structural and semantic signals that are often critical for accurate alignment across diverse data sources.

The limitations of existing unsupervised methods motivate the need for an end-to-end framework for generalized entity matching that directly maps input tables to matching results. Frequently used notations are summarized in Tab. 1.

Table 1. Frequently used notations.

Notation	Description
$\mathcal{T}_1$ / $\mathcal{T}_2$	Input anchor/candidate table for entity matching
n_1 / n_2	Number of entities in $\mathcal{T}_1$ / $\mathcal{T}_2$
G	Weighted heterogeneous graph used in the blocking module
U_1 / U_2	Entity nodes in G representing anchor / candidate entities
V	Content node set in G, representing semantic units
E_1 / E_2	Edges from U_1 / U_2 to V in G

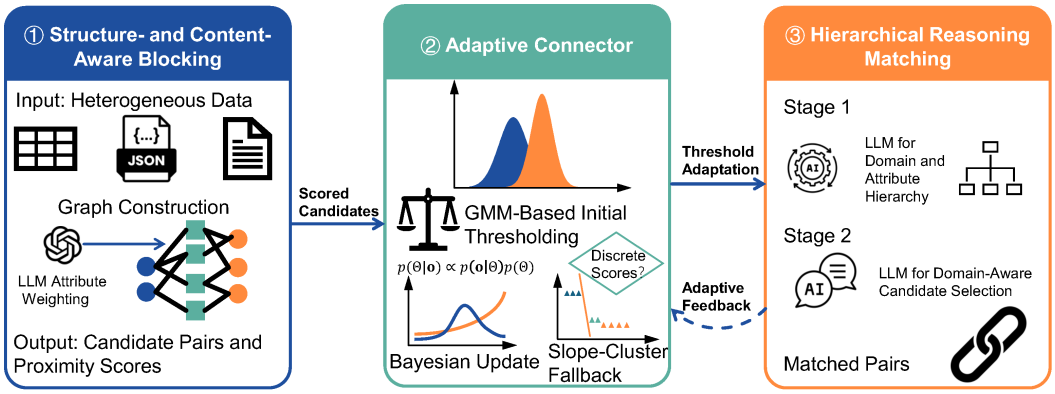


Fig. 2. Overview of the proposed adaptive framework for generalized entity matching.

3 Solution

3.1 Overview

We propose an end-to-end framework for unsupervised generalized entity matching that systematically addresses the fundamental challenges inherent in heterogeneous data structures, diverse schema configurations, and varying syntactic and semantic representations. The framework consists of three key modules:

- **Structure- and Content-Aware Blocking:** A graph-based blocking strategy with entity proximity that simultaneously leverages the natural structural dependencies inherent in data sources and content similarity measures, where the LLM is used to produce coarse attribute importance signals for edge weighting. It produces high-confidence candidate pairs and entity proximity scores that serve as informative signals for the matching.
- **Adaptive Connector:** A lightweight control module that analyzes entity proximity score distributions to adaptively guide matching, reducing unnecessary comparisons and computational overhead. It incorporates feedback from intermediate matching results to decide whether to continue or terminate, allowing the process to evolve from proximity-based prioritization to result-driven iterative refinement.
- **Hierarchical Reasoning Matching:** An LLM-powered hierarchical reasoning matching approach that first samples records to infer domain characteristics and attribute hierarchies, then performs contextually and semantically informed entity comparisons using domain-adaptive prompts weighted by component importance. Matching outcomes provide continuous feedback to the connector for dynamic process optimization.

Together, these modules form a unified and adaptive pipeline that transforms raw input tables into final match predictions in a fully unsupervised manner. Starting from raw tables with diverse schemas and heterogeneous structures, the structure- and content-aware blocking module generates candidate pairs with initial entity proximity scores. The adaptive connector uses these scores along with real-time feedback from the matching module to determine how extensively to proceed. Finally, the LLM-based hierarchical reasoning matching module automatically adapts to domain-specific characteristics and attribute hierarchies, producing final matched pairs. Fig. 2 provides an overview of the proposed framework. The following subsections describe each module and the complete pipeline in detail. Due to space limitations, we provide all detailed LLM prompts and examples in our technical report [1].

Role of LLMs. LLMs are used in two places in our pipeline: (i) in blocking, to infer coarse attribute importance signals that guide edge weighting; and (ii) in matching, as the main component for domain summarization and candidate selection. All remaining components, including graph construction, proximity computation, and connector updates, are implemented as non-LLM procedures.

We illustrate the behavior of each module using the running example introduced in Fig. 1, tracing the anchor entity through blocking, adaptive control, and final matching decisions.

3.2 Structure- and Content-Aware Blocking

We propose a structure- and content-aware blocking module that adopts a graph-based framework to prioritize promising entity pairs across tables. The core idea is to construct a weighted heterogeneous graph that captures both the structural relationships defined by the schema and the semantic similarities in the content. We apply multi-hop proximity to propagate match likelihood scores across the graph, effectively filtering out unlikely matches while retaining high-recall candidates.

Overview of Weighting. Rather than assigning weights directly to entity pairs, our blocking graph assigns weights to entity-content edges, in particular, where content nodes correspond to semantic units, including anchor-to-content and candidate-to-content edges, so that entity proximity emerges through shared content during information propagation.

To enhance robustness under schema heterogeneity, we incorporate lightweight LLM guidance in the form of coarse attribute importance signals inferred from a small sampled subset of records. These signals are used to set anchor-to-content edge weights during candidate generation.

Existing token-based blocking methods face challenges in such scenarios. For instance, Sparkly [17] uses token frequency scores based on trigram statistics, which work well when schema alignments are available. However, in the presence of diverse and unaligned schemas, such methods often degrade into string similarity and fail to reflect the relative importance of different attributes. Similarly, recent deep learning approaches [21] improve token representations through self-supervised learning, but they often ignore the structural context in which tokens appear. As a result, these approaches miss higher-level signals that help differentiate relevant from irrelevant candidates.

Our method takes a different approach by combining schema-based structural cues that reflect the origin and role of tokens with semantic similarity signals derived from the content itself. This unified design enables the graph to represent richer relationships and consider both local semantic features and global structural patterns when generating candidate matches.

Edge Weighting for Information Propagation. In this graph, edge weights modulate how semantic and structural signals propagate across shared content nodes. These weights determine the relative contribution of different content units when aggregating proximity scores between entities during candidate generation.

Asymmetric Roles of Anchor and Candidate Entities. We adopt asymmetric weighting schemes for anchor and candidate entities to reflect their different roles in blocking. Anchor-to-content weights emphasize how representative a content unit is for describing the anchor entity, while

candidate-to-content weights emphasize how discriminative the content is across different candidates. This asymmetry allows the graph to better prioritize informative shared content when generating candidate pairs.

Running Example (Blocking). For the running example in Fig. 1, the anchor entity is represented by the JSON record describing the claim *"COVID-19 vaccines cause infertility"*. Content nodes correspond to semantic units such as *COVID-19*, *COVID-19 vaccines*, and *infertility*, extracted from both the anchor and candidate records. Lightweight LLM guidance provides coarse attribute-importance signals that influence how anchor-to-content edge weights emphasize the representativeness of semantic units for the anchor, while candidate-to-content weights emphasize how discriminative the same units are across candidates. Through propagation over shared content nodes, the blocking graph assigns higher proximity scores to candidates that share informative semantic units with the anchor, yielding a ranked list of candidate entities.

To support efficient and accurate candidate generation, we generate a weighted heterogeneous graph $G = (U_1, U_2, V, E_1, E_2)$ for the blocking process. The graph consists of two types of nodes: (1) Entity nodes, where U_1 and U_2 represent anchor and candidate entities from tables $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively; and (2) Content nodes V, which encode *semantic units* at multiple levels of granularity. We define a semantic unit as a meaningful contiguous token sequence obtained via l-gram tokenization with $l \in \{1, 2, 3\}$ over primitive textual values. For example, "coffee," "coffee bean," and "coffee bean shop" are all semantic units of the textual value "coffee bean shop." Using multiple values of l captures phrases at different granularities, from finer to coarser, yielding increasingly specific semantic signals. We cap l at 3 to avoid overly long and noisy segments. Semantic units are extracted using a rule-based tokenization strategy, and each semantic unit is constructed as a content node in G.

Edges connect each entity to its associated content, with $E_1 \subset U_1 \times V$ and $E_2 \subset U_2 \times V$ denoting the edge sets from anchor and candidate entities to content nodes, respectively. These edges encode both structural dependencies and semantic associations, allowing the graph to propagate informative signals through shared content during candidate selection.

A key component of the graph is its edge weighting scheme, represented by weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ corresponding to edge sets E_1 and E_2. We adopt asymmetric weighting strategies to reflect the distinct roles of anchor and candidate entities. Anchor-to-content edges are weighted based on how well the content reflects the structure and semantics of the anchor entity. Candidate-to-content edges are weighted by the discriminative power or rarity of the content, enabling the graph to emphasize informative features that help distinguish true matches from non-matches.

We begin by describing the weighting strategy for anchor-to-content edges. To estimate attribute importance in a data-driven and adaptive manner, we leverage LLM-guided weighting. Specifically, we sample a small batch of anchor entities (e.g., 10) in $\mathcal{T}_1$, along with an equal number of candidate entities from $\mathcal{T}_2$, and prompt a large language model to generate a normalized attribute weight vector $\mathbf{w}_{\text{LLM}} \in \mathbb{R}^d_{\geq 0}$, where d is the number of top-level attributes in $\mathcal{T}_1$, and $\sum_{i=1}^{d} \mathbf{w}_{\text{LLM}}(i) = 1$. Each component $\mathbf{w}_{\text{LLM}}(i)$ reflects the relative discriminative utility of the i-th attribute for entity matching. In addition, the LLM outputs a distribution strategy vector $\mathbf{w}'_{\text{LLM}} \in \{0, 1\}^d$, where each bit $\mathbf{w}'_{\text{LLM}}(i)$ determines how the weight $\mathbf{w}_{\text{LLM}}(i)$ should be distributed:

- If $\mathbf{w}'_{\text{LLM}}(i) = 1$, the full attribute weight is assigned to each of its sub-elements (e.g., identifiers or atomic values).
- If $\mathbf{w}'_{\text{LLM}}(i) = 0$, the weight is instead divided proportionally across sub-elements (e.g., lists or key-value structures).

For each anchor entity $e \in \mathcal{T}_1$, we define a length vector $\boldsymbol{\ell}_e \in \mathbb{N}^d_{\geq 0}$, where $\boldsymbol{\ell}_e(i)$ denotes the length of the i-th attribute value (e.g., the number of semantic units for primitive types or sub-keys for

nested structures). Based on $\boldsymbol{\ell}_e(i)$, we construct a per-entity attribute weight vector $\mathbf{w}_e \in \mathbb{R}^d_{\geq 0}$:

$$\mathbf{w}_e(i) = \begin{cases} \mathbf{w}_{\text{LLM}}(i) \cdot \mathbb{1}_{\boldsymbol{\ell}_e(i)>0} & \text{if } \mathbf{w}'_{\text{LLM}}(i) = 0 \\ \mathbf{w}_{\text{LLM}}(i) \cdot \boldsymbol{\ell}_e(i) & \text{if } \mathbf{w}'_{\text{LLM}}(i) = 1 \end{cases}$$

We then re-normalize $\mathbf{w}_e$ to ensure that the total weight assigned across all attributes for entity u sums to 1: $\mathbf{w}_e(i) \leftarrow \frac{\mathbf{w}_e(i)}{\sum_{j=1}^{d} \mathbf{w}_e(j)}$. This step balances global attribute importance ($\mathbf{w}_{\text{LLM}}$) with entity-specific structural granularity ($\boldsymbol{\ell}_e$), yielding fine-grained adaptive weighting for anchor-to-content edges. It also standardizes weight scales across entities, thereby improving propagation stability. Importantly, this strategy generalizes from a small, representative sample, avoiding repeated LLM queries or deep attribute traversal, and keeping the blocking module lightweight.

The LLM-guided weighting mechanism addresses the challenge of attribute weighting under schema heterogeneity, traditionally a manual and costly task. For structured data, we distribute each attribute weight $\mathbf{w}_e(i)$ uniformly across its associated content nodes, proportionally to the attribute value length $\boldsymbol{\ell}_e(i)$. The edge weight from anchor entity e to content node c is defined as: $\mathbf{W}_1(e,c) = \sum_{i=1}^{d} (\mathbf{w}_e(i) \cdot f(c, \mathcal{T}_1(e,i))/\boldsymbol{\ell}_e(i))$, where $\mathcal{T}_1(e,i)$ is the i-th attribute value of e in $\mathcal{T}_1$, and $f(c, \mathcal{T}_1(e,i))$ denotes the frequency of content node c in that attribute.

For semi-structured data, where schema definitions can be partially nested, we recursively propagate weights to leaf-level attributes. Let d' be the number of such leaf attributes. After flattening, we compute: $\mathbf{W}_1(e,c) = \sum_{i=1}^{d'} [\mathbf{w}_e(i) \cdot f(c, \mathcal{T}_1(e,i))/(\boldsymbol{\ell}_e(i))]$, where $\mathbf{w}_e(i)$ is the recursively propagated weight for the i-th leaf, and $\boldsymbol{\ell}_e(i)$ is its content length.

For unstructured data such as free text, we treat the entire entity as a single attribute $\mathcal{T}_1(e,1)$ and assign uniform content weights: $\mathbf{W}_1(e,c) = f(c, \mathcal{T}_1(e,1))/\boldsymbol{\ell}_e(1)$. Throughout all cases, the frequency term $f(c,\cdot)$ captures the prominence of each content node, allowing the model to integrate semantic strength, structural cues, and statistical rarity during candidate generation.

To complement the anchor-to-content edge weights, we define the weighting strategy for candidate-to-content edges based on the discriminative power of each content node. Intuitively, content that appears frequently across candidate entities carries less distinguishing value, while rarer content is more informative for matching. We adapt the concept of inverse document frequency (IDF) from text retrieval [18] to this context. Let $d_{\text{cand}}(c)$ denote the number of candidate entities in $\mathcal{T}_2$ connected to content node c, and recall that n_2 is the total number of candidate entities. The edge weight from a candidate entity e' to content node c is defined as: $\mathbf{W}_2(e',c) = \log(1 + (n_2 - d_{\text{cand}}(c) + 0.5)/(d_{\text{cand}}(c) + 0.5))$, which mirrors the probabilistic IDF formulation used in the BM25 ranking function. This weighting strategy downweights common or generic content and upweights rare or distinctive signals, allowing the graph to emphasize features that are more likely to distinguish true matches. By integrating these asymmetric edge weights, our blocking graph captures both the structural semantics of anchor entities and the informativeness of candidate features, enhancing the recall in candidate generation.

Given the anchor-to-content edge weights $\mathbf{W}_1$ and candidate-to-content edge weights $\mathbf{W}_2$, we compute the entity proximity matrix $\mathbf{W} \in \mathbb{R}^{n_1 \times n_2}$ as $\mathbf{W} = \mathbf{W}_1 \cdot \mathbf{W}_2^\top$, where each entry $\mathbf{W}(e,e')$ represents the proximity between anchor entity e and candidate entity e'. Since both $\mathbf{W}_1$ and $\mathbf{W}_2$ contain nonzero values only for valid associations between entities and content nodes, the resulting proximity matrix $\mathbf{W}$ remains sparse. This sparsity allows for efficient computation. After computing the scores, we sort the nonzero proximity values for each anchor entity and return the ranked candidate pairs.

3.3 Adaptive Connector

For a given anchor entity, after deriving the proximity scores for all its candidates, we have the key observation that matched and unmatched candidates typically exhibit two distinct score distributions. Based on this observation, we propose an early-stop strategy that dynamically guides the matching process to avoid computational overheads. The main idea is to focus on the high-confidence candidates in the first distribution and ignore the unlikely candidates in the second. We initialize the matching procedure with a threshold to distinguish between the two distributions and begin with the candidates with the highest proximity scores.

During this procedure, we further leverage our matching module, which acts as an oracle to return binary match/non-match labels for queried pairs (details in the next subsection), to refine our threshold estimate using posterior updates. As oracle feedback accumulates, the connector progressively refines its estimate of the score distribution and adjusts the threshold accordingly. The intuition is that when early queries yield fewer matches, the connector should stop earlier; when promising matches appear, it should probe further to avoid missing true matches. This adaptive loop continues until sufficient evidence is collected, enabling entity-specific early stopping without compromising recall.

This strategy avoids the need to fix a global parameter such as top-k candidates per anchor, which fails to account for variations across different entities. It also circumvents the difficulty of setting a static threshold based only on noisy proximity scores, which can lead to suboptimal results. Prior work [17] shows that setting the threshold too high can eliminate true matches and reduce recall, while setting it too low can dramatically increase the number of false positives and inflate computational cost. In contrast, our adaptive framework supports entity-specific early stopping, improving overall efficiency without sacrificing matching quality.

Running Example (Connector). For the running example in Fig. 1, the blocking module produces a ranked list of candidate records. The adaptive connector queries candidates in descending score order and incorporates match/non-match feedback from the LLM-based matching module. For instance, when a high-scoring candidate discussing *COVID-19 infection effects* is judged as *unmatched*, the posterior estimate of the match ratio is updated accordingly. After such updates no longer change the estimated decision threshold, the connector terminates early, avoiding further unnecessary LLM calls.

Initial Threshold via Gaussian Mixture Model. The adaptive connector employs a Gaussian Mixture Model (GMM) to distinguish between the proximity score distributions of unmatched and matched entity pairs. These two distributions are modeled as Gaussian components: $\mathcal{N}(x \mid \mu_0, \sigma_0^2)$ and $\mathcal{N}(x \mid \mu_1, \sigma_1^2)$, where x denotes the proximity score and (μ_j, σ_j^2) are the mean and variance of the j-th component for $j \in \{0, 1\}$, corresponding to unmatched and matched pairs, respectively. The model is parameterized with mixing coefficients π_0 and π_1 such that $\pi_0 + \pi_1 = 1$. Given a sequence of the $|\mathbf{z}|$ descending proximity scores from the blocking model, denoted $\mathbf{z}$ where $z_i \in \mathbf{z}$ is the i-th largest proximity score, the GMM assumes the following mixture density: $p(x) = \pi_0 \mathcal{N}(x \mid \mu_0, \sigma_0^2) + \pi_1 \mathcal{N}(x \mid \mu_1, \sigma_1^2)$.

The goal of the connector is to estimate an initial threshold η_0 to differentiate between the two score distributions, defined as the point where the posterior probabilities of the two components are equal: $\pi_0 \mathcal{N}(x \mid \mu_0, \sigma_0^2) = \pi_1 \mathcal{N}(x \mid \mu_1, \sigma_1^2)$. To estimate the parameters $\Theta = \{\pi_j, \mu_j, \sigma_j^2\}_{j=0}^{1}$, we maximize the log-likelihood over the observed entity proximity scores: $\mathcal{L}(\mathbf{z}) = \sum_{i=1}^{|\mathbf{z}|} \log[\pi_0 \mathcal{N}(z_i \mid \mu_0, \sigma_0^2) + \pi_1 \mathcal{N}(z_i \mid \mu_1, \sigma_1^2)]$.

This is solved using the Expectation-Maximization algorithm. In E-Step, we compute the responsibility for each score z_i: $\gamma_{i,j} = \frac{\pi_j \mathcal{N}(z_i|\mu_j, \sigma_j^2)}{\pi_0 \mathcal{N}(z_i|\mu_0, \sigma_0^2) + \pi_1 \mathcal{N}(z_i|\mu_1, \sigma_1^2)}$, which represents the probability

that z_i belongs to component j. In M-Step, we update the parameters using the responsibilities: $\pi_j = \frac{1}{|\mathbf{z}|}\sum_{i=1}^{|\mathbf{z}|}\gamma_{i,j}$, $\mu_j = \frac{\sum_{i=1}^{|\mathbf{z}|}\gamma_{i,j}z_i}{\sum_{i=1}^{|\mathbf{z}|}\gamma_{i,j}}$ and $\sigma_j^2 = \frac{\sum_{i=1}^{|\mathbf{z}|}\gamma_{i,j}(z_i-\mu_j)^2}{\sum_{i=1}^{|\mathbf{z}|}\gamma_{i,j}}$. These steps are repeated until the change in the log-likelihood falls below a convergence tolerance.

Once the initial parameters $\hat{\Theta} = \{\hat{\pi}_j, \hat{\mu}_j, \hat{\sigma}_j^2\}_{j=0}^1$ are learned, the connector selects an initial decision threshold η_0 as the proximity score $z_i \in \mathbf{z}$ that minimizes the absolute difference between the two component densities:

$$\eta = \arg\min_{z_i \in \mathbf{z}} |\hat{\pi}_0 \mathcal{N}(z_i \mid \hat{\mu}_0, \hat{\sigma}_0^2) - \hat{\pi}_1 \mathcal{N}(z_i \mid \hat{\mu}_1, \hat{\sigma}_1^2)|.$$

This threshold provides an initial estimate of the boundary between likely matches and non-matches, and serves as the starting point for the connector's adaptive refinement during querying.

Bayesian Update of Mixing Coefficients. We then describe how the adaptive connector updates its proximity score distribution using observed oracle labels. When the matching module returns binary feedback for a subset of candidate pairs, this feedback provides new evidence for adjusting the GMM's parameters. Given initial parameters $\hat{\Theta}$ of the Gaussian Mixture Model (GMM), we aim to derive updated parameters Θ' conditioned on a set of oracle labels $\mathbf{o}$ where each element o_i indicates whether the corresponding score z_i is a match or not.

The update procedure is derived via Bayesian inference. In GMM, the component assignment for each score follows a Bernoulli distribution parameterized by π_1. The natural conjugate prior for a Bernoulli distribution is the Beta distribution. Thus, we place a prior over the mixing coefficient π_1, representing the probability that a score is from the matched component. Specifically, we have:

$$\pi_1 \sim \text{Beta}(\lambda_1, \lambda_0), \text{ and } \pi_0 = 1 - \pi_1,$$

where $\lambda_1, \lambda_0 > 0$ control the shape of the prior and represent pseudo-counts for matches and non-matches, respectively. The expectation of π_1 is given by $\mathbb{E}[\pi_1] = \frac{\lambda_1}{\lambda_0+\lambda_1}$.

Initially, the prior is constructed from all $|\mathbf{z}|$ unobserved scores in $\mathbf{z}$ from the blocking model. Let $|\mathbf{o}|$ be the number of observed labels so far. Then the remaining $|\mathbf{z}|-|\mathbf{o}|$ unobserved scores contribute to the prior pseudo-counts. We set $\lambda_1 = \hat{\pi}_1 \cdot (|\mathbf{z}| - |\mathbf{o}|)$, $\lambda_0 = \hat{\pi}_0 \cdot (|\mathbf{z}| - |\mathbf{o}|)$ with initial mixing coefficients, which reflects our decreasing reliance on the initial model as more true observations become available. The following lemma formalizes the posterior distribution after observing match outcomes. The proofs of all lemmas are deferred to our technical report [1] for brevity.

Lemma 1. *Given the prior $\pi_1 \sim Beta(\lambda_1, \lambda_0)$ and a set of $|\mathbf{o}|$ observations $\mathbf{o}$, where N_1 is the number of matches in $\mathbf{o}$ and $N_0 = |\mathbf{o}| - N_1$ is the number of non-matches, the posterior distribution of π_1 is: $\pi_1 \mid \mathbf{o} \sim Beta(\lambda_1 + N_1, \lambda_0 + N_0)$. The corresponding posterior expectation is $\mathbb{E}[\pi_1 \mid \mathbf{o}] = \frac{\lambda_1+N_1}{\lambda_1+\lambda_0+N_1+N_0}$ and $\mathbb{E}[\pi_0 \mid \mathbf{o}] = \frac{\lambda_0+N_0}{\lambda_1+\lambda_0+N_1+N_0}$.*

This posterior distribution reflects an updated confidence in the mixing coefficients for the matched component, allowing the connector to refine its estimate of the proximity score distribution using observed matches and non-matches.

Bayesian Update of Component Means and Variances. We describe how the connector updates the mean and variance parameters of each Gaussian component in the GMM after receiving feedback from the matching module. This update is performed in a Bayesian manner, leveraging the conjugacy between the Gaussian likelihood (for scores given a component) and the Normal–Inverse-Gamma prior over (μ_j, σ_j^2). The NIG prior is well-suited here because it jointly models uncertainty in the mean and variance while admitting closed-form posterior updates. For each component $j \in \{0, 1\}$ (unmatched/matched), the prior is specified as:

$$\mu_j \mid \sigma_j^2 \sim \mathcal{N}(\hat{\mu}_j, \sigma_j^2/\lambda_j),\ \sigma_j^2 \sim \text{InverseGamma}(a_j, b_j).$$

where $\hat{\mu}_j$ and $\hat{\sigma}_j^2$ are the initial estimates from the first Expectation-Maximization fitting, λ_j reflects the prior effective sample size, a_j plays the role of prior degrees of freedom, and b_j is the prior sum of squares. We set $a_j = \lambda_j$ and $b_j = (\lambda_j - 1)\hat{\sigma}_j^2$, which yields $\mathbb{E}[\sigma_j^2] = b_j/(a_j - 1) = \hat{\sigma}_j^2$ as desired.

Lemma 2. *Given the initial estimates $\mu_j = \hat{\mu}_j$ and $\sigma_j^2 = \hat{\sigma}_j^2$ for Gaussian component $j \in \{0, 1\}$ in the GMM, and a set $\mathbf{o}$ of $|\mathbf{o}|$ labeled observations with N_j assigned to component j, let the sample mean and variance of the observed scores in component j be:*

$$\bar{z}_j = \frac{1}{N_j} \sum_{i:o_i=j} z_i,\ s_j^2 = \frac{1}{N_j - 1} \sum_{i:o_i=j} (z_i - \bar{z}_j)^2,$$

and define the sum of squares $ss_j = (N_j - 1)s_j^2$. Then the posterior distribution of (μ_j, σ_j^2) is $NIG(\tilde{\mu}_j, \lambda'_j, a'_j, b'_j)$, with updated parameters:

$$\lambda'_j = \lambda_j + N_j,\ \tilde{\mu}_j = \frac{\lambda_j \hat{\mu}_j + N_j \bar{z}_j}{\lambda_j + N_j},\ a'_j = a_j + \frac{N_j}{2}$$

and $b'_j = b_j + \frac{ss_j}{2} + \frac{\lambda_j N_j (\bar{z}_j - \hat{\mu}_j)^2}{2(\lambda_j + N_j)}$. Consequently, the posterior expectations are $\mathbb{E}[\mu_j \mid \mathbf{o}] = \tilde{\mu}_j$ and $\mathbb{E}[\sigma_j^2 \mid \mathbf{o}] = b'_j/(a'_j - 1)$.

Iterative Refinement. After each feedback batch, the connector updates $\{\pi_j, \mu_j, \sigma_j^2\}_{j=0}^1$ via Lemmas 1 and 2, recomputes the threshold $\eta = \arg\min_{z_i \in \mathbf{z}} \left|\tilde{\pi}_0 \mathcal{N}(z_i \mid \tilde{\mu}_0, \tilde{\sigma}_0^2) - \tilde{\pi}_1 \mathcal{N}(z_i \mid \tilde{\mu}_1, \tilde{\sigma}_1^2)\right|$ and stops when η stabilizes. This process blends prior knowledge from initial scores with posterior updates from oracle feedback, yielding an adaptive and entity-specific matching policy. In practice, we observe that the Gaussian assumption may break down when similarity scores are highly discrete, requiring a separate treatment for such cases.

Handling Discrete Similarity Distributions. Although the adaptive connector relies on a GMM to separate the score distributions of matched and unmatched candidates, this assumption fails when the feature space is highly discrete. In such cases, the similarity scores take only a few distinct values, often due to limited token overlap, making the likelihood surface spiky rather than continuous. As a result, no smooth Gaussian can represent the data effectively, and the two components of the GMM may collapse into one [7].

To detect such cases, we define a distinct-value ratio, computed as the number of distinct similarity scores divided by the total number of candidate records for an anchor entity. A low ratio indicates that the scores are highly discrete and unsuitable for GMM-based modeling. When this ratio falls below a threshold τ (validated in our technical report [1]), the adaptive connector switches to a lightweight slope-cluster fallback strategy.

In this fallback mode, the connector identifies the maximum falling slope, that is, the largest score drop between adjacent candidates in the list $\mathbf{z}$ of descending proximity scores:

$$\eta = \arg \max_{z_i \in \mathbf{z} \setminus z_n} |z_{i+1} - z_i|.$$

Candidates with scores greater than or equal to η form the initial cluster for matching. The matching module then queries this cluster and receives feedback: if no candidate in the cluster is matched, the process stops; otherwise, the connector proceeds to the next cluster (the next score segment with the same proximity score) and repeats the procedure until no further matches appear.

This slope-based threshold approximates the point of maximal density change in the discrete similarity distribution, analogous to the intersection between Gaussian components in the continuous case. It provides a principled boundary between likely matches and non-matches when the GMM assumption is invalid. Overall, this alternative strategy complements the GMM-based connector,

Algorithm 1: Adaptive Connector for Anchor Entity e

Input: Proximity scores $\mathbf{z}$ for candidates of e
Output: Matched candidate set $\mathcal{M}_e$

Fit a GMM on $\mathbf{z}$ to estimate parameters $\hat{\Theta} = \{\hat{\pi}_j, \hat{\mu}_j, \hat{\sigma}_j^2\}_{j=0}^1$
Compute intersection threshold η_0 where the two component densities intersect
Initialize candidate set $C \leftarrow \{e' \mid z_{e'} \geq \eta_0\}$, match set $\mathcal{M}_e \leftarrow \emptyset$, and threshold bounds $\eta_{\text{high}} \leftarrow \eta_0$, $\eta_{\text{low}} \leftarrow \eta_0$
while $C \neq \emptyset$ **do**
    Query matching module: $\mathcal{M} \leftarrow \text{LLM}_s(e, C)$
    Update matched set: $\mathcal{M}_e \leftarrow \mathcal{M}_e \cup \mathcal{M}$
    Update cumulative observations $\mathbf{o}$ based on $\mathcal{M}$
    Update GMM $\Theta = \{\pi_j, \mu_j, \sigma_j^2\}_{j=0}^1$ via Bayesian posterior;
    Recompute new threshold η_{low} from updated GMM
    Update candidate set: $C \leftarrow \{e' \mid \eta_{\text{high}} \geq z_{e'} \geq \eta_{\text{low}}\}$
    Update upper bound: $\eta_{\text{high}} \leftarrow \eta_{\text{low}}$
if *distinct value ratio* $< \tau$ **then**
    Apply slope-cluster fallback strategy to estimate η
return matched candidate set $\mathcal{M}_e$

maintaining efficiency and accuracy even under discrete or sparse similarity distributions. The complete pipeline of the adaptive connector is detailed in Algorithm 1.

3.4 Hierarchical Reasoning Matching

The hierarchical reasoning matching module operates in two coordinated stages, leveraging large language models with specialized prompting roles to achieve precise entity matching through intelligent domain adaptation and attribute prioritization. The first stage is a triage LLM that infers the domain of the current matching task and assigns a hierarchical importance triage to attributes. The second stage is a domain expert LLM that, guided by this domain-specific knowledge and attribute hierarchy, performs fine-grained selection of true matches from candidate sets. We decompose the matching process into these two steps to enforce an explicit and consistent LLM reasoning workflow: we must first identify "what to look at", and then make the decision. With the triage LLM's inferred task domain and highlighted attributes, the second-stage matcher operates on a cleaner, more focused representation. For example, when matching products across two distinct online shopping platforms, the triage stage can infer an "e-commerce" domain, downweight platform-specific IDs and volatile price fields, and highlight category and description as high-importance attributes before the selection stage compares candidates.

Stage 1: Domain and Attribute Hierarchy Identification. We begin with small representative subsets $T_1 \subset \mathcal{T}_1$ and $T_2 \subset \mathcal{T}_2$ sampled from the anchor and candidate tables, respectively. Let A_1 and A_2 denote their attribute sets. For structured tables, these are the schema attributes; for semi-structured tables, they include all hierarchical keys extracted from records. From (T_1, T_2), the triage LLM produces: the inferred domain $\text{dom}(T_1, T_2)$ for the matching task, a high-importance attribute subset $A_h \subseteq A_1 \cup A_2$, and a low-importance attribute subset $A_\ell \subseteq A_1 \cup A_2$. Irrelevant attributes are excluded from both A_h and A_ℓ. Formally, the triage LLM is:

$$\text{LLM}_t : \{T_1, T_2\} \longrightarrow \{\text{dom}(T_1, T_2), A_h, A_\ell\}.$$

Table 2. Summary of datasets

Dataset	Left Table		Right Table		Labeled Pairs	
	#entity	#attr	#entity	#attr	#eval	%pos
Rel-HETER	534	6.00	332	7.00	946	11.63%
Semi-HOMO	2,616	8.65	64,263	7.34	17,223	18.63%
Semi-HETER	22,133	12.28	23,264	12.03	2,068	38.20%
Semi-REL	29,179	8.00	32,823	13.81	2,183	41.64%
Semi-TEXT-W	9,234	10.00	9,234	1.00	9,234	11.80%
Semi-TEXT-C	20,897	10.00	20,897	1.00	20,897	14.07%
Rel-TEXT	2,616	1.00	2,294	6.00	12,363	17.96%

Stage 2: Domain-Aware Candidate Selection. The adaptive connector (Section 3.3) produces, for each anchor entity e, a candidate set $C = \{e'_1, e'_2, \ldots, e'_n\}$, where n is the number of candidates. The second-stage LLM acts as a domain expert that leverages $(\mathrm{dom}(T_1, T_2), A_h, A_\ell)$ to decide which candidates are true matches. Formally, we define:

$$\mathrm{LLM}_s : \{e, C, \mathrm{dom}(T_1, T_2), \mathcal{A}_h, \mathcal{A}_\ell\} \longrightarrow \mathcal{M} \subseteq \{1, 2, \ldots, n\} \cup \emptyset,$$

where $\mathcal{M}$ is the index set of candidates selected as matches.

Rather than evaluating each candidate independently, LLM_s compares the entire batch C in one pass. This holistic evaluation allows it to exploit inter-candidate relationships, recognizing subtle similarities between true matches and distinguishing them from close non-matches. The output $\mathcal{M}$ of each iteration are combined as the set of confirmed matches for anchor e.

By explicitly modeling both domain knowledge and attribute importance hierarchy, this two-stage process enables: domain-adaptive feature prioritization, resilience against noisy attributes, and more accurate and efficient matching through LLM reasoning. Regarding implementation, the triage LLM and the domain expert LLM are instantiated via prompt engineering using the same base large language model, without any fine-tuning. The triage LLM generates a textual summary describing the inferred task domain and attribute-priority information, which is then directly embedded into the input prompt of the domain expert LLM. Thus, the two stages interact solely through explicit textual conditioning, without parameter sharing or model adaptation.

Running Example (Matching). For the running example in Fig. 1, the anchor claim *"COVID-19 vaccines cause infertility"* and its remaining candidates are processed by the LLM-based matching module. The triage stage identifies the public health domain and prioritizes attributes related to scientific evidence. The selection stage then jointly compares candidates and confirms those that explicitly refer to the same claim, while rejecting records discussing related but distinct topics (e.g., COVID-19 infection effects).

4 Experiments

We evaluate the proposed framework against representative baselines to assess its effectiveness, efficiency, and adaptability across diverse schema types. All experiments are conducted on a Linux server equipped with an Intel Xeon CPU @2.3GHz and 448GB RAM.

4.1 Experimental Setup

Datasets. We adopt the full Machamp benchmark [23], a real-world data collection covering a diverse range of table structures. The summary of these datasets is shown in Tab. 2. Dataset names indicate their structural type: structured (Rel), semi-structured (Semi), and unstructured (Text), as well as table heterogeneity: heterogeneous (HETER) or homogeneous (HOMO). In our unsupervised

Table 3. End-to-end effectiveness (S denotes supervised methods).

Methods	Rel-HETER			Semi-HOMO			Semi-HETER			Semi-REL			Semi-TEXT-W			Semi-TEXT-C			Rel-TEXT		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
ComEM	0.866	0.936	0.899	*0.947*	0.496	0.651	0.824	*0.975*	*0.893*	0.972	0.794	0.874	**0.733**	0.374	*0.495*	*0.957*	0.256	0.404	0.518	0.144	0.225
TDmatch	0.512	*0.991*	0.675	0.768	0.785	0.776	0.812	0.938	0.87	0.917	**0.951**	*0.934*	0.497	0.329	0.396	0.619	0.429	0.507	*0.731*	0.572	*0.642*
ZeroER	0.321	0.973	0.483	0.75	*0.903*	0.819	0.726	0.608	0.662	**1**	0.015	0.303	0.245	0.211	0.227	0.659	0.15	0.244	0.308	0.566	0.399
Sudowoodo	0.7	0.573	0.63	0.802	0.875	*0.837*	0.382	**1**	0.553	0.887	0.854	0.87	0.281	0.325	0.302	0.572	0.538	0.555	0.428	0.426	0.427
PromptEM (S)	**1**	**1**	**1**	0.934	**0.952**	**0.943**	**0.939**	0.579	0.716	0.925	*0.945*	**0.935**	0.449	*0.379*	0.411	0.697	**0.63**	*0.662*	0.592	*0.604*	0.598
GLEAM	*0.954*	0.936	*0.945*	**0.957**	0.742	0.836	*0.901*	0.943	**0.922**	*0.98*	0.821	0.893	*0.615*	**0.73**	**0.668**	**0.977**	*0.551*	**0.704**	**0.816**	**0.609**	**0.697**

end-to-end setting, ground-truth labels are used only for evaluation. While our experiments follow the clean-clean protocol adopted by Machamp, the proposed framework itself does not rely on this assumption and outputs matched entity pairs without requiring the absence of duplicates within either collection.

Baseline Methods. We compare our GLEAM framework against five representative entity matching methods. ComEM [27] leverages large language models (LLMs) for candidate pair matching. TDmatch [2] is an unsupervised approach that aligns textual and structured data through graph construction and random walks. ZeroER [29] generates feature vectors for candidate pairs and applies a Gaussian Mixture Model to determine a threshold separating matched and non-matched pairs. Sudowoodo [25] employs contrastive learning with entity transformation and negative sampling to distinguish entities. In addition, we include PromptEM [24], a state-of-the-art supervised method, using the same data splits as Machamp to ensure fair comparison. All baseline results are obtained by re-running the released implementations under a unified evaluation protocol. Hyperparameters follow the configurations reported in the corresponding baseline papers. Reported metrics are averaged over ten runs. All methods that use large language models, including our framework, employ gpt-4o as the underlying model with up to 10 concurrent workers. Unless otherwise noted, the distinct-value ratio threshold τ in the adaptive connector of GLEAM is fixed at 0.4, the best trade-off found in our technical report [1].

4.2 End-to-End Matching Effectiveness

In this subsection, we evaluate our framework against representative baselines on the end-to-end matching task, measured by precision (P), recall (R), and F1-score (F) between the returned entity pairs and the ground-truth correspondences for each dataset. In this paper, "end-to-end matching" refers to the full matching pipeline from raw datasets to returned entity pairs, comprising multiple stages such as blocking and final matching.

Tab. 3 reports the effectiveness across all datasets and methods, with the best results bolded and runner-up results italicized. At a glance, GLEAM leads in more than half of the benchmarks, specifically, on datasets Semi-HETER, Semi-TEXT-W, Semi-TEXT-C, and Rel-TEXT, demonstrating strong adaptability across both structured and text-heavy domains.

On dataset Rel-HETER, the supervised method PromptEM achieves perfect performance with 100% F1, while GLEAM ranks second, surpassing all other unsupervised alternatives by 4.6%. For Semi-HOMO, PromptEM again attains the highest score, with Sudowoodo leading the unsupervised group; our GLEAM follows closely, trailing by only 0.1%. On Semi-HETER, GLEAM attains the best overall F1, outperforming ComEM by 2.9%. For Semi-REL, PromptEM achieves the highest effectiveness, followed by TDmatch, while GLEAM ranks third but remains within a comparable range. On the Semi-TEXT-W benchmark, characterized by long text fields and weak schema alignment, GLEAM delivers the most substantial gain, an F1 improvement of 17.3% over the next best baseline ComEM. Similarly, on Semi-TEXT-C, GLEAM achieves the highest F1 among all methods, exceeding even the supervised PromptEM by 4.2%. On dataset Rel-TEXT, GLEAM once again leads with a margin of 5.5% over the second best baseline TDmatch.

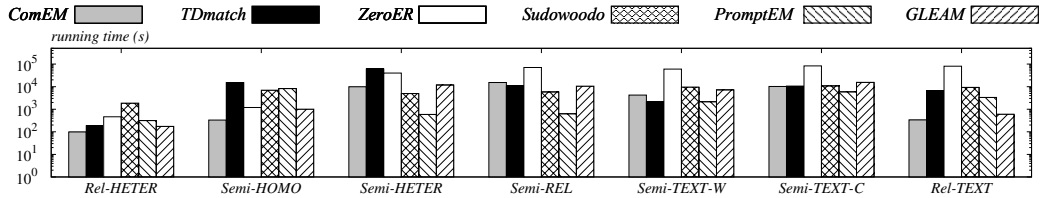


Fig. 3. End-to-end running time of various algorithms.

Table 4. End-to-end runtime statistics (mean, std, and 95% CI in seconds).

Methods	Rel-HETER			Semi-HOMO			Semi-HETER			Semi-REL			Semi-TEXT-W			Semi-TEXT-C			Rel-TEXT		
	Mean	Std	95% CI	Mean	Std	95% CI	Mean	Std	95% CI	Mean	Std	95% CI	Mean	Std	95% CI	Mean	Std	95% CI	Mean	Std	95% CI
ComEM	98.5	9.6	±6.9	332.3	19.5	±13.9	9854	389.2	±278.4	15287.5	313	±223.9	4237.5	140.6	±100.6	10203.7	372.6	±266.5	337.1	11.9	±8.5
TDmatch	186.4	3.7	±2.6	15152.5	424.6	±303.8	60569.1	2510.7	±1796	10676.3	290.4	±207.7	2100.6	116.4	±83.2	10639.8	299.3	±214.1	6669.5	242.4	±173.4
ZeroER	462.4	15.7	±11.2	1196.1	32.1	±23	39988.1	1238.7	±886.1	70784.9	2603	±1862.8	60056.2	1740.2	±1244.8	82775.5	2586.1	±1850	80379.7	2711.1	±1939.4
Sudowoodo	1863.6	27.2	±19.5	6983.9	121.9	±87.2	4946.6	104.4	±74.7	5832.1	150	±107.3	9590.6	221.2	±158.2	10806.1	243.2	±174	9284.5	464.5	±332.3
PromptEM	313.5	19.4	±13.9	8227.1	14	±10	589.9	5	±3.6	625.4	17.1	±12.2	2147.2	33	±23.6	5854.1	15.1	±10.8	3319.2	12.1	±8.6
GLEAM	173.3	15.8	±11.3	1009.9	70.5	±50.4	11784.5	520	±372	10480.9	589.4	±421.6	7199.6	276.9	±198.1	15485.8	404.5	±289.3	603.1	25.4	±18.2

From the perspective of precision and recall, different methods exhibit distinct trade-offs across datasets. Some baselines emphasize recall over precision (e.g., TDMatch and ZeroER on Rel-HETER), whereas GLEAM achieves a more balanced trade-off between precision and recall, leading to stable F1 performance across datasets.

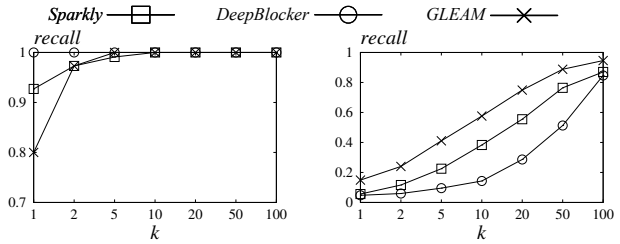
Overall, across diverse structural and textual configurations, GLEAM consistently ranks as the best unsupervised framework and remains competitive with supervised counterparts such as PromptEM, highlighting the strength of its adaptive connector and hierarchical reasoning modules.

4.3 Efficiency Analysis

We further evaluate the efficiency of the end-to-end generalized entity matching algorithms, with results summarized in Fig. 3. For all efficiency experiments, we perform 10 independent runs and report the mean runtime, standard deviation, and 95% confidence intervals computed using Student's t-distribution to ensure statistically sound comparisons across methods. We report runtime statistics to provide a fair and complete comparison across baselines. For clarity, Fig. 3 visualizes the mean runtime, while detailed numerical statistics are provided in Tab. 4. The runtime magnitude varies across datasets, reflecting differences in schema complexity and data modality. As shown, GLEAM achieves competitive performance relative to recent baselines, maintaining a consistent trade-off between computational cost and matching quality.

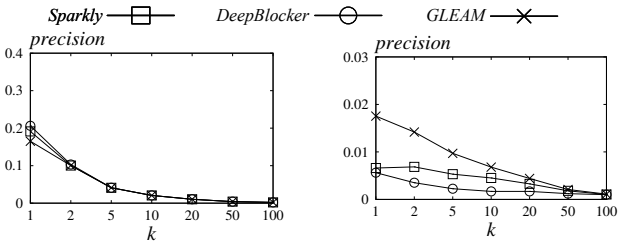
On the Rel-HETER benchmark, where most methods finish within a few hundred seconds, GLEAM remains comparable to ComEM and is notably faster than other pipelines. On datasets Semi-HOMO, Semi-HETER, Semi-REL, Semi-TEXT-W and Semi-TEXT-C, all models exhibit longer processing times due to semi-structured attributes and higher schema variability. In these settings, GLEAM remains competitive among unsupervised systems. On the Rel-TEXT dataset, dominated by free-text fields, GLEAM runs markedly faster than other methods, trailing only ComEM .

Overall, the measurements indicate that GLEAM achieves competitive efficiency while preserving its unsupervised and adaptive design. Across all datasets, the relative confidence intervals remain small and do not affect the comparative efficiency or scalability conclusions. The results further confirm that the Bayesian connector and hierarchical prompting introduce manageable computational overhead, keeping end-to-end runtime within the same order of magnitude as conventional EM baselines. This demonstrates that the adaptive architecture of GLEAM scales effectively without incurring excessive overhead, despite its unsupervised nature and LLM-based components.



(a) Rel-HETER (b) Semi-TEXT-W

Fig. 4. Varying k: Recall of various blocking algorithms.



(a) Rel-HETER (b) Semi-TEXT-W

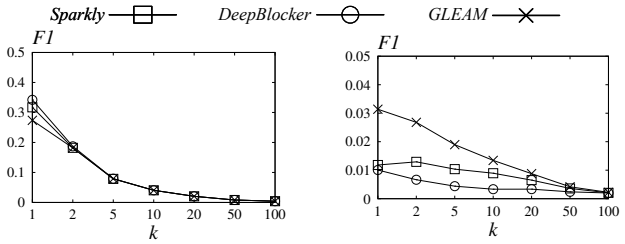
Fig. 5. Varying k: Precision of various blocking algorithms.

4.4 Blocking Module Evaluation

Effectiveness of Blocking. To assess the effectiveness of our structure- and content-aware blocking module, we compare GLEAM with two widely used blocking methods: Sparkly [17], a token-based approach, and DeepBlocker [20], a deep learning–based method. We follow the parameter settings reported in their original papers for a fair comparison. For each anchor entity, we vary the number of candidate entities k from 1 to 100, and report recall as well as precision and F1-score of true matches at $k \in \{1, 2, 5, 10, 20, 50, 100\}$. We focus on Rel-HETER and Semi-TEXT-W as representative heterogeneous settings; results on the remaining datasets are deferred to our technical report [1].

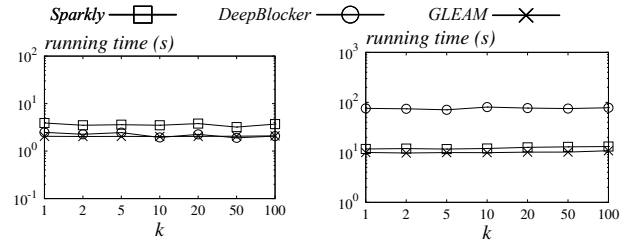
Fig. 4 summarizes the recall results across datasets. Overall, GLEAM outperforms both Sparkly and DeepBlocker in most settings, particularly on heterogeneous and text-rich datasets, demonstrating its ability to capture both structural and semantic signals during the blocking procedure. On Rel-HETER, DeepBlocker attains the highest recall across all k, but GLEAM quickly narrows the gap, within 2.7% at $k = 2$, and remains competitive as k increases. On Semi-TEXT-W, GLEAM remains superior throughout, achieving margins of up to 19% over Sparkly at $k = 10$ and 46% over DeepBlocker at $k = 20$.

Figs. 5 and 6 report blocking precision and F1-score, respectively, as k increases on Rel-HETER and Semi-TEXT-W. Due to space constraints, results on the remaining datasets are provided in our technical report [1]. As expected, blocking precision and F1-score decrease rapidly with larger k, since more candidates are intentionally retained to avoid false negatives. Moreover, the absolute precision and F1-score values vary across datasets, largely due to differences in the density of true matching pairs relative to the overall candidate space. This behavior is inherent to recall-oriented candidate generation and reflects the trade-off between coverage and noise. Importantly, this trade-off does not affect the end-to-end matching quality, as the downstream matching stage explicitly filters these candidates and restores precision.



(a) Rel-HETER (b) Semi-TEXT-W

Fig. 6. Varying k: F1 of various blocking algorithms.



(a) Rel-HETER (b) Semi-TEXT-W

Fig. 7. Varying k: Running time of various blocking algorithms.

These results validate the effectiveness and robustness of our adaptive blocking strategy, especially on heterogeneous and text-centric datasets where traditional token-based or neural blocking methods struggle to preserve both coverage and efficiency.

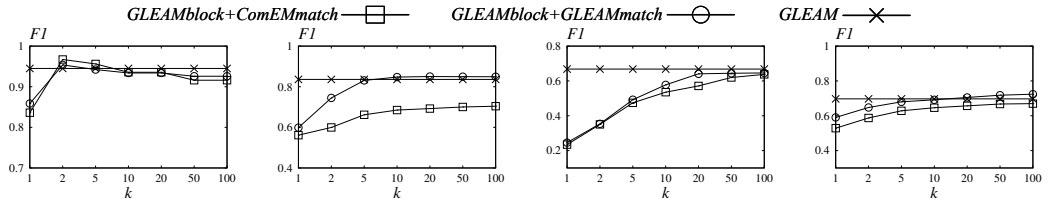
Efficiency of Blocking. We next evaluate the efficiency of the blocking module in terms of runtime, as shown in Fig. 7. Overall, GLEAM exhibits substantially lower runtime than DeepBlocker and comparable runtime to Sparkly, except on the relatively small dataset Rel-HETER. On large dataset, GLEAM is slightly slower than Sparkly but remains consistently more efficient than DeepBlocker. Results on other datasets are deferred to our technical report [1].

These results demonstrate that our structure- and content-aware blocking module attains high effectiveness while maintaining competitive efficiency, validating that the design improvements do not come at the cost of excessive computational overhead.

4.5 Adaptive Matching and Connector Analysis

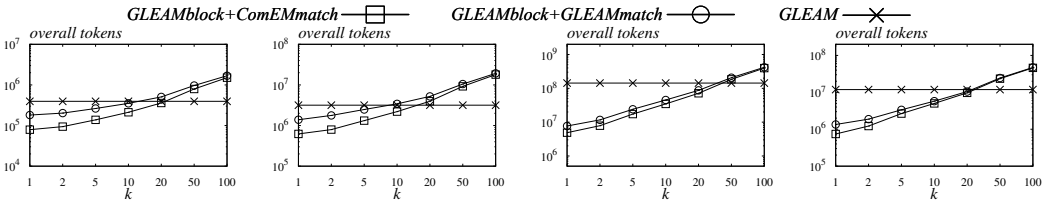
Effectiveness of Adaptive Matching. To evaluate the effectiveness of our adaptive matching module, we conduct an ablation study that isolates the impact of the matching component while keeping the same blocking module across variants. Specifically, we compare three configurations: (i) GLEAMblock+ComEMmatch, which uses our blocking module combined with the light prompt–based LLM matcher of ComEM [27]; (ii) GLEAMblock+GLEAMmatch, which replaces ComEM's matcher with our own matching module but disables the adaptive connector; and (iii) GLEAM, the full model that integrates both our matching module and the adaptive connector. For brevity, we omit the "GLEAMblock" prefix hereafter. For each anchor entity, we vary the number of candidate entities k from 1 to 100 and report the F1-score between the predicted and ground-truth entity pairs at $k \in \{1, 2, 5, 10, 20, 50, 100\}$. We report results on Rel-HETER, Semi-HOMO, Semi-TEXT-W, and Rel-TEXT, and defer results on the remaining datasets to our technical report [1].

Fig. 8 presents the results. Overall, GLEAMmatch consistently outperforms ComEMmatch, while the full GLEAM model achieves comparable or superior F1 across datasets by adaptively determining



(a) Rel-HETER (b) Semi-HOMO (c) Semi-TEXT-W (d) Rel-TEXT

Fig. 8. Varying k: F1 of different LLM matching policies.



(a) Rel-HETER (b) Semi-HOMO (c) Semi-TEXT-W (d) Rel-TEXT

Fig. 9. Varying k: Overall tokens of different LLM policies.

when to stop deepening candidate exploration. This adaptivity avoids blindly fixing the candidate number k: smaller k risks missing true matches, whereas larger k introduces noisy candidates and unnecessary computation.

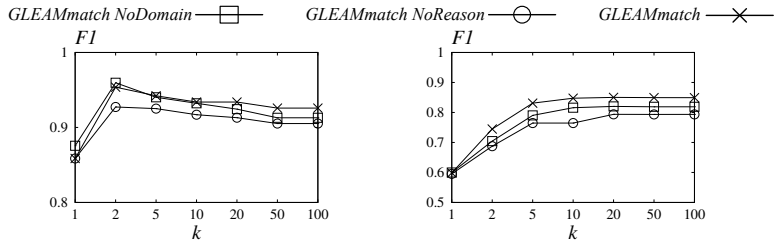
On Rel-HETER, GLEAMmatch achieves a higher F1 at $k = 1$ with a 2.2% lead, while ComEMmatch slightly outperforms it for $k \in [2, 20]$. As k increases, GLEAMmatch again surpasses ComEMmatch. Compared to both GLEAMmatch and ComEMmatch, the full GLEAM model gains up to 8.6% and 10.9%, respectively, at $k = 1$, and still maintains margins of 2% and 2.9% at $k = 100$. On Semi-HOMO, GLEAMmatch consistently exceeds ComEMmatch by up to 16.9% at $k = 5$. The full GLEAM model further improves performance for $k \leq 5$ and trails slightly as k grows. On Semi-TEXT-W, GLEAMmatch again outperforms ComEMmatch with up to 6.9% improvement at $k = 20$. The adaptive connector further boosts effectiveness, achieving a substantial 42.2% gain at $k = 1$. On Rel-TEXT, GLEAMmatch improves upon ComEMmatch by up to 6.2% at $k = 1$. The full GLEAM model remains superior for $k < 20$ and maintains competitive performance at larger k.

These results confirm that the adaptive connector effectively balances exploration depth and matching confidence, while the hierarchical reasoning component ensures robust and schema agnostic effectiveness across domains.

Efficiency and Token Usage. We next evaluate LLM efficiency by measuring total token consumption, defined as: Overall Tokens = Input Tokens + Output Tokens $\times$ 4, following the gpt-4o pricing ratio of 1:4 between input and output.[1] We vary candidate sizes per anchor record $k \in [1, 100]$ and report overall tokens when k is $\{1, 2, 5, 10, 20, 50, 100\}$. We report results on Rel-HETER, Semi-HOMO, Semi-TEXT-W, and Rel-TEXT as shown in Fig. 9, while the remaining datasets are provided in our technical report [1].

Across the reported datasets, GLEAM consistently maintains a flat token cost regardless of the number of candidates, while ComEMmatch and GLEAMmatch exhibit a steep growth trend as the number of candidates increases. For example, on Semi-TEXT-W, GLEAM holds constant at 143M tokens, while ComEMmatch grows from 4.9M to 396M and GLEAMmatch from 7.7M to 419M, saving 3-4$\times$ tokens with even higher F1. On Rel-TEXT, GLEAM requires only 12M, while

[1]OpenAI API Pricing: https://platform.openai.com/docs/pricing



(a) Rel-HETER (b) Semi-HOMO

Fig. 10. Ablation of the matching module: F1 vs. k.

Table 5. End-to-end matching performance with different LLM backbones (precision, recall, and F1).

Models	Rel-HETER			Semi-HOMO			Semi-HETER			Semi-REL			Semi-TEXT-W			Semi-TEXT-C			Rel-TEXT		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
GPT-4o-mini	0.911	0.836	0.872	0.96	0.613	0.748	0.835	**0.975**	0.90	0.952	**0.824**	0.889	0.594	**0.733**	0.656	0.86	0.535	0.659	*0.85*	0.417	0.56
GPT-4o	*0.954*	0.936	*0.945*	*0.957*	**0.742**	**0.836**	*0.901*	0.943	*0.922*	*0.98*	*0.821*	**0.893**	0.615	*0.73*	**0.668**	**0.977**	*0.551*	**0.704**	0.816	**0.609**	**0.697**
Llama-4-17B	0.823	**0.973**	0.892	0.953	0.573	0.716	0.838	*0.963*	0.896	0.965	0.573	0.719	0.447	0.235	0.308	*0.958*	0.155	0.267	0.735	0.251	0.374
Llama-3.3-70B	0.92	0.936	0.928	0.951	*0.631*	*0.759*	0.844	*0.963*	0.90	0.973	0.589	0.733	**0.899**	0.41	0.563	0.939	0.48	0.636	0.768	0.349	0.48
Qwen3-32B	0.907	**0.973**	0.939	**0.972**	0.584	0.73	**0.919**	0.933	**0.926**	**0.982**	0.71	0.824	0.667	0.37	0.476	0.947	0.534	0.683	**0.871**	0.424	0.57
Qwen3-235B	**0.99**	0.936	**0.963**	0.947	0.602	0.736	0.871	0.962	0.914	0.975	*0.821*	*0.891*	*0.833*	0.549	*0.662*	0.925	**0.563**	*0.70*	0.801	*0.587*	*0.677*

ComEMmatch and GLEAMmatch rise from under 1M to over 47M, reducing token consumption by more than 75%.

These results demonstrate that the adaptive connector effectively regulates LLM usage by dynamically controlling exploration depth and terminating matching once sufficient confidence is reached. This design yields large efficiency gains without compromising accuracy, confirming that GLEAM achieves a strong balance between effectiveness and computational cost.

4.6 Hierarchical Reasoning Matching Analysis

To validate the effectiveness of the proposed two-stage hierarchical reasoning in the matching module, we conduct an ablation study with three variants: (i) the full module with domain-aware summarization and the complete selection prompt (GLEAMmatch), (ii) a variant that omits domain summarization and performs selection directly from raw attributes (GLEAMmatch NoDomain), and (iii) a variant that retains domain summarization but uses a simplified selection prompt without explicit reasoning guidance (GLEAMmatch NoReason). We vary the candidate set size $k \in \{1, 2, 5, 10, 20, 50, 100\}$ per anchor record and report F1-score.

Fig. 10 shows consistent trends on Rel-HETER and Semi-HOMO (results on the remaining datasets are provided in our technical report [1] due to space constraints). Across all values of k, GLEAMmatch consistently achieves the highest F1-score, while GLEAMmatch NoReason performs worst. On Rel-HETER, removing explicit reasoning guidance in the selection prompt reduces F1 by 2.7% at the peak $k = 2$. The effect is larger on Semi-HOMO: at $k = 10$, the full model reaches 0.845 F1, compared to 0.816 for GLEAMmatch NoDomain and 0.765 for GLEAMmatch NoReason.

These results support our design of an explicit two-stage reasoning pipeline: domain summarization provides a consistent performance lift, while a richer selection prompt improves matching quality. Together, they confirm that separating schema understanding from final selection yields more robust matching behavior.

4.7 Backbone Model Analysis

To examine whether the effectiveness of GLEAM depends on a specific LLM, we evaluate the end-to-end matching performance using multiple backbone models, including GPT-4o-mini, GPT-4o, Llama-4-17B, Llama-3.3-70B, Qwen3-32B, and Qwen3-235B, covering both proprietary and open-source LLMs with varying capacities.

Overall, Tab. 5 shows consistent trends across both proprietary and open-source backbones. Larger models (e.g., GPT-4o and Qwen3-235B) generally achieve a stronger balance between precision and recall, while mid-sized models exhibit similar but slightly less stable behavior. Smaller models tend to struggle more with group-level candidate comparison, particularly on text-heavy datasets such as Semi-TEXT-W and Rel-TEXT, where records contain longer textual attributes.

For example, on these datasets, smaller models such as Llama-4-17B show substantially lower F1 compared with larger models, suggesting greater difficulty in identifying true matches. At the same time, open-source models such as Qwen3-32B remain competitive on structured and semi-structured datasets.

These results suggest that GLEAM generalizes well across different LLM families and model sizes, and that its effectiveness is not tied to any specific backbone.

5 Related Work

Our work is related to several lines of research in entity matching and data management, including traditional and schema-agnostic entity matching, graph-based entity matching, end-to-end entity matching pipelines, and the emerging use of large language models (LLMs) in data management.

Traditional and Schema-Agnostic Entity Matching. Entity matching has been extensively studied in the data management community. Early approaches primarily relied on rule-based systems or supervised learning models with manually engineered features and schema-specific similarity functions [5, 10, 26, 28]. To reduce dependence on explicit schema alignment, a series of schema-agnostic entity matching methods have been proposed, leveraging pre-trained language models to capture semantic similarity across attributes or records [12, 15, 19]. These methods substantially improve robustness to schema variation. However, they primarily focus on learning schema-agnostic similarity functions and typically consider records with relatively similar structural representations, without explicitly modeling cross-structure or hierarchical heterogeneity.

Graph-Based Entity Matching. A prominent line of work formulates entity matching as a graph-based learning problem, where entities, attribute values, or tokens are modeled as nodes and edges encode similarity or contextual relationships [2, 9, 11]. Representative approaches such as GraphER and FlexER construct heterogeneous or multiplex graphs and apply graph neural networks for collective inference across records. GraphER employs graph convolutional networks (GCNs) over entity-attribute-token hierarchies to capture semantic and structural dependencies [11]. FlexER constructs a candidate-pair graph and performs message passing using GraphSAGE to distinguish matched and unmatched pairs [9]. While effective, many existing graph-based approaches rely on labeled record pairs to train matching models or guide representation learning. TDmatch [2] adopts an unsupervised graph-based formulation that avoids labeled data, but still faces limitations in providing matching guidance and scaling to large datasets.

End-to-End Entity Matching Pipelines. Beyond individual matching models, entity matching systems are commonly organized as end-to-end pipelines consisting of blocking and matching stages [10, 27]. Recent work emphasizes jointly optimizing these stages to reduce error propagation and improve overall effectiveness. Representative approaches explore supervised contrastive learning for blocking and candidate selection [3, 16], while other methods adopt probabilistic or representation-based scoring mechanisms to separate matched and non-matched pairs within an

end-to-end setting [25, 29]. Our evaluation in Section 4.2 follows this end-to-end paradigm and compares against representative baselines to assess overall matching effectiveness.

LLMs in Data Management. Large language models are rapidly emerging as general-purpose reasoning engines for data-centric tasks, spanning schema understanding, query generation (e.g., text-to-SQL), data discovery, and knowledge extraction [8, 13, 30]. Our framework aligns with this evolution by embedding LLM-guided reasoning within an adaptive, unsupervised pipeline, leveraging language understanding for attribute weighting and comparative selection without relying on extensive prompting or fine-tuning.

6 Conclusion

We present an end-to-end framework for generalized entity matching powered by large language models. The proposed approach integrates structure- and content-aware blocking, hierarchical reasoning for matching, and an adaptive connector that reduces computational overhead. Extensive experiments show that GLEAM achieves up to 25.7% F1 improvement over the state-of-the-art supervised method while maintaining consistently high efficiency. As future work, we plan to relax the static data assumption by extending GLEAM to support temporal and dynamic settings with insertions, updates, and deletions, enabling continuous entity matching over evolving datasets. In addition, although GLEAM already scales well on moderately large benchmarks, we aim to further explore distributed and streaming extensions to support enterprise-scale deployments.

Acknowledgments

This research was supported by the Ministry of Education, Singapore, under an MOE AcRF TIER 3 Grant (MOE-MOET32022-0001).

References

[1] 2026. Code and Technical Report. https://github.com/Blondig/GLEAM.

[2] Naser Ahmadi, Hansjorg Sand, and Paolo Papotti. 2022. Unsupervised Matching of Data and Text. In *IEEE International Conference on Data Engineering, ICDE*. 1058–1070. doi:10.1109/ICDE53745.2022.00084

[3] Alexander Brinkmann, Roee Shraga, and Christian Bizer. 2024. SC-Block: Supervised Contrastive Blocking Within Entity Resolution Pipelines. In *European Semantic Web Conference, ESWC (Lecture Notes in Computer Science, Vol. 14664)*. 121–142. doi:10.1007/978-3-031-60626-7_7

[4] Xingguang Chen, Fangyuan Zhang, Jinchao Huang, and Sibo Wang. 2023. Efficient Approximation Framework for Attribute Recommendation. *Proc. ACM Manag. Data* 1, 4 (2023), 239:1–239:26. doi:10.1145/3626726

[5] Peter Christen. 2008. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*. 1065–1068. doi:10.1145/1401890.1402020

[6] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *Proc. VLDB Endow.* 11, 11 (2018), 1454–1467. doi:10.14778/3236187.3236198

[7] Robin Fuchs, Denys Pommeret, and Cinzia Viroli. 2022. Mixed Deep Gaussian Mixture Model: a clustering model for mixed datasets. *Adv. Data Anal. Classif.* 16, 1 (2022), 31–53. doi:10.1007/S11634-021-00466-3

[8] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (2024), 1132–1145. doi:10.14778/3641204.3641221

[9] Bar Genossar, Roee Shraga, and Avigdor Gal. 2023. FlexER: Flexible Entity Resolution for Multiple Intents. *Proc. ACM Manag. Data* 1, 1 (2023), 42:1–42:27. doi:10.1145/3588722

[10] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *Proc. VLDB Endow.* 9, 12 (2016), 1197–1208. doi:10.14778/2994509.2994535

[11] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*. 8172–8179. doi:10.1609/AAAI.V34I05.6330

[12] 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. doi:10.14778/3421424.3421431

[13] Zixuan Li, Yutao Zeng, Yuxin Zuo, Weicheng Ren, Wenxuan Liu, Miao Su, Yucan Guo, Yantao Liu, Xiang Li, Zhilei Hu, Long Bai, Wei Li, Yidan Liu, Pan Yang, Xiaolong Jin, Jiafeng Guo, and Xueqi Cheng. 2024. KnowCoder: Coding Structured Knowledge into LLMs for Universal Information Extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*. 8758–8779. doi:10.18653/V1/2024.ACL-LONG.475

[14] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. Rotom: A Meta-Learned Data Augmentation Framework for Entity Matching, Data Cleaning, Text Classification, and Beyond. In *Proceedings of the International Conference on Management of Data, SIGMOD*. 1303–1316. doi:10.1145/3448016.3457258

[15] 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. In *Proceedings of the International Conference on Management of Data, SIGMOD*. 19–34. doi:10.1145/3183713.3196926

[16] Arthur Ning, Flavius Frasincar, and Tarmo Robal. 2025. SC-Block++: A Blocking Algorithm Based on Adaptive Flood Regularization. In *Proceedings of the ACM/SIGAPP Symposium on Applied Computing, SAC*. 1258–1266. doi:10.1145/3672608.3707946

[17] Derek Paulsen, Yash Govind, and AnHai Doan. 2023. Sparkly: A Simple yet Surprisingly Strong TF/IDF Blocker for Entity Matching. *Proc. VLDB Endow.* 16, 6 (2023), 1507–1519. doi:10.14778/3583140.3583163

[18] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009), 333–389. doi:10.1561/1500000019

[19] Kai Sheng Teong, Lay-Ki Soon, and Tin Tin Su. 2020. Schema-Agnostic Entity Matching using Pre-trained Language Models. In *ACM International Conference on Information and Knowledge Management, CIKM*. 2241–2244. doi:10.1145/3340531.3412131

[20] 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. doi:10.14778/3476249.3476294

[21] 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. doi:10.14778/3476249.3476294

[22] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *Proceedings of the IEEE International Conference on Data Mining, ICDM*. 613–622. doi:10.1109/ICDM.2006.70

[23] Jin Wang, Yuliang Li, and Wataru Hirota. 2021. Machamp: A Generalized Entity Matching Benchmark. In *ACM International Conference on Information and Knowledge Management, CIKM*. 4633–4642. doi:10.1145/3459637.3482008

[24] Pengfei Wang, Xiaocan Zeng, Lu Chen, Fan Ye, Yuren Mao, Junhao Zhu, and Yunjun Gao. 2022. PromptEM: Prompt-tuning for Low-resource Generalized Entity Matching. *Proc. VLDB Endow.* 16, 2 (2022), 369–378. doi:10.14778/3565816.3565836

[25] Runhui Wang, Yuliang Li, and Jin Wang. 2023. Sudowoodo: Contrastive Self-supervised Learning for Multi-purpose Data Integration and Preparation. In *IEEE International Conference on Data Engineering, ICDE*. 1502–1515. doi:10.1109/ICDE55515.2023.00391

[26] Sibo Wang, Xiaokui Xiao, and Chun-Hee Lee. 2015. Crowd-Based Deduplication: An Adaptive Approach. In *Proceedings of the International Conference on Management of Data, SIGMOD*. 1263–1277. doi:10.1145/2723372.2723739

[27] 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 *Proceedings of the International Conference on Computational Linguistics, COLING*. 96–109.

[28] Steven Whang and Hector Garcia-Molina. 2010. Entity Resolution with Evolving Rules. *Proc. VLDB Endow.* 3, 1 (2010), 1326–1337. doi:10.14778/1920841.1921004

[29] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *Proceedings of the International Conference on Management of Data, SIGMOD*. 1149–1164. doi:10.1145/3318464.3389743

[30] Mengyi Yan, Yaoshu Wang, Kehan Pang, Min Xie, and Jianxin Li. 2024. Efficient Mixture of Experts based on Large Language Models for Low-Resource Data Preprocessing. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*. 3690–3701. doi:10.1145/3637528.3671873

Received October 2025; revised January 2026; accepted February 2026