EA-Agent: A Structured Multi-Step Reasoning Agent for Entity Alignment

Anonymous ACL submission

Abstract

Entity alignment (EA) aims to identify entities across different knowledge graphs (KGs) that refer to the same real-world object and plays a critical role in knowledge fusion and integration. Traditional EA methods mainly rely on knowledge representation learning, but their performance is often limited under noisy or sparsely supervised scenarios. Recently, large language models (LLMs) have been introduced to EA and achieved notable improvements by leveraging rich semantic knowledge. However, existing LLM-based EA approaches typically treat LLMs as black-box decision makers, resulting in limited interpretability, and the direct use of large-scale triples substantially increases inference cost. To address these challenges, we propose **EA-Agent**, a reasoning-driven agent for EA. EA-Agent formulates EA as a structured reasoning process with multi-step planning and execution, enabling interpretable alignment decisions. Within this process, it introduces attribute and relation triple selectors to filter redundant triples before feeding them into the LLM, effectively addressing efficiency challenges. Experimental results on three benchmark datasets demonstrate that EA-Agent consistently outperforms existing EA methods and achieves state-of-the-art performance. The source code is available at https://anonymous.4open.science/r/EA-Agent-5696.

1 Introduction

Knowledge graphs (KGs) represent real-world knowledge in a structured form and have been widely applied to various knowledge-driven tasks, including question answering (Saxena et al., 2022; Wang et al., 2024b; Xu et al., 2024), personalized recommendation (Jiang et al., 2024b; Cui et al., 2024; Wang et al., 2025), and knowledge reasoning (Wang et al., 2024a; Tan et al., 2025; Jiang et al., 2025). However, due to differences in data sources and construction methodologies, KGs are inherently heterogeneous.

To enable knowledge fusion across different KGs, entity alignment (EA) plays a crucial role (Zhang et al., 2022). EA aims to identify equivalent entities in different KGs that refer to the same real-world object, constituting one of the most fundamental techniques for knowledge fusion and a key step toward integrating heterogeneous information (Zeng et al., 2021; Zhu et al., 2024). Previous EA methods are mainly based on knowledge representation learning, where entities are aligned by learning embeddings from different KGs and measuring their similarities (Bordes et al., 2013; Wang et al., 2018). However, such methods heavily depend on high-quality structural information and sufficient labeled training data, and thus often suffer from limited robustness in scenarios with noise or sparse supervision.

Recently, the emergence of large language models (LLMs) has brought revolutionary advances to a wide range of KG-related tasks (Sun et al., 2023; Zhang and Soh, 2024). Owing to their strong semantic understanding and reasoning capabilities, LLMs substantially enhance the modeling of entities and their semantic and relational information. Recently, several studies have incorporated LLMs into EA and achieved state-of-the-art (SOTA) performance on multiple benchmark datasets (Jiang et al., 2024a; Yang et al., 2024a,b). These LLM-based EA methods typically treat LLMs as the final decision maker, selecting the optimal aligned entity from candidate sets through multi-round voting or reflection mechanisms.

Despite the remarkable improvements achieved by LLM-based EA methods, two key issues remain. **First, LLM-based EA methods lack interpretability.** Most existing approaches directly feed entities and their triples into LLMs and obtain the alignment results. Consequently, the entire alignment process remains a black-box decision

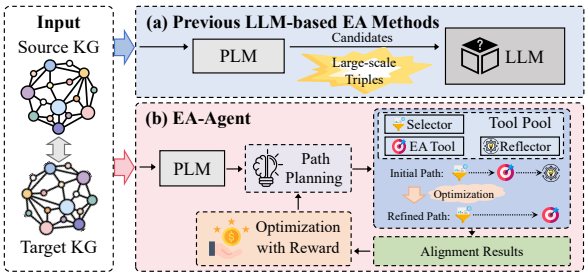

Figure 1: A simple comparison of previous LLM-based EA methods and our proposed EA-Agent.

procedure, making it difficult to identify which information is critical to alignment decisions. This opacity hinders error analysis and prevents systematic correction of erroneous alignments. **Second, the scale of triples poses severe efficiency issues.** Current methods usually input a large number of attribute and relation triples into LLMs, which significantly increases prompt length and inference cost, leading to substantial token consumption. Moreover, many of these triples are redundant or even noisy, which distorts the model's judgment and further undermines the stability and reliability of the alignment results.

To address the above limitations, we propose **EA-Agent**, a reasoning-driven agent for entity alignment. As shown in Figure 1, EA-Agent introduces a new learning paradigm that decomposes EA into a structured reasoning process with multi-step planning and execution. Specifically, we construct a tool pool consisting of triple selectors, an EA tool, and an (optional) reflector. These components together form a structured reasoning process of "*information selection → alignment decision → result verification*", enabling the agent to autonomously plan and execute tool invocation paths to produce interpretable and controllable alignment decisions. An *agent optimization mechanism* is further proposed to jointly refine its path planning and alignment behaviors within this structured reasoning process. Moreover, to mitigate the efficiency issues caused by large-scale triples, EA-Agent incorporates *attribute and relation triple selectors* into the tool pool, which filter redundant triples before LLM invocations, significantly reducing token consumption while preserving highly discriminative information. Experimental results on EA benchmarks demonstrate the effectiveness of EA-Agent.

In summary, our contributions are as follows:

1) We propose EA-Agent, a novel reasoning-driven agent for EA, which introduces a new learning paradigm that formulates EA as a structured reasoning process with multi-step planning and execution.

2) We design an agent optimization mechanism based on reward-guided offline policy optimization, which enables the agent to iteratively refine its tool planning and alignment behaviors through path rewriting.

3) Extensive experiments on three public benchmark datasets demonstrate that EA-Agent outperforms existing SOTA methods, achieving up to 3.17% improvement in Hits@1 and consistent gains in MRR.

2 Related Work

Existing EA methods are predominantly built upon knowledge representation learning, where entities from two KGs are encoded into a shared embedding space and aligned via similarity computation. These approaches can be broadly categorized into three types: translation-based methods, GNN–based methods, and pretrained language model (PLM)–based methods.

Translation-based EA methods, such as TransE (Bordes et al., 2013) and its variants (Lin et al., 2015; Chen et al., 2016; Zhu et al., 2017; Sun et al., 2017, 2018), learn low-dimensional embeddings for entities and relations in different KGs and map them into a unified vector space, where EA is performed via distance minimization. **GNN-based EA approaches** (Wang et al., 2018; Wu et al., 2019a; Li et al., 2019; Wu et al., 2019b) generate entity embeddings by aggregating neighborhood information. GCN-Align (Wang et al., 2018) is the first to apply GCN to EA, encoding graph structures and mapping entities into a unified embedding space. KECG (Li et al., 2019) leverages graph attention networks to down-weight irrelevant neighbors, mitigating noise introduced by heterogeneous KG structures.

With the rise of pretrained language models (PLMs), **PLM–based EA methods** have attracted increasing attention. These methods leverage the rich semantic knowledge contained in PLMs to encode entities. BERT-INT (Tang et al., 2020) encodes entity descriptions and interaction information using BERT. TEA (Zhao et al., 2023) transforms both attribute and relation triples into unified text sequences and formulates EA as a bidirectional textual entailment task between entities

across KGs. More recently, with the emergence of LLMs featuring stronger reasoning abilities, a new line of LLM-based EA approaches has been proposed. ChatEA (Jiang et al., 2024a) reformulates KG structures into code-like formats to enhance the understanding of LLMs, and adopts a two-stage alignment strategy to improve accuracy. Seg-Align (Yang et al., 2024a) introduces sample segmentation to reduce token consumption and enables LLMs to handle large-scale EA. LLMEA (Yang et al., 2024b) feeds entity embedding similarities and edit distances into an LLM for reasoning. LLM-Align (Chen et al., 2024) employs heuristic-based triple importance estimation and a multi-round voting mechanism to stabilize LLM predictions.

Unlike previous work, we propose a reasoning-driven agent that addresses EA through explicit multi-step tool planning and execution. This paradigm facilitates structured and interpretable reasoning, while addressing efficiency issues through triple selection.

3 Problem Definition

In this section, we first introduce the formal definitions of knowledge graphs and entity alignment. Based on these definitions, we present our task formulation.

Definition 1 *(Knowledge Graph) A knowledge graph is defined as* $\mathcal{KG} = \{\mathcal{E}, \mathcal{R}, \mathcal{A}, \mathcal{V}, \mathcal{T}^a, \mathcal{T}^r\}$*, where* $\mathcal{E}$*,* $\mathcal{R}$ *denote the sets of entities and relations, respectively;* $\mathcal{A}$ *and* $\mathcal{V}$ *represent the sets of attributes and attribute values. A knowledge graph consists of two categories of triples: attribute triples and relation triples. Attribute triples describe the attributes of entities and are defined as* $\mathcal{T}^a = \{(e, a, v) \mid e \in \mathcal{E}, a \in \mathcal{A}, v \in \mathcal{V}\}$*. Relation triples describe different relations between entities and are defined as* $\mathcal{T}^r = \{(h, r, t) \mid h, t \in \mathcal{E}, r \in \mathcal{R}\}$.

Definition 2 *(Entity Alignment) Given a source knowledge graph* $\mathcal{KG}_s = \{\mathcal{E}_s, \mathcal{R}_s, \mathcal{A}_s, \mathcal{V}_s, \mathcal{T}_s^a, \mathcal{T}_s^r\}$ *and a target knowledge graph* $\mathcal{KG}_t = \{\mathcal{E}_t, \mathcal{R}_t, \mathcal{A}_t, \mathcal{V}_t, \mathcal{T}_t^a, \mathcal{T}_t^r\}$*, the task of EA is to identify pairs of entities that refer to the same real-world object across the two graphs. Formally, the goal is to construct the set of aligned entity pairs* $\mathcal{S}_{st} = \{(e_i, e_j) \mid e_i \in \mathcal{E}_s, e_j \in \mathcal{E}_t, e_i \Leftrightarrow e_j\}$*, where the notation* $e_i \Leftrightarrow e_j$ *indicates that* e_i *from* $\mathcal{KG}_s$ *and* e_j *from* $\mathcal{KG}_t$ *share the same semantics and thus correspond to the same real-world object.*

In this work, we consider EA on knowledge graphs as a multi-step decision-making problem. Given a source entity e_s from the source knowledge graph $\mathcal{KG}_s$ and a candidate entity set $\mathcal{C}(e_s)$ retrieved from the target knowledge graph $\mathcal{KG}_t$, the objective is to determine the corresponding target entity $\hat{e}_t \in \mathcal{C}(e_s)$.

4 Method

4.1 Overview

We propose EA-Agent, a reasoning-driven agent that decomposes EA into a multi-step process of tool planning and execution. The available tools for EA-Agent include an Attribute Triple Selector, a Relation Triple Selector, an Entity Alignment Tool, and a Reflector. Figure 2 presents the overall architecture of EA-Agent, which consists of three key stages. In the **Path Planning** stage, EA-Agent observes various statistics of the source entity and the similarity scores of its candidate entities to generate a tool path. In the **Tool Calling** stage, EA-Agent sequentially invokes these tools specified in the planned path. The Attribute and Relation Triple Selectors distill informative triples to streamline the alignment process, while the Entity Alignment Tool and the Reflector collaborate to generate the initial alignment results. In the **Agent Optimization** stage, a reward function evaluates both the alignment accuracy and the quality of the tool path. Guided by this reward, EA-Agent is further fine-tuned via the offline policy updating to continuously improve its planning and decision-making capabilities.

4.2 Path Planning

To initialize the alignment process, we utilize TEA (Zhao et al., 2023) to retrieve the top-k semantically similar entities as candidates, serving as essential inputs for the subsequent reasoning process.

Given the retrieved candidates, EA-Agent performs path planning by jointly considering the structural characteristics of the source entity and the similarity scores of its candidates. The objective of path planning is to enable the agent to autonomously design a multi-step tool path by determining which tools to invoke and in what order. Based on these observed signals, EA-Agent assesses both the informativeness of the entity features and the uncertainty of the candidate set: it first decides whether to invoke attribute or rela-

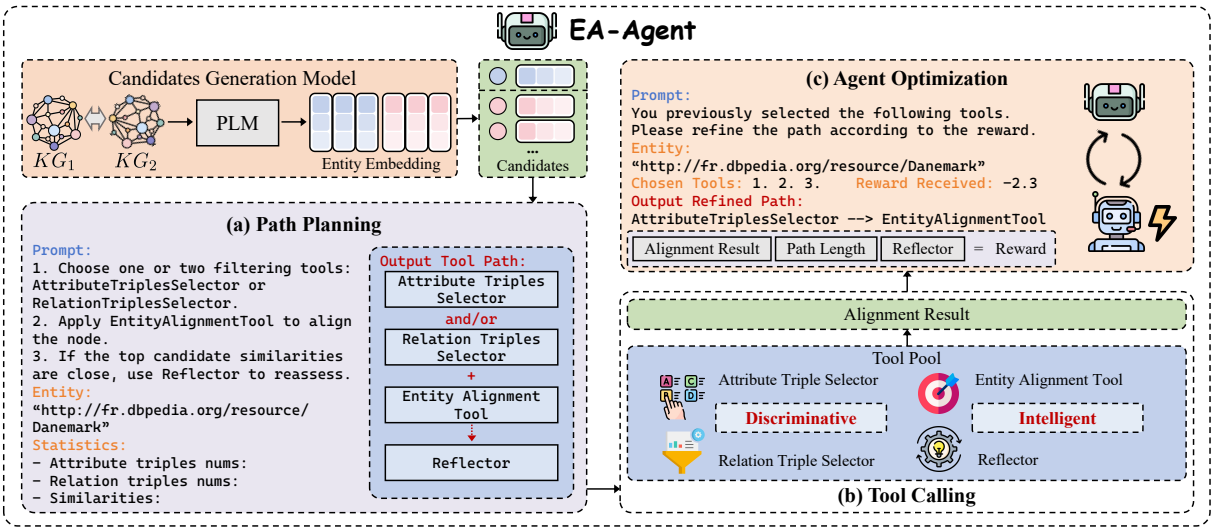

Figure 2: The overview of our proposed EA-Agent, which consists of three main stages: (a) Path Planning stage, (b) Tool Calling stage, and (c) Agent Optimization stage.

tion triple selectors to filter redundant information, then applies the Entity Alignment Tool, and finally activates the Reflector only when the candidate similarities indicate ambiguity.

To support this decision process, we provide EA-Agent with key statistics extracted from both the source entity and the candidate set:

Entity Statistics

```
Attribute triples: {attr_cnt_all}
Attribute types: {attr_cnt}
Relation triples: {rel_cnt_all}
Relation types: {rel_cnt}
Has name attribute: {signal_attr}
Candidate similarities: top1={top1_score},
top2={top2_score}, top3={top3_score}
```

This structured reasoning process ensures that each reasoning step plays a clear role in the alignment procedure, enabling EA-Agent to generate an interpretable and adaptive tool path.

4.3 Tool Calling

Once the tool path is determined during the Path Planning stage, the system executes each tool step-by-step according to the planned order. EA-Agent provides a tool pool consisting of an Attribute Triple Selector, a Relation Triple Selector, an Entity Alignment Tool, and a Reflector. The execution follows a hierarchical pipeline: the triple selectors first extract the most discriminative structural and semantic information from raw triples, which is then utilized by the Entity Alignment Tool to produce an initial prediction. When the alignment result is uncertain, the Reflector re-evaluates and revises the initial prediction based on prior contextual signals. This Tool Calling stage enables EA-Agent to exert fine-grained control over the information utilized during cross-KG alignment, resulting in more robust and reliable alignment performance. The concrete descriptions of the tools used are given here.

- **Attribute Triple Selector**: It aims to retain the most informative attributes for entity discrimination. It combines an entropy-based criterion with predefined important attributes to avoid losing key semantic signals. The entropy of an attribute a is computed as

$$H(a) = -\sum_{v \in V(a)} p(v) \log p(v), \qquad (1)$$

where $V(a)$ denotes the set of attribute values and $p(v)$ represents the empirical distribution of value v among the candidate entities. Attributes with lower entropy are considered more discriminative and are preferentially selected, while important attributes are always preserved.

- **Relation Triple Selector**: It is inspired by inverse-frequency. Given a relation r with frequency $freq(r)$ in the graph and the total number of relation triples N, its score is defined as

$$I(r) = \log\left(\frac{N}{freq(r)+1}\right). \qquad (2)$$

Rare relations are assigned higher scores since they are more discriminative for EA, similar

to the inverse document frequency (IDF) principle in information retrieval.

- **Entity Alignment Tool**: It is driven by an LLM, which takes the selected attribute and relation triples as input and outputs the final alignment prediction by jointly reasoning over semantic, attribute, and relational information.
- **Reflector**: It is an LLM-based module for result verification and correction. When the alignment is uncertain, it re-evaluates the candidates based on prior context and provides a refined prediction, helping to reduce hallucinations and improve robustness.

4.4 Agent Optimization

Based on the above two stages, EA-Agent is capable of generating reasonable tool usage paths and completing EA. However, our experiments show that, due to the inherent instability of LLMs, single-round planning can result in redundant or inefficient paths as well as unstable alignment outcomes (See details in Appendix A.). To address this issue, we introduce an Agent optimization stage that continuously provides feedback and correction signals to EA-Agent through a reward function, enabling the path planning policy to be progressively improved.

Reward function. To evaluate the quality of the tool paths and to provide supervision signals for policy optimization, we design a comprehensive reward function. The reward function accounts for not only the correctness of the final alignment but also the efficiency of the tool path and the necessity of Reflector invocation. In this way, EA-Agent is guided to learn a more efficient and stable tool planning strategy while maintaining high alignment accuracy.

Specifically, for each source entity, after the agent completes the tool calling stage, it obtains an alignment prediction $\hat{y}$. The corresponding reward score is composed of three factors:

$$\gamma = \gamma_{\mu} + c \cdot \gamma_{\text{ref}} + \gamma_{\text{e}}. \tag{3}$$

First, γ_{μ} measures the correctness of the final alignment result and serves as the core component of the reward function. If the predicted result $\hat{y}$ matches the ground-truth label y, the reward is set to 1; otherwise, it is set to 0:

$$\gamma_{\mu} = \begin{cases} 1, & \text{if } \hat{y} = y, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Second, γ_{ref} assesses the rationality of activating the Reflector. This term encourages the Reflector to be activated when necessary to correct erroneous predictions, while suppressing unnecessary or even harmful reflections. Let $\hat{y}$ denote the initial prediction and $\tilde{y}$ denote the refined prediction after reflection. It is defined as:

$$\gamma_{\text{ref}} = \begin{cases} 1, & \text{if } \hat{y} \neq y \text{ and } \tilde{y} = y, \\ -\alpha, & \text{if } \hat{y} = y \text{ and } \tilde{y} = y, \\ -1, & \text{if } \hat{y} = y \text{ and } \tilde{y} \neq y. \end{cases} \tag{5}$$

When the initial prediction is incorrect and is successfully corrected by the Reflector, a positive reward is assigned; when the initial prediction is already correct but the Reflector is unnecessarily invoked without changing the result, a mild penalty is imposed to discourage redundant reflection; when the Reflector mistakenly alters a correct prediction into an incorrect one, the maximum penalty is applied.

Finally, γ_{e} constrains the computational overhead by penalizing long tool paths. Let the path length be l, and we define:

$$\gamma_{\text{e}} = e^{-\beta \cdot l}, \tag{6}$$

where β is a coefficient used to balance alignment accuracy and path complexity. Longer paths lead to larger penalties, thereby encouraging EA-Agent to select simpler and more efficient tool invocation paths while preserving alignment performance.

Policy Updating. After obtaining the complete reward signals, we adopt an offline policy updating strategy to continuously optimize the agent's path planning policy. Specifically, for each tool invocation path, we preserve three types of key information to construct a policy update triple: (1) the initial tool path generated by EA-Agent; (2) the corresponding reward score γ; and (3) the refined path rewritten by EA-Agent under reward guidance. These policy update triples, which contain preference information, are incorporated into an offline trajectory dataset $\mathcal{D}^*$. During training, we employ supervised fine-tuning (SFT) to optimize the agent policy model, with the objective function defined as:

$$\mathcal{L}_{\text{SFT}} = -\mathrm{E}_{(q,\mu)\sim\mathcal{D}^*} \left[\log \pi_{\theta}(\mu \mid q) \right], \tag{7}$$

where q denotes the input context for path planning, μ denotes the target tool path obtained under reward guidance, and π_{θ} is the agent policy model

parameterized by θ. After each iteration of policy updating, the updated agent is redeployed to execute path planning, tool calling, and reward evaluation, continuously generating new trajectory data for the next iteration of offline optimization.

4.5 Complexity Analysis

EA-Agent is a multi-stage agent whose overall workflow follows a closed-loop iterative process consisting of path planning, tool execution, reward evaluation, and policy updating. At each iteration, the agent plans and executes a tool invocation path, evaluates the resulting trajectory using our proposed reward function, and updates its policy in an offline manner. This forms a closed-loop "planning–execution–evaluation–updating" process that iterates until convergence, enabling EA-Agent to progressively learn more efficient and stable tool planning strategies.

From the perspective of computational complexity, EA-Agent adopts the offline policy training and inference rather than online decision-making, which achieves better training stability and controllable computational cost. In addition, the triple selectors effectively reduce prompt length and inference cost by filtering redundant triples. The complete pseudocode of EA-Agent is provided in Appendix B.

5 Experiments

In this section, we evaluate EA-Agent on public datasets. We conduct extensive experiments to demonstrate the effectiveness of our method by answering the following research questions: **RQ1**: How does EA-Agent perform compared with SOTA baselines on benchmark EA datasets? **RQ2**: How do different components of EA-Agent affect the final alignment performance? **RQ3**: Can EA-Agent provide an interpretable alignment process through tool planning and agent optimization? **RQ4**: How does EA-Agent compare in terms of cost and efficiency to an LLM-based baseline?

5.1 Experiment Settings

5.1.1 Datasets

We evaluate EA-Agent on the widely used DBP15K dataset (Sun et al., 2017), which consists of three cross-lingual EA datasets: FR–EN, JA–EN, and ZH–EN. The statistical characteristics of the datasets are reported in Appendix D.

5.1.2 Baseline

We compare EA-Agent with 10 representative baselines across three categories: translation-based, GNN-based, and PLM-based methods. All baseline models follow the hyperparameter settings reported in their original papers. For brevity, we detail the specific models in Appendix C.

5.1.3 Implement Details

We use TEA (Zhao et al., 2023) as the SLM to generate Top-10 candidate entity lists, due to its strong Hits@10 performance. We adopt Qwen3-32B, Qwen3-8B (Yang et al., 2025), and Llama3-8B-Instruct (Dubey et al., 2024) as backbone LLMs for EA-Agent. The agent is optimized using LoRA with the AdamW optimizer for three epochs. For fair comparison with prior work, all datasets are split into training and test sets with a ratio of 3:7. All experiments are conducted on two NVIDIA A100 80G GPUs. Detailed hyperparameters are provided in Appendix E.

5.2 Performance Comparison (RQ1)

Table 1 reports the overall performance of EA-Agent on the DBP15K datasets. In all experiments, we use TEA to generate Top-10 candidate entities and adopt Qwen3-32B as the backbone agent. Thus, the reported Hits@10 represents the recall performance of the PLM-based EA model.

The results for RQ1 show that EA-Agent consistently outperforms state-of-the-art EA methods across all datasets. EA-Agent achieves competitive performance on FR–EN and establishes new SOTA results on JA–EN and ZH–EN, exceeding the strongest baselines by 3.17 and 1.37 Hits@1 points, respectively. Despite TEA already providing high-recall candidate sets, the consistent gains in Hits@1 and MRR indicate that EA-Agent is effective at selecting the correct entity from high-quality candidates.

We compare EA-Agent with LLM-based baselines: LLMEA (ERNIE), ChatEA (GPT-4), and Seg-Align (GPT-3.5). Their strong performance largely derives from the underlying LLMs' inherent reasoning capacity. Conversely, EA-Agent achieves comparable performance using open-source models, demonstrating that gains primarily stem from our proposed tool planning and optimization rather than the LLM itself.

Table 1: Overall entity alignment performance on the DBP15K datasets.

Method	FR-EN			JA-EN			ZH-EN		
	Hits@1	MRR	Hits@10	Hits@1	MRR	Hits@10	Hits@1	MRR	Hits@10
MTransE	24.40	0.335	55.6	27.90	0.349	57.5	30.80	0.364	61.4
JAPE	32.40	0.430	66.7	36.30	0.476	68.5	41.20	0.490	74.5
GCN-Align	37.30	0.532	74.5	39.90	0.546	74.5	41.30	0.549	74.4
RDGCN	88.60	0.911	95.7	76.70	0.812	89.5	70.80	0.746	84.6
BERT-INT	98.70	0.990	99.2	80.60	0.820	83.5	81.40	0.820	83.7
SDEA	96.60	0.980	99.5	84.80	0.890	95.2	87.00	0.910	96.6
TEA	95.80	0.970	99.7	89.00	0.920	97.8	84.50	0.887	98.2
LLMEA	95.70	0.957	–	91.10	0.911	–	89.80	0.898	–
ChatEA	99.00	0.995	–	–	–	–	–	–	–
Seg-Align	98.70	0.987	–	90.70	0.907	–	95.30	0.953	–
EA-Agent	**99.00**	**0.996**	**99.7***	**94.27**	**0.950**	**97.8***	**96.67**	**0.971**	**98.2***

*Hits@10 of EA-Agent is determined by the Top-10 candidates retrieved by TEA.

Table 2: Ablation study on the impact of different components in EA-Agent.

Method	FR–EN		JA–EN	
	Hits@1	MRR	Hits@1	MRR
All Components	**97.30**	**0.975**	**90.67**	**0.913**
w/o Path Planning	94.85	0.950	87.29	0.880
w/o Selector	93.34	0.935	86.75	0.847
LLM-Only	89.44	0.901	83.50	0.841

Table 3: Performance (%) of EA-Agent with different LLM backbones on DBP15K.

LLM	FR–EN	JA–EN	ZH–EN
Qwen3-8B	88.10 / .887	85.95 / .867	83.63 / .840
Llama3-8B-Instruct	92.34 / .929	89.69 / .901	90.13 / .907
Qwen3-32B	**99.00 / .996**	**94.27 / .950**	**96.67 / .971**

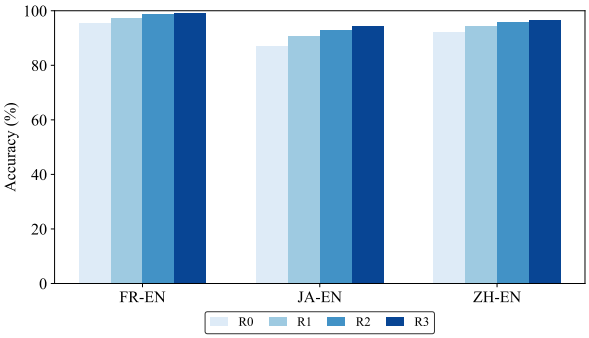

Figure 3: Accuracy improvements over multiple optimization rounds on the FR–EN, JA–EN, and ZH–EN datasets. Each group of bars corresponds to a dataset, where colors from light to dark denote Round 0 to Round 3, respectively.

5.3 Ablation Study (RQ2)

To answer RQ2 and systematically evaluate the contribution of each component in EA-Agent, we conduct a series of ablation studies on the DBP15K benchmark. The ablation experiments focus on three key aspects of the agent: **the Path Planning stage and the associated triple selectors**, **different LLMs** used as the agent backbone, and **the Agent Optimization stage**.

Path Planning stage. We evaluate the path planning and triple selection mechanisms on the FR–EN and JA–EN datasets using three ablated variants: (1) LLM-Only, which performs EA without triples; (2) w/o Selector, which feeds all triples without filtering; and (3) w/o Path Planning, which invokes all tools without adaptive planning. The first two variants assess the impact of triple selection, while the third evaluates the role of adaptive path planning.

As shown in Table 2, directly using an LLM for EA performs poorly, indicating that structured triples provide essential alignment signals. Although using all triples improves performance, it also introduces noise and token redundancy. Moreover, compared with the full EA-Agent, the variant without path planning exhibits a slight performance degradation, mainly because the Reflector is overused, which in turn leads to more erroneous corrections.

Different LLMs as backbone. We further evaluate EA-Agent with three backbone LLMs—Qwen3-32B, Qwen3-8B, and Llama-3-8B-Instruct—on the FR–EN, JA–EN, and ZH–EN datasets. The experimental results are reported in Table 3. EA-Agent achieves the best performance with Qwen3-32B, and the comparison with Qwen3-8B indicates a clear scaling-law effect. Llama-3-8B-Instruct slightly outperforms Qwen3-8B, likely due to its instruction tuning that enhances agent reasoning.

Agent Optimization stage. We analyze EA-Agent across multiple optimization rounds on the FR–EN,

JA–EN, and ZH–EN datasets, as shown in Figure 3. Alignment accuracy consistently improves with successive rounds, demonstrating that the reward function effectively guides EA-Agent toward better tool-planning strategies. The gains gradually diminish after the second round, indicating convergence to a stable and near-optimal policy. Based on this observation, we adopt three optimization rounds in all experiments to balance performance and computational cost.

5.4 Case Study (RQ3)

To further illustrate the interpretability and the effectiveness of both the tool planning and Agent Optimization stages, we present a representative case study from the FR-EN dataset.

Given a source entity "http://fr.dbpedia.org/resource/Saint-Isidore_(Roussillon)" in the source knowledge graph, EA-Agent first generates a tool path at Round 0 as *AttributeTripleSelector* → *RelationTripleSelector* → *EntityAlignmentTool* → *Reflector*. Following this, the Entity Alignment Tool produces an initial prediction "http://dbpedia.org/resource/Saint-Isidore,_MontÃľrÃľgie,_Quebec". However, the Reflector incorrectly revises it to an incorrect final alignment result "http://dbpedia.org/resource/Saint-Isidore,_ChaudiÃÎre-Appalaches,_Quebec". Due to unnecessary reflector invocation, EA-Agent receives a low reward. During the agent optimization phase, guided by the reward signal, the agent rewrites the path as *AttributeTripleSelector* → *RelationTripleSelector* → *EntityAlignmentTool*. In the subsequent round, the optimized path produces the correct target entity and achieves a higher reward.

This case demonstrates that explicit tool planning yields transparent reasoning paths, while reward-guided optimization effectively eliminates suboptimal tool usage, improving both alignment accuracy and efficiency.

5.5 Efficiency Analysis (RQ4)

We further analyze the efficiency of EA-Agent from three aspects: token cost, inference time, and training cost.

Table 4 reports the average token consumption per entity for different LLM-based EA methods. Seg-Align reduces token usage by discarding triples and using only entity names, sacrificing accuracy. In contrast, EA-Agent leverages informative attribute and relation triples while consuming only 672 tokens, significantly less than ChatEA, which also considers triples. This efficiency gain mainly benefits from the Path Planning stage and the triple selectors that filter redundant triples before LLM invocation, allowing EA-Agent to better balance efficiency and effectiveness.

Table 4: Average token consumption per entity for different LLM-based EA methods.

Method	Avg. Tokens per Entity
ChatEA	9,803
Seg-Align	159
EA-Agent	672

Table 5: Average time cost per entity (path planning + entity alignment) on different datasets.

Dataset	FR-EN	JA-EN	ZH-EN
Time (s)	1.53+2.35	1.50+2.48	1.55+2.68

We further analyzed the average time cost per entity on the FR-EN, JA-EN, and ZH-EN datasets, as shown in Table 5. These results indicate that EA-Agent is highly efficient in practice, with a short end-to-end processing time per entity, making it suitable for large-scale EA scenarios.

Regarding training efficiency, EA-Agent employs parameter-efficient fine-tuning with LoRA and updates only a small subset of backbone parameters. Agent optimization is performed offline on collected trajectories, avoiding costly online reinforcement learning. Consequently, training remains affordable even with large backbone models and typically converges within a few iterations.

6 Conclusion

In this paper, we propose EA-Agent, a reasoning-driven agent that addresses EA through structured multi-step planning and tool execution, overcoming the limitations of existing LLM-based methods in interpretability and efficiency. By integrating explicit path planning, selective tool calling, and reward-guided agent optimization, EA-Agent enables transparent and effective alignment decisions. Extensive experiments on three public datasets demonstrate that EA-Agent consistently achieves state-of-the-art performance.

Limitations

Despite its effectiveness, EA-Agent has several limitations. First, the set of tools used by the agent is manually designed based on prior knowledge of the entity alignment task. While these tools are sufficient to achieve strong performance in our experiments, extending EA-Agent to automatically design tool sets remains an interesting direction for future work. Second, the agent optimization in EA-Agent is performed in an offline and supervised manner using reward-guided trajectory rewriting, rather than end-to-end reinforcement learning. While this design improves training stability, it may limit exploration in more complex long-horizon planning scenarios.

References

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

Muhao Chen, Yingtao Tian, Mohan Yang, and Carlo Zaniolo. 2016. Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. *arXiv preprint arXiv:1611.03954*.

Xuan Chen, Tong Lu, and Zhichun Wang. 2024. Llm-align: Utilizing large language models for entity alignment in knowledge graphs. *arXiv preprint arXiv:2412.04690*.

Yachao Cui, Hongli Yu, Xiaoxu Guo, Han Cao, and Lei Wang. 2024. Rakcr: Reviews sentiment-aware based knowledge graph convolutional networks for personalized recommendation. *Expert Systems With Applications*, 248:123403.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2025. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9505–9523.

Xuhui Jiang, Yinghan Shen, Zhichao Shi, Chengjin Xu, Wei Li, Zixuan Li, Jian Guo, Huawei Shen, and Yuanzhuo Wang. 2024a. Unlocking the power of large language models for entity alignment. *arXiv preprint arXiv:2402.15048*.

Yangqin Jiang, Yuhao Yang, Lianghao Xia, and Chao Huang. 2024b. Diffkg: Knowledge graph diffusion model for recommendation. In *Proceedings of the 17th ACM international conference on web search and data mining*, pages 313–321.

Chengjiang Li, Yixin Cao, Lei Hou, Jiaxin Shi, Juanzi Li, and Tat-Seng Chua. 2019. Semi-supervised entity alignment via joint knowledge embedding model and cross-graph model. Association for Computational Linguistics.

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015. Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379*.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-sequence knowledge graph completion and question answering. *arXiv preprint arXiv:2203.10321*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M Ni, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697*.

Zequn Sun, Wei Hu, and Chengkai Li. 2017. Cross-lingual entity alignment via joint attribute-preserving embedding. In *International semantic web conference*, pages 628–644. Springer.

Zequn Sun, Wei Hu, Qingheng Zhang, and Yuzhong Qu. 2018. Bootstrapping entity alignment with knowledge graph embedding. In *Ijcai*, volume 18.

Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, and Wenjie Zhang. 2025. Paths-over-graph: Knowledge graph empowered large language model reasoning. In *Proceedings of the ACM on Web Conference 2025*, pages 3505–3522.

Xiaobin Tang, Jing Zhang, Bo Chen, Yang Yang, Hong Chen, and Cuiping Li. 2020. Bert-int: A bert-based interaction model for knowledge graph alignment. *interactions*, 100:e1.

Jiapu Wang, Kai Sun, Linhao Luo, Wei Wei, Yongli Hu, Alan W Liew, Shirui Pan, and Baocai Yin. 2024a. Large language models-guided dynamic adaptation for temporal knowledge graph reasoning. *Advances in Neural Information Processing Systems*, 37:8384–8410.

Shijie Wang, Wenqi Fan, Yue Feng, Lin Shanru, Xinyu Ma, Shuaiqiang Wang, and Dawei Yin. 2025. Knowledge graph retrieval-augmented generation for llm-based recommendation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27152–27168.

Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024b. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19206–19214.

Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. 2018. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 349–357.

Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, Rui Yan, and Dongyan Zhao. 2019a. Relation-aware entity alignment for heterogeneous knowledge graphs. *arXiv preprint arXiv:1908.08210*.

Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, and Dongyan Zhao. 2019b. Jointly learning entity and relation representations for entity alignment. *arXiv preprint arXiv:1909.09317*.

Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. 2024. Retrieval-augmented generation with knowledge graphs for customer service question answering. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 2905–2909.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Linyan Yang, Jingwei Cheng, and Fu Zhang. 2024a. Advancing cross-lingual entity alignment with large language models: Tailored sample segmentation and zero-shot prompts. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8122–8138.

Linyao Yang, Hongyang Chen, Xiao Wang, Jing Yang, Fei-Yue Wang, and Han Liu. 2024b. Two heads are better than one: Integrating knowledge from knowledge graphs and large language models for entity alignment. *arXiv preprint arXiv:2401.16960*.

Kaisheng Zeng, Chengjiang Li, Lei Hou, Juanzi Li, and Ling Feng. 2021. A comprehensive survey of entity alignment for knowledge graphs. *AI Open*, 2:1–13.

Bowen Zhang and Harold Soh. 2024. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. *arXiv preprint arXiv:2404.03868*.

Rui Zhang, Bayu Distiawan Trisedya, Miao Li, Yong Jiang, and Jianzhong Qi. 2022. A benchmark and comprehensive survey on knowledge graph entity alignment via representation learning. *The VLDB Journal*, 31(5):1143–1168.

Yu Zhao, Yike Wu, Xiangrui Cai, Ying Zhang, Haiwei Zhang, and Xiaojie Yuan. 2023. From alignment to entailment: A unified textual entailment framework for entity alignment. *arXiv preprint arXiv:2305.11501*.

Ziyue Zhong, Meihui Zhang, Ju Fan, and Chenxiao Dou. 2022. Semantics driven embedding learning for effective entity alignment. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2127–2140. IEEE.

Beibei Zhu, Ruolin Wang, Junyi Wang, Fei Shao, and Kerun Wang. 2024. A survey: knowledge graph entity alignment research based on graph embedding. *Artificial Intelligence Review*, 57(9):229.

Hao Zhu, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2017. Iterative entity alignment via joint knowledge embeddings. In *IJCAI*, volume 17, pages 4258–4264.

A Analysis of Single-Round Planning

To further analyze the stability and efficiency of the planning strategy, we compare single-round planning (Round 0) with optimized planning after multiple optimization rounds (Round 3). Specifically, we evaluate the Recall of the Reflector and the average tool path length under the two settings, as reported in Table 6 and Table 7, respectively. In addition, the alignment accuracy across planning rounds is illustrated in Figure 4.

The results reveal clear limitations of single-round planning. As shown in Table 6, the Reflector recall in Round 0 is significantly higher than that in Round 3, indicating that EA-Agent frequently resorts to reflection due to uncertainty in early-stage planning. Meanwhile, Table 7 shows that single-round planning produces longer tool paths on average, suggesting the presence of redundant and inefficient tool invocations. Consistently, Figure 4 demonstrates that alignment accuracy under single-round planning is notably lower than that achieved after optimization. These observations indicate that planning based solely on LLM is prone to redundancy and instability, leading to suboptimal alignment results.

Table 6: Comparison of the Reflector recall between single-round planning (Round 0) and optimized planning (Round 3).

Planning Round	**Reflector Recall (%)**
Round 0	90.07
Round 3	19.69

Algorithm 1: EA-Agent Training Phase

Input: Training entities $\mathcal{E}_{\text{train}}$; candidate set $\mathcal{C} \subset \mathcal{KG}_t$; initial policy π_0
Output: Optimized agent policy π
Initialize policy $\pi \leftarrow \pi_0$;
for *round* $r = 1, \ldots, R$ **do**
  Initialize trajectory dataset $\mathcal{D}_r \leftarrow \emptyset$;
  for *each* $e_s \in \mathcal{E}_{train}$ **do**
    Retrieve candidate set $\mathcal{C}(e_s) \subset \mathcal{C}$;
    Compute statistics $\mathcal{S}(e_s)$;
    Construct query $q = (\mathcal{S}(e_s), \mathcal{C}(e_s))$;
    Sample tool path $\mu \sim \pi(\cdot \mid q)$;
    Execute μ to obtain prediction $\hat{e}_t$;
    Compute reward γ according to Eq. (3);
    $\mathcal{D}_r \leftarrow \mathcal{D}_r \cup \{(q, \mu, \gamma)\}$;
  Update policy π by minimizing $\mathcal{L}_{\text{SFT}}$ in Eq. (7);
return π

Algorithm 2: EA-Agent Inference Phase

Input: Test entities $\mathcal{E}_{\text{test}}$; candidate set $\mathcal{C} \subset \mathcal{KG}_t$; trained policy π
Output: Aligned entities $\hat{e}_t$
for *each* $e_s \in \mathcal{E}_{test}$ **do**
  Retrieve candidate set $\mathcal{C}(e_s) \subset \mathcal{C}$;
  Compute statistics $\mathcal{S}(e_s)$;
  Construct query $q = (\mathcal{S}(e_s), \mathcal{C}(e_s))$;
  Sample tool path $\mu \sim \pi(\cdot \mid q)$;
  Execute μ to obtain aligned entity $\hat{e}_t$;
return $\hat{e}_t$

B Pseudocode of the EA-Agent Framework

In this section, we illustrate the pseudocode of EA-Agent during the training and testing phases in Algorithm 1 and Algorithm 2, respectively.

C Baseline Details

We compare EA-Agent with 10 representative EA methods covering different categories. The translation-based methods include MTransE (Chen et al., 2016) and JAPE (Sun et al., 2017), the GNN-based methods include GCN-Align (Wang et al., 2018) and RDGCN (Wu et al., 2019a), and the PLM–based methods include BERT-INT (Tang et al., 2020), SDEA (Zhong et al., 2022), TEA (Zhao et al., 2023), LLMEA (Yang et al., 2024b), ChatEA (Jiang et al., 2024a), and Seg-Align (Yang et al., 2024a), among which LLMEA, ChatEA, and Seg-Align are LLM-based EA approaches. All baseline models follow the hyperparameter settings reported in their original papers.

Table 7: Average tool path length under different planning rounds.

Planning Round	Avg. Path Length
Round 0	3.61
Round 3	2.98

Table 8: Statistics of the DBP15K datasets.

Dataset	Lang.	Entity	Rel.	Attr.	Rel. triples	Attr. triples
FR–EN	(FR)	66,858	1,379	4,547	192,191	528,665
	(EN)	105,889	2,209	6,422	278,590	576,543
JA–EN	(JA)	65,744	2,043	5,882	164,373	354,619
	(EN)	95,680	2,096	6,066	233,319	497,230
ZH–EN	(ZH)	66,469	2,830	8,113	153,929	379,684
	(EN)	98,125	2,317	7,173	237,674	567,755

D Dataset Statistics

DBP15K is constructed from DBpedia, a large-scale multilingual knowledge graph that contains abundant inter-lingual links across different language editions. Three cross-lingual EA benchmark datasets are derived from its subgraphs, namely French–English (FR–EN), Japanese–English (JA–EN), and Chinese–English (ZH–EN).

Table 8 reports the detailed statistics of the DBP15K datasets used in our experiments.

E Detailed Implementation Hyperparameters

For LoRA-based fine-tuning, we set the rank to $r = 4$ with a scaling factor $\alpha = 8$. LoRA is applied only to the query and value projection matrices (q_{proj}, v_{proj}) in the attention layers. The dropout rate is set to 0.05, and no bias parameters are updated. We train EA-Agent for three epochs using the AdamW optimizer with an initial learning rate of 2×10^{-4}.

All input sequences are tokenized using the official tokenizer of each backbone model, with truncation and padding to a maximum length of

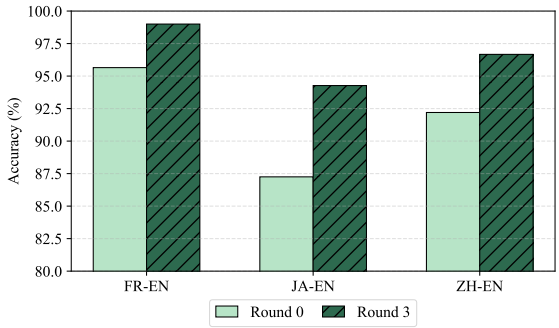

Figure 4: Alignment accuracy across different planning rounds.

1024 tokens. The per-device training batch size is set to 1, and gradient accumulation with 8 steps is adopted to form a larger effective batch size. Mixed-precision training with bfloat16 is enabled.

In the Tool Calling stage, both the attribute and relation triple selectors retain at most 5 triples. The path efficiency coefficient in the reward function is set to $\beta = 0.2$, and the mild penalty coefficient for the Reflector is set to $\alpha = 0.5$. During inference, deterministic decoding is used for Llama3-8B-Instruct, while a temperature of 0.1 is applied to Qwen models to reduce decoding variance. All experimental results are reported as the average of multiple independent runs.

F Prompt Templates for EA-Agent

To concretely instantiate the proposed EA-Agent framework, we design a set of structured prompts that govern path planning, tool calling, reflection, and policy optimization. For clarity and reproducibility, we present all prompt templates used in EA-Agent in this appendix.

Tool Pool Definition

Tool_Pool = [
{
"name": "*AttributeTripleSelector*",
"definition": "Selects important attribute triples of an entity by removing common or uninformative attributes.",
"usage": "*AttributeTripleSelector*[entity] → list of (entity, attribute, value)"
},
{
"name": "*RelationTripleSelector*",
"definition": "Selects informative relation triples (outgoing and incoming) based on their distinctiveness.",
"usage": "*RelationTripleSelector*[entity] → list of (subject, relation, object)"
},
{
"name": "*EntityAlignmentTool*",
"definition": "Aligns a source entity with the most similar target entity from a candidate list.",
"usage": "*EntityAlignmentTool*[source_entity, candidates] → best target entity"
},
{
"name": "*Reflector*",
"definition": "Reevaluates the alignment result and suggests a better match if needed.",
"usage": "*Reflector*[source_entity, candidates, initial_alignment] → confirmed or revised target"
}]

Source Entity Prompt

Descriptions: This prompt instructs the agent to perform path planning for entity alignment by selecting an appropriate sequence of tools.

You are an expert in **Entity Alignment**.

Steps:
1. Choose one or two filtering tools: *AttributeTripleSelector*, *RelationTripleSelector*.
2. Apply *EntityAlignmentTool* to align the entity.
3. If the top candidate similarities are close, use *Reflector* to reassess.

Available tools:
`{tool_pool}`

Entity: `{entity_iri}`
Statistics:
- Attribute triples: `{attr_cnt_all}`
- Attribute types: `{attr_cnt}`
- Relation triples: `{rel_cnt_all}`
- Relation types: `{rel_cnt}`
- Has name attribute: `{signal_attr}`
- Top-1 sim: `{top1_score}`, Top-2 sim: `{top2_score}`, Top-3 sim: `{top3_score}`

Output the sequence numbers and tool names only, one per line:
1. <ToolName>
2. <ToolName>
3. <ToolName> (optional)
4. <ToolName> (optional)

Entity Alignment Prompt

Descriptions: This prompt instructs the LLM-based Entity Alignment Tool to identify the corresponding target entity from a candidate set.

You are given a source entity and several candidate entities from another knowledge graph. Each entity is represented as triples (subject, predicate, object). Candidates are sorted by similarity.

Entity: `{source_iri}`
Triples:
`{source_triples}`

Candidates:
`{candidate_blocks}`

Please enter the IRI that best matches the candidate entity using the following format:
`[IRI]`

Reflection Prompt

Descriptions: This prompt enables the Reflector to verify and reassess the initial alignment result.

You are performing an entity alignment task. Given a source entity and several candidate target entities (with their triples), you previously selected one of them.

Now, reflect on whether that choice was optimal.

Entity: `{source_iri}`
Triples:
`{source_triples}`

Candidates:
`{candidate_blocks}`

Initial choice: `{initial_choice}`

Is this the best match?
- If yes, return it.
- If not, return a better one.

Please enter the best matching IRI using the following format:
`[IRI]`

Path Rewriting Prompt

Descriptions: This prompt instructs the agent to rewrite the previous tool path based on observed reward feedback.

You are optimizing the tool selection process for an entity alignment task.

Entity: `{entity}`
Candidate Similarities: `{top1_score}`, `{top2_score}`, `{top3_score}`
Previous Tools: `{old_tools}`
Reward: `{reward}`

Available tools:
- *AttributeTripleSelector*
- *RelationTripleSelector*
- *EntityAlignmentTool*
- *Reflector* (use only when similarities are close)

Please generate an improved sequence of tools.

Output the sequence numbers and tool names only, one per line:
1. <ToolName>
2. <ToolName>
3. <ToolName> (optional)
4. <ToolName> (optional)