CaRL-EM: Cost-Aware Reinforcement Learning for Entity Matching with LLMs

Anonymous ACL submission

Abstract

Entity matching (EM) requires fine-grained contextual understanding and domain knowledge. Recent work shows that large language models (LLMs) can serve as strong matchers across domains, but most methods either make independent pairwise decisions or rely on manually designed composite pipelines, thus lacking flexibility in realistic multi-candidate settings. At the same time, they typically ignore inference cost at scale. We formulate LLM-based EM with candidates as a cost-aware sequential decision problem and propose CaRL-EM, a reinforcement learning controller that manages LLM operations. Given the state of an anchor record, its candidate set, and the cost, CaRL-EM adaptively chooses among different operators (MATCH / COMPARE / SELECT / DECIDE) and model capacities to maximize a quality–cost objective. The policy interacts with abstract operators, allowing the same controller to be reused with different underlying LLM backends at inference time without retraining. Experiments on 7 benchmarks show that CaRL-EM (i) learns to dynamically plan the usage of inexpensive and expensive operators based on task complexity, (ii) achieves robust zero-shot transfer across diverse datasets and domains, and (iii) consistently achieves a better quality–cost trade-off than strong LLM-based baselines and manually designed pipelines, yielding a lower inference cost at comparable or higher quality.

1 Introduction

Entity matching (EM) is a core component of entity resolution pipelines, supporting data integration, knowledge base construction, and downstream analytics in multiple domains (Shahbazi et al., 2023). Given an anchor record and a set of candidates retrieved by blocking, an EM system must identify which candidates refer to the same real-world entity (Papadakis et al., 2020; Thirumuruganathan et al., 2021; Paulsen et al., 2023).

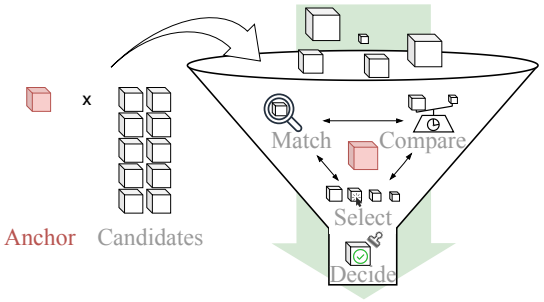

Figure 1: **Overview of CaRL-EM.** The RL agent ranks the candidates by sequentially choosing between tools MATCH, COMPARE and SELECT.

Recent work shows that large language models (LLMs) can rival supervised deep models in pairwise matching, even under zero- or few-shot prompting (Peeters and Bizer, 2023; Li et al., 2024). However, most LLM-based EM methods focus on whether an anchor matches each candidate in isolation, ignoring mutual interactions among candidates and global consistency constraints. In addition, EM at an industrial scale involves millions of anchors and large candidate sets, making cost a major concern (Konda et al., 2016). This becomes more significant with the application of LLMs.

To address these issues, COMEM (Wang et al., 2025) use a pipeline to allow more interaction among candidates. While these pipelines improve quality, they remain treating cost as a fixed byproduct of the architecture, applying the same sequence of operations regardless of an instance's difficulty (Chen et al., 2023; Ong et al., 2025).

In this paper, we formulate LLM-based multi-candidate EM with blocking as a cost-aware sequential decision problem and propose CaRL-EM, as shown in Figure 1, a reinforcement learning (RL) controller that manages LLM-based EM operations. For each anchor and its candidate set, CaRL-EM

maintains a compact state. We detail the state representation in §3.2. At each step, the controller selects a high level operator, MATCH, COMPARE, SELECT, or DECIDE, and a model capacity, thereby deciding whether to perform low-cost local refinement, more expensive listwise selection over the current shortlist of candidates, or terminate and output a prediction. The policy is trained with cost-aware rewards under an abstract two level cost model, more details in §3.1. CaRL-EM treats these operators as black box actions annotated with abstract cost and is separated from the underlying LLMs, so that once the policy has been trained, stronger LLMs can be plugged in at test time without retraining the controller. In summary, our work makes the following contributions:

- **Problem formulation.** We formulate LLM-based multi-candidate EM with blocking as a cost-aware sequential decision problem under RL, where a policy observes the matching state and accumulated cost information, and decides which operator to apply next. To the best of our knowledge, this is the first work to cast LLM-driven EM into a cost-aware sequential RL process.
- **CaRL-EM.** We propose CaRL-EM, a cost-aware RL controller that combines MATCH / COMPARE / SELECT / DECIDE operators and two level abstract cost models to reduce unnecessary calls. The controller is independent of specific LLMs, so backends can be swapped at test time without retraining.
- **High efficiency and performance.** On 7 benchmarks spanning products, citations, and movies, CaRL-EM outperforms the best manually designed composite pipelines, achieves higher F1 score, and **reduces 78%** of the cost This yields a better quality cost trade-off. Compared to domain-specific supervised EM models, CaRL-EM attains approximately 89% of their performance without any fine-tuning, while **reducing 94%** of the expense.

2 Related Work

2.1 Traditional and Pretrained EM

Early EM mainly used string similarity and manual rules (Papadakis et al., 2020; Barlaug and Gulla, 2021). While neural models such as DeepER and DeepMatcher improved semantic capture, they remain heavily dependent on labeled data (Ebraheem et al., 2017; Mudgal et al., 2018). Recent pretrained language models further leverage cross-encoders like DITTO (Li et al., 2020) or dual-encoders (Shah et al., 2018; Tracz et al., 2020). However, these methods typically treat $(anchor, candidate)$ pairs independently, ignoring mutual interactions and global consistency. Moreover, their reliance on fine-tuning limits their transferability to new domains (Li et al., 2020; Peeters and Bizer, 2021).

2.2 LLM-Based Strategies

LLMs can perform pairwise matching in a zero- or few-shot manner with performance often rivaling supervised models, while providing explanations of their decisions (Peeters et al., 2025). Most methods prompt the LLM to output a Yes/No decision for each pair, which keeps the pairwise limitation.

To go beyond independent decisions, COMEM analyzes three LLM-driven strategies for multi-candidate EM: MATCH, COMPARE, and SELECT (Wang et al., 2025). It manually builds a pipeline that uses cheap steps to re-rank candidates and then calls a stronger LLM to select the final match. However, the pipeline structure is the same for all anchors, and the inference cost is implicitly determined by the chosen pipeline.

2.3 Cost-Aware Control with RL

LLMs often work well without much tuning, but they can be costly to run at scale. This has led to growing interest in controlling LLM inference under resource constraints. Recent work uses lightweight policies or RL to decide when to call external tools or how much context to use, trading off accuracy for token cost (Feng et al., 2025; Zhang et al., 2024). These approaches are mainly studied in question answering or reasoning tasks rather than EM. In contrast, EM tasks that require large-scale LLM calls typically rely on fixed pipelines.

We cast multi candidate EM as a cost-aware sequential decision problem. CaRL-EM learns to chooses MATCH, COMPARE, SELECT, and DECIDE, as well as LLMs with different costs, to navigate the accuracy and cost trade-off.

3 CaRL-EM: A Cost-Aware RL Controller for LLM-Based EM

We consider the standard EM setting with blocking. For each anchor record a, the blocking retrieves a candidate set $C(a) = \{c_1, \ldots, c_n\}$. The goal is to decide the matching candidate, or None. We focus

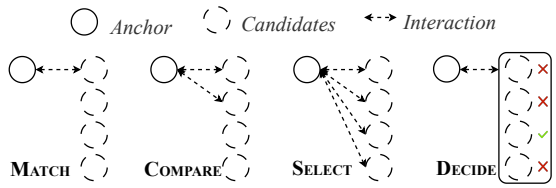

Figure 2: Interaction patterns of the operators used in CaRL-EM: **MATCH**, **COMPARE**, **SELECT**, and **DECIDE**.

on the common clean-clean scenario in which at most one candidate is a true match (Gemmell et al., 2011). We cast multi-candidate EM for each anchor as a cost-aware sequential decision problem and learn a policy that chooses which action to apply next, balancing both expected matching quality and inference cost. All hyperparameters introduced in this section, including ρ, λ, η, σ, γ, and top-k, are specified with their values in Appendix A.

3.1 LLM-Based Operators and Abstract Cost

CaRL-EM uses four high-level actions: three LLM-based operators (MATCH, COMPARE, SELECT) and a terminal action (DECIDE), as shown in Figure 2. The LLM-based operators are implemented with dedicated prompting templates in Appendix B.

MATCH. Given an anchor candidate pair (a, c_i), a MATCH call asks a lightweight matcher, Flan-T5-xl (Raffel et al., 2020) is used for MATCH and COMPARE in our experiment, to output YES/NO. We use the model probability of emitting YES to update the internal confidence score.

COMPARE. A COMPARE call presents a with two candidates (c_i, c_j) and asks which candidate is more likely to match a; the resulting preference is translated into a small update of their respective scores s_i and s_j, sharpening the score distribution.

SELECT. A SELECT call presents a and a shortlist $\tilde{C}(a)$ of top-k candidates ranked by the current scores. It prompts the LLM to perform listwise reasoning to identify the most likely match within this group or output None. The selected candidate receives a score boost, while others in the shortlist are penalized. This listwise feedback refines the global ranking without making a final decision.

DECIDE. DECIDE is the only terminal operator. The action set includes $\{\text{DECIDE}(i)\}_{i=1}^{n} \cup \{\text{DECIDE}(\text{None})\}$. The policy directly chooses a candidate or None without a specific thresholding.

Operators vary significantly in complexity and input scope, often necessitating different model capacities. Local operations like MATCH and COMPARE are computationally lighter and can be handled by smaller models, whereas the listwise SELECT operator requires reasoning over a larger context of multiple candidates, often demanding a larger, more capable model (Wang et al., 2025). This disparity in both model size and prompt length leads to distinct inference costs.

We capture this difference using a two level abstract cost model. Each operator is assigned a cost label in $\{\text{low}, \text{high}\}$, which we normalize to numerical values, e.g., κ_ℓ for low-cost local MATCH/COMPARE calls and $\kappa_h = \rho\kappa_\ell$ for high-cost shortlist SELECT calls. We set ρ=2.5 by default. Increasing it to ρ=5 yields nearly the same $F1_{\text{macro}}$, but increases MATCH/COMPARE usage and leads to higher cross-benchmark variance; more details are provided in Appendix C.

These numbers are not tied to any particular pricing scheme and are intended to reflect relative expense in terms of tokens and API calls. This abstraction decouples the learned policy from a specific model. By training on relative cost tiers, CaRL-EM allows users to swap different underlying LLMs for the operator at inference time without retraining the controller. Given a sequence of actions $\alpha_{1:T}$ for an anchor, the total abstract cost is

$$\mathcal{C}(\alpha_{1:T}) = \sum_{t=1}^{T} \text{cost}(\alpha_t),$$

and this cost enters the RL objective as a penalty, encouraging the policy to use cheap operators whenever they suffice.

3.2 MDP Formulation

Actions change the candidate state and incur cost, so decisions are sequential, including when to stop with DECIDE. We therefore model the decision process for each anchor as an episodic Markov decision process (MDP) (Sutton et al., 1998). At each step, an action invokes an operator that updates the state, and we learn a policy π to maximize the expected return $\mathbb{E}_\pi\left[\sum_{t=1}^{T} r_t\right]$ until DECIDE or $T_{\max}$.

State. At step t, the policy observes a state vector $\mathbf{x}_t$ that captures the current belief state, resource consumption, and interaction history. Formally, let N be the maximum number of candidates and H be the length of the action history window. The state vector is constructed as the concatenation of several feature groups:

$$\mathbf{x}_t = [\mathbf{v}_{\text{score}} \oplus \mathbf{v}_{\text{mask}} \oplus \mathbf{v}_{\text{freq}} \oplus \mathbf{v}_{\text{global}} \oplus \mathbf{v}_{\text{hist}} (\oplus \mathbf{v}_{\text{emb}})].$$

$\mathbf{v}_{\text{score}}$ contains the current confidence scores, $\mathbf{v}_{\text{mask}}$ indicates whether each candidate is still eligible to be processed by a MATCH operator (e.g., has not exceeded a per-candidate call limit), and $\mathbf{v}_{\text{freq}}$ tracks the frequency of each candidate in COMPARE calls. The global context $\mathbf{v}_{\text{global}}$ consists of the accumulated abstract cost, the normalized time step $t/T_{\max}$, and the current maximum candidate score. To detect loops or repetitive patterns, $\mathbf{v}_{\text{hist}}$ encodes the last H high level actions as a one-hot vector. Optionally, dense semantic embeddings of the anchor and the current top-scoring candidate ($\mathbf{v}_{\text{emb}}$) are appended to provide grounding. During training, we add small Gaussian noise σ into $\mathbf{x}_t$ to enhance policy robustness.

Action. At step t, the controller chooses a discrete action α_t that specifies an operator and its operands. Actions include MATCH(i) for inspecting candidate i, COMPARE(j) for evaluating a selected champion–challenger pair, SELECT(k) for listwise selection over the current top-k candidates, and terminal DECIDE(i) / DECIDE(None). The episode ends when DECIDE is chosen or when $T_{\max}$ is reached. Executing α_t invokes the corresponding LLM operator, updates candidate scores and usage statistics, and accumulates abstract cost. The full equations of candidate score updating are provided in Appendix D.

Reward. We design the reward to favor correct final decisions while keeping the cost low. Let $s_i^{(t)}$ be the score of candidate i at step t, $y_i \in \{0,1\}$ its label, and S_t the active candidates. We define a global margin Φ_t as the score gap between the best true match and the best non-match; if no match exists, it is the negative best score:

$$\Phi_t = \begin{cases} \max\limits_{i:y_i=1} s_i^{(t)} - \max\limits_{j:y_j=0} s_j^{(t)}, & \text{if } \exists i \in S_t, y_i = 1, \\ -\max\limits_{j\in S_t} s_j^{(t)}, & \text{otherwise,} \end{cases}$$

At a terminal step T, the agent outputs $\hat{y}_T \in S_T \cup \{\varnothing\}$ and receives a correctness reward:

$$R_{\text{term}}(\hat{y}_T, y) = \begin{cases} R_{\text{sel,cor}}, & \hat{y}_T = i, y_i = 1, \\ R_{\text{sel,wro}}, & \hat{y}_T = i, y_i = 0, \\ R_{\text{none,cor}}, & \hat{y}_T = \varnothing, \forall i, y_i = 0, \\ R_{\text{none,wro}}, & \hat{y}_T = \varnothing, \exists i, y_i = 1. \end{cases}$$

The total terminal reward is formulated as:

$$r_T = R_{\text{term}} + r_T^{\text{eff}} - \lambda\,\text{cost}(\alpha_T) - \beta\,\text{early}_T + \epsilon_T,$$

where r_T^{eff} rewards stopping before the deadline $T_{\max}$, $\text{cost}(\alpha_t)$ reflects the abstract cost, and early_T penalizes premature decisions with low confidence.

For non-terminal steps $t < T$, we use shaping to guide the agent (Ng et al., 1999; Wiewiora, 2003):

$$r_t = -\lambda\,\text{cost}(\alpha_t) + \eta(\gamma\,\Phi_{t+1} - \Phi_t) + \eta_k\,\mathbb{I}[\alpha_t = \text{SELECT}](\gamma\,\Phi_{t+1}^{(k)} - \Phi_t^{(k)}) + \epsilon_t. \quad (301)$$

3.3 Policy, Training, and Inference

CaRL-EM uses a lightweight policy network $\pi_\theta(\alpha_t \mid \mathbf{x}_t)$, implemented as a 3-layer MLP (768–384–192), which maps the state vector $\mathbf{x}_t$ to a distribution over high level actions. We train π_θ with Proximal Policy Optimization (PPO) and a learned value baseline (Schulman et al., 2017).

At test time, the controller starts from the initial state and iteratively selects actions until it chooses DECIDE or reaches the step limit $T_{\max}$. Because the controller interacts only with abstract operators and their cost labels, and is decoupled from the underlying LLMs that implement them, stronger LLMs can be plugged in at inference time without retraining the policy, as long as the operator interfaces and relative cost levels are preserved. This allows CaRL-EM to benefit from future improvements in LLM capabilities while retaining the learned decision strategy.

4 Experiments

LLMs used in our experiments. We evaluate CaRL-EM with a diverse set of LLM backends, covering both proprietary commercial APIs (OpenAI, 2024) and open weight models (Chung et al., 2024; Grattafiori et al., 2024; Agarwal et al., 2025; ERNIE Team, 2025; Team et al., 2025; Yang et al., 2025); more detail are in Table 1. For open weight models, we report a price range collected from several major inference providers[1].

Datasets and Zero-shot Transfer Setting. We train our approach on Abt-buy (AB) and evaluate it on standard entity resolution benchmarks extensively used in prior literature (Mudgal et al., 2018). These datasets cover diverse domains, including e-commerce, academic citations, movies, and restaurants. Following the experimental protocol established in COMEM (Wang et al., 2025),

[1]Prices are taken from providers: DeepInfra https://deepinfra.com/; Fireworks AI https://fireworks.ai/; Together.ai https://www.together.ai/; Novita https://novita.ai/; Groq https://groq.com/

Model Name	Size	CoT	Cost ($/1M tokens)	
			Input	Output
Flan-T5-xl	3B	–	0.10	0.10
Llama-3.1	8B	–	$0.03_{\sim 0.22}$	$0.03_{\sim 0.22}$
GPT-oss	20B	✓	$0.05_{\sim 0.10}$	$0.20_{\sim 0.50}$
ERNIE 4.5	21B	✓	0.07	0.28
Gemma 3	27B	–	0.1	0.2
Qwen3	30B	✓	$0.20_{\sim 0.89}$	$0.20_{\sim 7.90}$
GPT-4o mini	–	–	0.15	0.60

Table 1: Model specifications and costs. CoT: Chain-of-Thought support. For open weight models, we report the price range from multiple providers.

we employ a blocking stage using Sparkly to retrieve the top-10 most likely candidates from the target table for each anchor record. Blocking recall@10 is high across all datasets, ranging from 94.89%-to 99.96%, so remaining errors mainly reflect the quality of decisions. The final evaluation set contains 400 anchors per dataset, including 300 anchors with one true match, and the rest have no true match. This setup shifts the task from isolated pairwise classification to a realistic *multi-candidate selection* problem.

To evaluate the generalization capability of CaRL-EM, we use a zero-shot transfer protocol. CaRL-EM is trained only once on the AB dataset and evaluates it on the other seven benchmarks without any fine-tuning or adaptation. Since AB comes from a different domain and source than the test datasets, the controller cannot learn target-domain knowledge during training, and there is no data leakage. In contrast, for supervised baselines, like DITTO (Li et al., 2020), we sample an additional 5,000 labeled pairs from each target dataset to train domain-specific models. The specific datasets are: Abt-buy (AB), Amazon-Google (AG), DBLP-ACM (DA), DBLP-Scholar (DS), IMDb-TMDb (IM), IMDb-TVDb (IT), TMDb-TVDb (TT), and Walmart-Amazon (WA).

Hardware platforms and their price. All experiments involving local computation are conducted on NVIDIA H100 GPUs. To ensure a realistic and fair economic comparison, we standardize the GPU compute cost at $5.98 per hour. This rate is derived by averaging the on-demand pricing for H100 instances across four major cloud service providers; more details are provided in Appendix E.

Evaluation Metrics. Traditional pairwise F1 scores compute performance over isolated pairs, ignoring the mutual exclusivity often required in real-world applications, such as an anchor has at most one valid match (Christen, 2012). Given that our system makes a holistic decision over the candidate set (a, C), we employ instance level metrics rather than pair level ones. This protocol is strictly more demanding: a "Success" is counted only when the exact ground-truth is selected from K candidates; selecting a wrong candidate counts as both a False Positive and a False Negative. Notably, under the assumption of single-match validity, this instance level protocol aligns mathematically with the standard pairwise F1 score. Accordingly, we report $\mathbf{F1_{id}}$ to measure selection accuracy on anchors with valid matches, and $\mathbf{F1_{none}}$ to evaluate rejection sensitivity on those without. We define $\mathbf{F1_{macro}}$ as their unweighted average to ensure a balanced assessment that penalizes both selecting when none exists and misses.

Cost Calculation. We perform a comprehensive economic analysis covering both computational and API costs. For LLM-based components, costs are derived from token usage based on the standardized API pricing across multiple platforms, as shown in Table 1. For the training of CaRL-EM and the full lifecycle of the supervised baseline DITTO, we incorporate GPU compute costs standardized at $5.98/hour. During the inference phase, CaRL-EM's policy network is implemented as a lightweight MLP. Its computational overhead on a commodity CPU is negligible compared to the network latency of LLM API calls. Therefore, we assume zero hardware cost for CaRL-EM's inference, as it can be efficiently served without a GPU. For CaRL-EM, we distinguish deployment-time inference cost from one-time offline training cost. The latter is incurred once; we report it transparently in Appendix F.

5 Results

5.1 Main Results

Table 2 reports instance level $F1_{\text{macro}}$ and cost on 7 datasets. DITTO is trained in-domain, so it is trained separately on each target dataset. In contrast, CaRL-EM is trained only once on the AB dataset and then used zero-shot on the other datasets without any additional fine-tuning. Overall, CaRL-EM keeps $F1_{\text{macro}}$ competitive while using less inference cost, which leads to a higher efficiency score under our metric.

Comparison with the supervised baseline. DITTO is an in-domain supervised model, so it is trained separately on each dataset. CaRL-EM is trained once on AB and then transferred to the

Method	Training	AG	DA	DS	IM	IT	TT	WA	Avg.	Cost ($)	Eff. Score
Reference: Supervised (In-domain)											
DITTO (Li et al., 2020)	per-dataset	63.3	96.8	88.4	93.9	89.8	87.0	79.8	85.64	2.22	73.2
Zero-shot / Transfer Settings											
Matching (Peeters et al., 2025)	none	29.18	70.88	71.27	43.91	34.15	35.44	26.09	46.66	2.19	40.2
Comparing	none	47.35	**85.04**	**77.89**	61.66	33.60	46.29	43.71	58.08	0.54	134.4
Selecting	none	42.55	66.90	59.75	**92.20**	**84.00**	84.25	63.70	69.94	0.15	499.7
COMEM (Wang et al., 2025)	none	**60.01**	56.94	69.54	87.32	75.46	84.54	**78.76**	74.25	0.59	160.0
CaRL-EM (Ours)	**AB-only**	54.65	78.50	77.75	88.70	82.65	**85.80**	64.60	**76.09**	**0.13**	**623.7**

Table 2: $F1_{\text{macro}}$ and inference cost on 7 benchmarks. DITTO is a supervised reference trained in-domain, with a separate model for each dataset. COMEM is the strongest LLM pipeline baseline in our comparison. We focus on methods under the zero-shot and transfer settings. Eff. Score is a log-efficiency metric, defined as $F1_{\text{macro}}/\ln(1+\text{Cost})$, to capture the quality cost trade-off.

other 7 datasets with no extra tuning. Under this setting, CaRL-EM achieves an average $F1_{\text{macro}}$ of 76.09, while DITTO reaches 85.64. This means CaRL-EM scores about 89% of DITTO. However, the cost of CaRL-EM is 5.9% of DITTO's cost. This shows that the CaRL-EM can reuse one policy across datasets and keep the cost low. As we discuss in §5.3, DITTO does not transfer well across domains without retraining.

Comparison with LLM-based methods. Among zero-shot methods, CaRL-EM achieves the best average $F1_{\text{macro}}$. It also outperforms the strongest hand-crafted baseline, COMEM. At the same time, it uses less cost, with about a 4.5× reduction. Intuitively, a fixed pipeline is more likely to pick a candidate even when the evidence is weak, which leads to false positives. In contrast, CaRL-EM first applies the global SELECT operator and then uses MATCH and COMPARE for local checks. When the information is not enough, it is also more likely to stop with a no-match decision. We further support this point with $F1_{\text{none}}$ in Table 3.

5.2 Cost-Efficiency Analysis

Cost quality trade-off. Figure 3 compares methods in the dataset level total cost and $F1_{\text{macro}}$.[2] In the figure, CaRL-EM points are mostly in the high-quality, low-cost region. Without retraining the controller, swapping in a stronger backend LLM usually improves $F1_{\text{macro}}$, but it also increases cost. Under the same LLM, CaRL-EM (GPT-4O MINI) achieves a higher $F1_{\text{macro}}$ than the state-of-the-art cost-efficient baseline COMEM, while using only 22% of its average total cost. Overall, CaRL-EM forms a new Pareto frontier (Deb et al., 2002). This shows that it offers the best quality cost trade-off among the methods we evaluate. The gain mainly comes from the controller's adaptive decisions, rather than any single LLM.

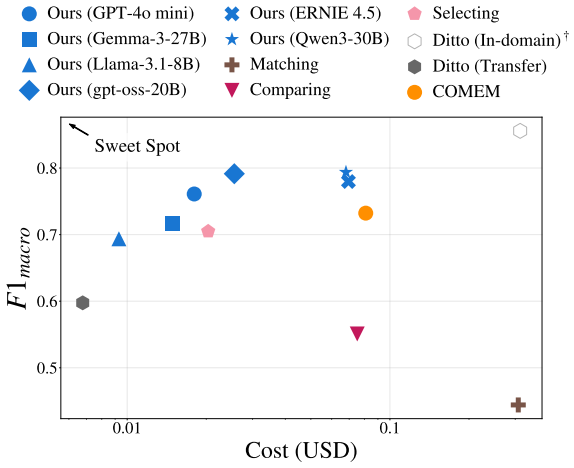

Figure 3: Pareto plot: total cost to process the 7 test datasets and mean $F1_{\text{macro}}$. Points closer to the top-left indicate a better quality cost trade-off. † DITTO (In-domain) is fine-tuned separately for each dataset; its training and cost accounting also differ from other methods, so we report it only as a reference.

5.3 Transfer and Robustness to LLM Backends

Zero-shot transfer. Table 3 compares CaRL-EM with COMEM and a transfer baseline based on DITTO. DITTO is trained on AB and then applied to the other datasets without retraining, so its $F1_{\text{macro}}$ drops from 85.64 to 59.75. In contrast, CaRL-EM is also not retrained, yet it still achieves a high $F1_{\text{macro}}$. This suggests that the learned decision strategy captures potential patterns in multi-candidate EM that transfer across datasets.

Swapping LLM backends. The same controller can also work with different LLM backends. We test this by swapping the backend models at inference time, without retraining the controller. As shown in Table 3, stronger backends usually im-

[2] We report the cost of inference for all zero-shot methods in this plot. CaRL-EM's one-time offline cost is reported separately in Appendix F.

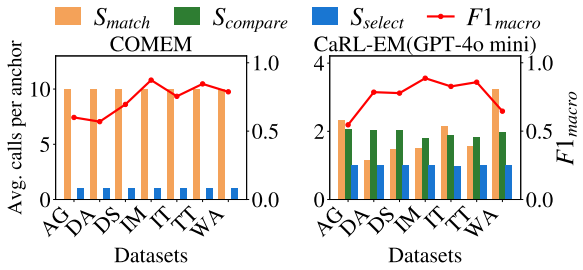

Figure 4: Average number of operator calls per anchor on each dataset.

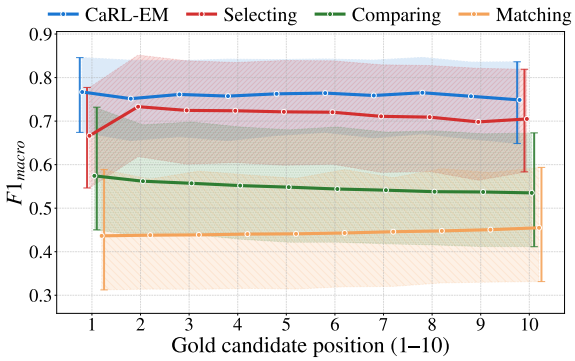

Figure 5: $F1_{\text{macro}}$ as a function of the gold candidate's initial position in the list. Lines show the mean performance at each position. Shaded regions indicate variation across datasets.

prove $F1_{\text{macro}}$, but they also increase cost. The controller mainly adjusts how often it uses MATCH and COMPARE to balance quality and cost. This pattern matches our design goal: the policy learns the decision sequence, rather than overfitting to one dataset or one specific LLM.

5.4 Controller Behavior and Position Bias

We inspect the learned policy to understand how it manages LLM operators and how this affects robustness to position bias in long candidate lists (Liu et al., 2024).

Operator usage patterns. Figure 4 shows the average number of operator calls per anchor. Compared with COMEM, CaRL-EM uses fewer MATCH and COMPARE calls. At the same time, the amount of tool usage varies across domains and datasets, showing that the learned policy does not follow a fixed call pattern. We also observe that on relatively harder datasets such as AG and WA, CaRL-EM tends to run more steps before making a final decision. In contrast, COMEM follows a fixed pipeline and cannot adjust its strategy across instances or datasets.

Mitigating long-context position bias. We run a controlled test to isolate position effects in the 7 test datasets. For each anchor, we force the gold candidate to appear at a fixed position (0–9) in the initial list. We keep the gold position fixed and shuffle the other candidates. We repeat this process 10 times for each position and report the average result.

As shown in Figure 5, the Selecting baseline is more sensitive to where the gold candidate appears in the list. It performs notably worse when the gold candidate is placed at position 1, and it still varies across other positions. The Matching and Matching baselines are flatter, but Comparing declines slightly, while Matching increases slightly. The shaded bands indicate gaps across datasets. This gap is larger for Matching and Comparing, which suggests weaker stability across domains. In contrast, CaRL-EM stays more stable across positions and shows the smallest cross-dataset variation. A key reason is that CaRL-EM does not rely on a single long listwise call. It can flexibly combine MATCH/COMPARE and SELECT, and it can perform mutual verification between various operators, which reduces dependence on the initial order.

5.5 Qualitative Analysis: Policy Behavior

To understand how CaRL-EM achieves efficiency, we visualize the decision trajectory on two cases from the AG dataset, as shown in Figure 6.

Case 1 (easy): From the initial confidence scores, the gold candidate (c2) has a clear advantage. The policy first calls SELECT on the top-4 list. It then uses two COMPARE calls as quick checks. It finally decides on c2. This takes only a few steps and avoids checking all candidates one by one; in this case, the policy does not call the pairwise MATCH operator.

Case 2 (Hard): In this case, several candidates start with similar scores, and the gold candidate (c0) doesn't have the highest confidence score. The policy again calls SELECT on the top-4 list, but the first result is c4. It then calls MATCH for local verification and finds that it conflicts with the result from the previous step. Therefore, it runs multiple MATCH calls and gradually improves the confidence of c0. This shows that the policy spends more local checks on hard cases, but stops early on easy ones, which allocates the budget based on instance difficulty. This adaptive behavior demonstrates that CaRL-EM acts as a "System 2" thinker, allocating computational resources dynamically based on instance difficulty (Kahneman, 2011).

Backend	$F1_{\text{id}}$	$F1_{\text{none}}$	$F1_{\text{macro}}$	Avg. S_{match}	Avg. S_{compare}	Avg. S_{select}	Cost
DITTO†	73.52	45.99	59.75	–	–	–	**0.05**
COMEM	**85.66**	60.78	73.22	10.00	–	1.00	0.59
CaRL-EM$_L$ (Llama-3.1)	81.23	57.49	69.36	**1.42**	2.03	**0.99**	$0.07_{(\sim 0.20)}$
CaRL-EM$_E$ (Gemma3)	82.86	60.56	71.70	1.69	2.02	**0.99**	0.11
CaRL-EM (GPT-4o mini)‡	83.99	68.20	76.09	1.92	1.95	**0.99**	0.13
CaRL-EM$_G$ (GPT-oss)	85.41	72.91	79.14	2.03	1.91	**0.99**	$0.18_{(\sim 0.34)}$
CaRL-EM$_E$ (ERNIE 4.5)	83.77	72.17	77.97	2.47	**1.89**	**0.99**	0.49
CaRL-EM$_Q$ (Qwen3)	85.41	**73.40**	**79.37**	2.18	1.90	**0.99**	$0.48_{(\sim 1.57)}$

Table 3: Transfer and backend robustness on the seven target datasets. We report $F1_{\text{id}}$, $F1_{\text{none}}$, and $F1_{\text{macro}}$, as well as the average number of operator calls per episode (S_{match}, S_{compare}, S_{select}). † DITTO is trained on AB and applied to the other datasets without retraining. ‡ CaRL-EM is trained once on AB with GPT-4o mini, and we swap the backend LLMs at test time without retraining the controller.

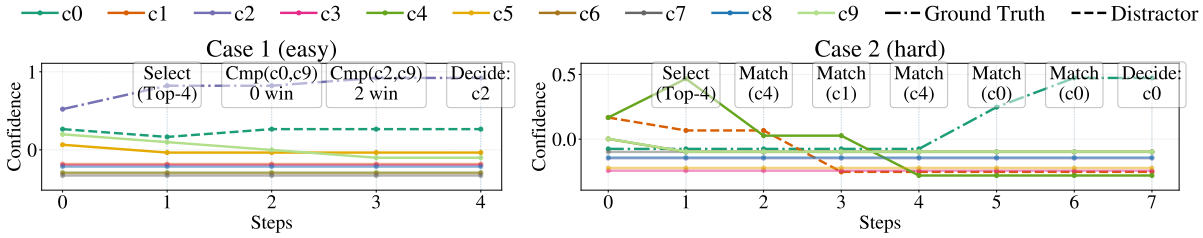

Figure 6: Two cases from AG processed by CaRL-EM. Colors denote different candidates. Lines show how each candidate's confidence changes with each action. Initial confidence comes from a similarity score between the anchor and the candidates.

5.6 Ablation and Design Choices

We run ablations to understand which components matter. We retrain each variant on AB and report results averaged over the other seven datasets. Table 4 summarizes accuracy, stability (Std.), and normalized inference cost.

Cost-aware reward. Removing the cost term keeps $F1_{\text{macro}}$ almost unchanged, but the cost increases by 15%. This shows that the cost term is needed to learn a more efficient policy.

Potential-based shaping. Removing the global potential term Φ_t reduces the $F1_{\text{id}}$ and $F1_{\text{macro}}$ value and makes performance unstable. This shows that the shaping signal helps the policy separate true and false candidates during the episode.

Operators. Each operator plays a different role. Without MATCH, the model is cheaper but its accuracy drops, especially on none cases. Without SELECT, the model is the cheapest and $F1_{\text{none}}$ is high, but $F1_{\text{id}}$ drops and overall performance is lower. Without COMPARE, performance also drops, which suggests that pairwise checks help when top candidates are close.

Variant	$F1_{\text{id}}$	$F1_{\text{none}}$	$F1_{\text{macro}}$	Std.	Cost
Full (ours)	**83.99**	68.20	**76.09**	**12.24**	0.13
w/o cost	82.96	69.16	76.07	13.05	0.15
w/o Φ_t	83.03	68.89	75.93	14.56	0.13
w/o Match	81.91	49.86	65.87	12.59	0.10
w/o Compare	80.04	67.31	73.69	15.04	0.13
w/o Select	75.13	**71.36**	73.26	20.60	**0.08**

Table 4: Ablation study of CaRL-EM variants. We report instance level $F1_{\text{id}}$, $F1_{\text{none}}$, macro-average $F1_{\text{macro}}$, the standard deviation across benchmarks, and inference cost, all averaged over the 7 benchmarks.

6 Conclusion

We study LLM-based EM in the practical blocked setting, where each anchor comes with a small candidate set and cost quickly becomes the bottleneck at scale. We propose CaRL-EM, a reinforcement learning controller that treats EM as a sequential decision process and decides when to use MATCH, COMPARE, SELECT, or DECIDE, while also choosing between cheaper and stronger model capacities. To our knowledge, this is the first work that casts LLM-driven EM as a cost-aware sequential RL problem.

Across seven benchmarks under zero-shot transfer, CaRL-EM learns to spend less on easy cases and check more on hard ones, and it yields a better quality cost trade-off than strong LLM baselines and hand-designed pipelines. More broadly, our formulation offers a way to think about trading off decision strategy performance and cost through a learned controller, rather than a fixed pipeline.

Limitations

Our study focuses on the clean-clean setting where an anchor has at most one true match. While this assumption fits many standard benchmarks and blocking-based pipelines, it does not cover settings with multiple valid matches, one-to-many links, or noisy/duplicate-heavy tables. Extending CaRL-EM to those cases would likely require changes to both the state (e.g., tracking multiple plausible matches) and the stopping/decision rule.

In addition, we mainly consider small candidate pools (top-10 in our protocol). When candidate sets become much larger, a controller that directly reasons over the whole list may become less effective, and a different design (e.g., multi-stage pruning, hierarchical control, or tighter coupling with retrieval) may be needed to keep both cost and decision quality under control.

Finally, CaRL-EM is trained with a coarse two level abstract cost model. This makes backend swapping simple and keeps the policy less tied to a specific pricing scheme, but it does not capture the full complexity of real deployment costs, such as prompt-length differences, token-based billing, latency constraints, batching effects, and provider-specific pricing. As a result, the learned behavior may not be cost-optimal under a different cost surface, and deployments may require recalibrating the cost function (or retraining) to match the target environment.

Ethical Considerations

We do not collect new data. Our experiments use public entity-matching benchmarks and we do not release any additional data. The main risk is incorrect matches, which can lead to wrong merges and downstream errors. High-stakes use should include auditing and human review. We discourage using entity matching to link personal identities across datasets and emphasize legal and ethical compliance.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Nils Barlaug and Jon Atle Gulla. 2021. Neural networks for entity matching: A survey. *ACM Trans. Knowl. Discov. Data*, 15(3).

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*.

Peter Christen. 2012. Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2017. Deeper–deep entity resolution. *arXiv preprint arXiv:1710.00597*.

Baidu ERNIE Team. 2025. Ernie 4.5 technical report. https://yiyan.baidu.com/blog/publication/ERNIE_Technical_Report.pdf.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.

Jim Gemmell, Benjamin IP Rubinstein, and Ashok K Chandra. 2011. Improving entity resolution with global constraints. *arXiv preprint arXiv:1108.6016*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daniel Kahneman. 2011. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York.

Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: toward building entity matching management systems. *Proc. VLDB Endow.*, 9(12):1197–1208.

Huahang Li, Longyu Feng, Shuangyin Li, Fei Hao, Chen Jason Zhang, and Yuanfeng Song. 2024. On leveraging large language models for enhancing entity resolution: a cost-efficient approach. *arXiv preprint arXiv:2401.03426*.

Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

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 2018 international conference on management of data*, pages 19–34.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 278–287, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2025. Routellm: Learning to route llms with preference data, 2024. *URL https://arxiv. org/abs/2406.18665*, 4.

OpenAI. 2024. Gpt-4o mini: advancing costefficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.*, 53(2).

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):1507–1519.

Ralph Peeters and Christian Bizer. 2021. Dual-objective fine-tuning of bert for entity matching. *Proc. VLDB Endow.*, 14(10):1913–1921.

Ralph Peeters and Christian Bizer. 2023. Using chatgpt for entity matching. In *European Conference on Advances in Databases and Information Systems*, pages 221–230. Springer.

Ralph Peeters, Aaron Steiner, and Christian Bizer. 2025. Entity matching using large language models. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 529–541. OpenProceedings.org.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Kashif Shah, Selcuk Kopru, and Jean-David Ruvini. 2018. Neural network based extreme classification and similarity models for product matching. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 8–15, New Orleans - Louisiana. Association for Computational Linguistics.

Nima Shahbazi, Nikola Danevski, Fatemeh Nargesian, Abolfazl Asudeh, and Divesh Srivastava. 2023. Through the fairness lens: Experimental analysis and evaluation of entity matching. *Proc. VLDB Endow.*, 16(11):3279–3292.

Richard S Sutton, Andrew G Barto, and 1 others. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

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):2459–2472.

Janusz Tracz, Piotr Iwo Wójcik, Kalina Jasinska-Kobus, Riccardo Belluzzo, Robert Mroczkowski, and Ireneusz Gawlik. 2020. BERT-based similarity learning for product matching. In *Proceedings of Workshop on Natural Language Processing in E-Commerce*, pages 66–75, Barcelona, Spain. Association for Computational Linguistics.

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 31st International Conference on Computational Linguistics*, pages 96–109.

Eric Wiewiora. 2003. Potential-based shaping and q-value initialization are equivalent. *J. Artif. Int. Res.*, 19(1):205–208.

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*.

Xuechen Zhang, Zijian Huang, Ege Onur Taga, Carlee Joe-Wong, Samet Oymak, and Jiasi Chen. 2024. Treacle: Thrifty reasoning via context-aware llm and prompt selection. *CoRR*.

Symbol	Meaning	Value
N	max #candidates	10
H	action history length	10
$T_{\max}$	max decision steps per anchor	18
ρ	cost ratio κ_h/κ_ℓ	2.5
k	top-k shortlist size(s) for SELECT	$\{4\}$
λ	per-step cost penalty weight in reward	1.0
η	global potential shaping weight	0.3
η_k	local shaping weight for SELECT	0.5η
γ	shaping discount factor	0.99
σ	Gaussian noise	0.005

Table 5: Hyperparameter values for CaRL-EM .

A Hyperparameter Values

Table 5 lists the concrete hyperparameter values used in our experiments.

B Prompt Templates

B.1 Match

Output constraint: the first line must be exactly one token from `[YES]` or `[NO]`.

```
Decide if the two records refer to the SAME
↪ real-world entity.
Output EXACTLY ONE token on the first line: [YES]
↪ for same, [NO] for different.
[ANCHOR]
{anchor}
[CANDIDATE]
{candidate}
Answer:
```

B.2 Compare

Output constraint: the first line must be exactly one token from `[0]` or `[1]`.

```
Which candidate better matches the anchor?
Return EXACTLY ONE token: [0] for the first
↪ candidate, [1] for the second.
[ANCHOR]
{anchor}
[0]
{a}
[1]
{b}
Answer:
```

B.3 Select

Output constraint: the first line must be exactly one token from `[0..n-1]` or `[NONE]`.

```
Return only one of [0..n-1] or [NONE].
Select the single best-matching candidate for the
↪ anchor.
Answer with EXACTLY one token on the first line:
↪ [0..{n-1}] or [NONE].
Do NOT include any other words or punctuation.
[ANCHOR]
{anchor}
OPTIONS:
[0] {c0}
[1] {c1}
...
[n-1] {cn-1}
```

C Sensitivity to the Abstract Cost Ratio ρ

We test the robustness of the two-level abstract cost ratio ρ by changing it from ρ=2.5 to ρ=5, retraining CaRL-EM on Abt-Buy (AB) and evaluating zero-shot on the seven target benchmarks. As shown in Table 6, $F1_{\text{macro}}$ remains nearly unchanged. However, the learned policy makes slightly more MATCH/COMPARE calls and exhibits slightly higher variance across benchmarks.

D Confidence Update Rules

We maintain per-candidate internal confidence scores $c_i^{(t)} \in [-1, 1]$.

MATCH update. Let $s \in [0, 1]$ be the matcher probability of [YES]. We map it to $\tilde{s} = 2s - 1 \in [-1, 1]$ and update candidate i by exponential smoothing:

$$c_i \leftarrow (1 - \alpha)c_i + \alpha\tilde{s}, \tag{1}$$

where $\alpha = \alpha_{\text{hi}}$ if $|s - 0.5| > match\_margin$, otherwise $\alpha = \alpha_{\text{lo}}$.

COMPARE update. Given a compared pair (i, j), the comparator returns a winner and a confidence $q \in [0, 1]$. Let $\delta = q - 0.5$ and set $\gamma = \gamma_{\text{hi}}$ if $|\delta| > compare\_margin$, else $\gamma = \gamma_{\text{lo}}$. Then:

$$(c_i, c_j) \leftarrow \begin{cases} (c_i + \gamma|\delta|,\ c_j - \gamma|\delta|), & \text{if } i \text{ wins}, \\ (c_i - \gamma|\delta|,\ c_j + \gamma|\delta|), & \text{if } j \text{ wins}. \end{cases} \tag{2}$$

SELECT update. Let K_t be the current top-k shortlist. If the selector outputs a valid index $r \in K_t$:

$$c_r \leftarrow c_r + b^+, \quad \forall j \in K_t \setminus \{r\} : c_j \leftarrow c_j - b^-. \tag{3}$$

If the selector output is invalid (e.g., parsing failure), we penalize the shortlist:

$$\forall j \in K_t :\ c_j \leftarrow c_j - b^{\text{inv}}. \tag{4}$$

Finally, all scores are clipped to $[-1, 1]$.

E H100 pricing from different cloud service platforms

Table 7 lists the H100 GPU server quotes we collected from 4 common cloud service providers. According to this, we estimated the average price of H100 servers on the market.

ρ	$F1_{\text{id}}$	$F1_{\text{none}}$	$F1_{\text{macro}}$	**Avg.** S_{match}	**Avg.** S_{compare}	**Avg.** S_{select}	**Std.**	**Cost**
2.5	83.99	68.20	76.09	1.92	1.95	**0.99**	12	0.13
5	83.17	71.23	77.20	2.42	1.58	0.99	15	**0.13**

Table 6: Sensitivity to the abstract cost ratio ρ (mean over the seven target benchmarks, zero-shot transfer).

Provider	**Instance**	**GPUs per instance**	**USD / GPU*h**
AWS	p.548xlarge	8	≈ 3.93
Google	A3-highgpu-1g	1	≈ 3.00
Azure	Standard-NC40ads-H100-v5	1	≈ 6.98
Oracle	BM.GPU.H100.8	8	10.00

Table 7: H100 GPU server price 4 $/h .

F One-time Offline Cost for CaRL-EM

We report the token and GPU usage during training on AB, as shown in Table 8.

Item	**Time/h**	**Tokens**	**Cost ($)**
MATCH	-	335,592	0.04
COMPARE	-	1,694,532	0.18
SELECT	-	2,122,750	0.32
PPO training (H100)	2.75	-	16.45

Table 8: One-time offline cost on AB.