Decomposition vs Augmentation: Isolating the Impact of Multi-Agent Architectures in Entity Resolution

Zachary Zeng
zengz1@ufl.edu
University of Florida
Gainesville, Florida, USA

Abstract

Recent work reports that multi-agent large language model (LLM) systems outperform single-LLM pipelines on entity resolution (ER), but these systems typically bundle agent decomposition with retrieval-augmented generation or other external knowledge, conflating two distinct design choices. This paper isolates that question through a controlled three-way comparison on Abt-Buy among a Magellan random-forest baseline, a single `gpt-oss-120b` pipeline, and a multi-agent `gpt-oss-120b` pipeline whose syntactic and semantic agents share only local, computable tools (string similarity, embeddings, price math) under an orchestrator, strictly without external knowledge. Across five independent runs the two LLM pipelines reach near-identical F_1-scores (0.9065 ± 0.0039 vs. 0.8973 ± 0.0048), agree on 98.7% of pairs (Cohen's $\kappa = 0.942 \pm 0.005$), and are statistically indistinguishable in per-pair accuracy under a pooled McNemar's exact test ($p = 0.26$); yet the multi-agent pipeline consumes 8.7× more tokens and runs 6.6× longer. The two pipelines are not redundant, the multi-agent system trades recall for precision (precision 0.875 vs. 0.859, recall 0.920 vs. 0.960) by rejecting color and finish variants under tool-based evidence. This may point to agent decomposition without knowledge augmentation acts as a precision-recall knob rather than a strict performance increase, and that the F1-score gains reported for multi-agent ER in prior work should be attributed primarily to external knowledge rather than to decomposition itself.

CCS Concepts

• **Information systems** → **Information extraction**; • **Computing methodologies** → *Knowledge representation and reasoning*; Machine learning approaches; • **Applied computing**;

Keywords

Entity Resolution, Large Language Models, Multi-Agent Systems, Knowledge Augmentation, Retrieval-Augmented Generation, Agent Decomposition, Zero-shot Classification, Product Matching

ACM Reference Format:
Zachary Zeng. 2026. Decomposition vs Augmentation: Isolating the Impact of Multi-Agent Architectures in Entity Resolution. In *Proceedings of ACM SIGKDD Undergraduate Consortium (KDD-UC '26).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

KDD-UC '26, Jeju, Korea

ACM ISBN 978-1-4503-XXXX-X/2026/08
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Entity resolution (ER), or record linkage, is the task of identifying which records across one or more datasets refer to the same real-world entity [2]. It is a critical step in data cleaning whose precision directly bounds downstream analysis quality, with applications from census deduplication to academic publication disambiguation [3]. Large language models (LLMs) have recently emerged as a third option alongside rule-based systems and feature-engineered classifiers, leveraging pre-trained world knowledge to assess entity similarity through natural-language reasoning rather than labeled training data and hand-crafted features [11, 13]. This expressiveness, however, comes at a real cost: each pair requires an API call that incurs token-based charges and network latency, so the practical case for LLM-based ER hinges on whether the accuracy gains justify the overhead [7, 10].

A second wave of LLM-based ER work has moved beyond monolithic prompting toward multi-agent architectures, in which specialized agents analyze different aspects of a candidate pair and a coordinator produces the final verdict. These systems report meaningful accuracy gains over single-LLM baselines [1, 12], but they typically bundle agent decomposition with retrieval-augmented generation (RAG), self-annotated training data, or other forms of external knowledge. As a result, the reported gains conflate two distinct design choices: *decomposing* the reasoning across agents, and *augmenting* the agents with information beyond the pair itself. It is not clear whether the multi-agent architecture alone helps ER, or whether the accuracy gains are attributable to the augmentation component and would disappear if the agents were given only the pair and local tools.

This paper isolates that question with a controlled three-way comparison on the Abt-Buy product-catalog benchmark. I evaluated a Magellan random-forest baseline [6], a single `gpt-oss-120b` model baseline, and a multi-agent `gpt-oss-120b` pipeline under identical blocking, test pairs, and metrics. The multi-agent pipeline does not include retrieval and external knowledge. Instead, it features a syntactic agent equipped with string-similarity tools (edit distance, Jaccard, and BM25) and a semantic agent equipped with sentence-embedding cosine similarity and a price comparator. These two agents are coordinated by an orchestrator that merges their verdicts into a final decision. This design isolates the effect of decomposition, holding both the underlying model and available information fixed.

This paper makes three contributions. First, it provides a three-way benchmark between Magellan (Random Forest), a single-LLM

pipeline, and a multi-agent LLM pipeline on the Abt-Buy benchmark under identical conditions, with no RAG or external knowledge given to the LLM pipelines. Second, it reports a cost-efficiency evaluation that includes precision, recall, F1-score, token consumption, and end-to-end runtime averaged over five independent runs, exposing the cost-performance trade-off for practitioners. Third, it presents an error analysis of the 25 pairs on which the two LLM pipelines disagree on a representative run (and 115 pooled discordant pairs across all five runs), categorizing their false positives and false negatives to reveal complementary error profiles tied to the architectural differences.

The results show that the two LLM pipelines reach near-identical F1-score (single-LLM 0.9065 ± 0.0039, multi-agent 0.8973 ± 0.0048, Magellan 0.6066 ± 0.0144), agree on 98.7% of pairs (Cohen's κ = 0.942 ± 0.005), and are not statistically distinguishable in per-pair accuracy under a pooled McNemar's exact test ($p = 0.26$, $n = 115$ discordant pairs). Despite this near-equivalence, the multi-agent pipeline consumes 8.7× more tokens and runs 6.6× longer, while occupying a different point on the precision-recall curve: higher precision (0.875 vs. 0.859) at the cost of recall (0.920 vs. 0.960), trading aggressive matches on sparse listings for stricter rejection of color and finish variants.

2 Related Work

2.1 Entity Resolution Background

Entity resolution has been studied for decades under several names (record linkage, entity matching, deduplication), and modern end-to-end pipelines decompose the task into *blocking*, *matching*, and *clustering* [2, 3]; this paper fixes blocking across all pipelines and focuses on the matching stage. Early ER relied on Fellegi-Sunter-style probabilistic rules with hand-tuned similarity thresholds [5]; modern supervised approaches such as Magellan [6] (this paper's baseline) automatically generate a dense feature vector of attribute-level similarities (edit distance, Jaccard, Monge-Elkan, TF-IDF cosine) and train a standard classifier on labeled pairs. Deep-learning approaches like DITTO [8] further improved matching by fine-tuning a pre-trained language model on labeled pairs, but they still require task-specific labeled data and per-dataset fine-tuning, motivating the shift toward prompting-based methods.

2.2 LLM-Based Entity Resolution

LLM-based ER lets a pre-trained model decide whether two records describe the same entity directly from a natural-language prompt, without task-specific training [5]. Wang et al. [11] identified three prompting strategies: *match* (one pair at a time), *compare* (one record against many), and *select* (pick the best match from a set); this paper's single-LLM baseline uses the *match* strategy as the most direct comparison to traditional pair-classifiers. BoostER complements this design by using the LLM as a selective verifier on top of a cheaper base matcher, reducing the number of LLM calls per dataset [7], and prompt-engineering work has further explored single-attribute prompts and caching to contain token cost [10]. Importantly, recent cross-dataset evaluation shows that fine-tuned small models can rival frontier closed models on entity matching [13], which justifies the use of a medium-sized open model, `gpt-oss-120b`, in this study.

2.3 Multi-Agent LLM Systems and the Knowledge-Augmentation Confound

A new paradigm of LLM work uses multi-agent architectures, where specialized agents analyze different aspects of an input and an orchestrator coordinates their verdicts [4]. For ER, Althaf et al. [1] report gains over single-LLM baselines using a layered framework whose agents retrieve context from an indexed knowledge store via a RAG component, and CMAS [12] pairs decomposition with self-annotated pseudo-labels and retrieved few-shot demonstrations. In every reported gain, however, agent decomposition is coupled with an additional source of information beyond the input pair, whether retrieval over a knowledge store, self-annotated training data, in-context demonstrations, or external analyzers, so it is not possible to tell from the existing literature whether the improvements stem from decomposition or from the knowledge augmentation that accompanies it. This study targets that confound directly: agents have access only to local, computable tools over the pair itself, isolating what decomposition alone contributes to performance and efficiency.

3 Method

3.1 Data and Blocking

I use the Abt-Buy product dataset from HuggingFace[1]: 1,081 listings from Abt.com and 1,092 from Buy.com, paired into 9,575 labeled candidate pairs with 1,028 verified true matches. Each record carries a name, description, and price (with some missing values), and the dataset is presplit into train (5,743 pairs, 616 positive), validation, and test (each 1,916 pairs, 206 positive) splits, yielding a 10.8% positive-class rate on the test split used for all final metrics.

To reduce the $1{,}081 \times 1{,}092 = 1{,}180{,}452$-pair Cartesian product, I apply TF-IDF cosine-similarity blocking on product names with character 2- and 3-gram features and a similarity threshold of 0.2, retaining 16,629 candidate pairs that contain 100% of true matches. Blocking is shared across all three pipelines, so observed differences in the matching metrics are attributable to the matcher rather than the blocker.

3.2 Magellan Pipeline

The traditional ML baseline follows the Magellan approach [6], using a Random Forest classifier with manually engineered features. Because the original `py_entitymatching` library was incompatible with my Python 3.12 environment, I reimplemented the feature set based on Magellan's source code using `scikit-learn` and `jellyfish` to replicate the auto-generated feature behavior. Each candidate pair is encoded as 14 features: 8 string-similarity measures over names (Jaccard on 3-grams and word tokens, TF-IDF cosine, Monge-Elkan with Jaro-Winkler, raw and normalized Levenshtein, Needleman-Wunsch and Smith-Waterman alignment), 2 over descriptions (3-gram Jaccard and TF-IDF cosine), and 4 over prices (exact-match indicator, absolute-difference similarity, and two Levenshtein variants on price strings).

The Random Forest classifier from `scikit-learn` was trained on the predefined HuggingFace training split using the fourteen features above. Hyperparameters were tuned via `GridSearchCV` with

[1] https://huggingface.co/datasets/matchbench/Abt-Buy

a `PredefinedSplit` on the validation set, sweeping `n_estimators`, `max_depth`, `min_samples_split`, `max_features`, and `class_weight` (full grid in the released code). The classification threshold was also tuned on the validation set to maximize F1-score rather than using the default 0.5.

3.3 Single-LLM Pipeline

The single-LLM baseline utilizes the `gpt-oss-120b` model with a direct prompt-response mechanism. For each candidate pair, the model receives a system prompt instructing it to analyze product names, descriptions, and prices. It must respond with a JSON object containing a binary verdict (`MATCH` or `NO MATCH`), a confidence score in the range $[0.0, 1.0]$, and a brief rationale for its response. The system prompt provides explicit guidance on factors to consider: name variations (typos, abbreviations, word order), matching specifications (model numbers, capacities, etc.), compatible price ranges, and overlapping descriptions.

Pairs are processed sequentially with the temperature fixed at 0 to ensure reproducibility. The results of each LLM response are cached locally. This allows for post-hoc analysis of the model's reasoning and significantly reduces execution time and computational overhead for subsequent runs.

3.4 Multi-Agent LLM Pipeline

The multi-agent LLM architecture decomposes entity resolution into specialized lenses handled by two reviewer agents that are coordinated by a central orchestrator. All agents and the orchestrator use `gpt-oss-120b`, and each agent has a fixed tool-set.

The syntactic agent focuses on string-level similarity and is given three Model Context Protocol (MCP) tools that it is prompted to call before finalizing its verdict: a normalized Levenshtein edit distance over product names (returning 0.0 for identical strings and values approaching 1.0 for fully dissimilar ones), word-token Jaccard similarity over product names, and a symmetric BM25 relevance score normalized to $[0, 1]$.

The semantic agent focuses on deeper semantic analysis and is given two MCP tools that it is also instructed to call before deciding: a price comparator that returns the price ratio, absolute difference, and a similarity flag (ratio ≥ 0.8) over the two parsed prices, and an embedding cosine similarity computed between `all-MiniLM-L6-v2` sentence-transformer embeddings of the product names.

Tools are served via FastMCP over a `stdio` transport, running as a subprocess that communicates with the agents through standard input/output, which separates tool execution from LLM inference. The orchestrator receives the original product pair alongside both agents' structured responses (verdict, confidence, and reasoning) and is prompted to review and produce a final decision. Every component (agent or orchestrator) returns a structured JSON response of the form `{verdict, confidence, reasoning}`, and the orchestrator's verdict serves as the final binary match decision for each candidate pair. Figure 1 illustrates the complete system architecture.

3.5 Evaluation Pipeline

All three pipelines share a unified evaluation harness over the 206 positive pairs in the test split. Each pipeline emits a verdict, a confidence, and a parse-error flag (failed JSON parses are recorded rather than silently dropped); runtime is timed around the full prediction loop, and tokens are summed across all LLM calls per pair (zero for Magellan, since inference is local). Each LLM run uses an isolated response cache so cached results from earlier runs cannot short-circuit measurement, and Magellan's run-to-run variability comes solely from the Random Forest `random_state`. Means and standard deviations are aggregated across the five runs.

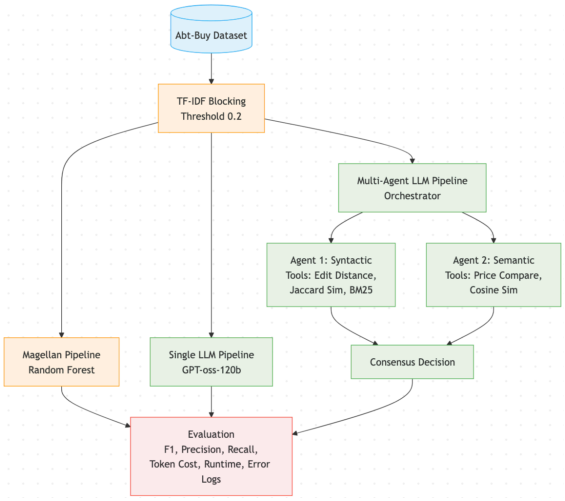

Figure 1: System architecture. All three pipelines share TF-IDF blocking. The multi-agent pipeline dispatches each pair to a syntactic and semantic agent equipped with MCP tools, after which an orchestrator makes the final decision.

4 Experiments

4.1 Evaluation Metrics

The primary metric is F1-score, which balances false positives and false negatives without privileging either precision or recall. For the LLM pipelines I additionally track total tokens consumed and end-to-end runtime, capturing the practical overhead of LLM-based ER relative to traditional ML. The multi-agent pipeline is compared on F1-score against both the Magellan and single-LLM baselines, on runtime against both, and on tokens against the single-LLM, to determine whether any performance delta justifies the increased computational cost.

To quantify agreement between the two LLM pipelines beyond chance, I compute Cohen's Kappa (κ) on each run and report the mean and standard deviation across runs ($\kappa > 0.8$ indicates near-perfect agreement). To assess whether the residual F1 gap reflects a genuine difference in per-pair accuracy or only run-to-run noise, I additionally apply a pooled McNemar's exact test across discordant pairs from all five runs. McNemar's test is appropriate because both pipelines classify the same pairs (paired observations) and conditions on the discordant cells of the 2×2 contingency table.

4.2 Experiment Setup

All experiments ran on an Apple M4 Pro (24 GB RAM, macOS) with Python 3.12 and uv for dependency management. LLM calls used `gpt-oss-120b` via the OpenAI Python SDK at temperature 0, issued sequentially for comparable runtime measurements. Each pipeline was evaluated on the same 1,916 test-split pairs across 5 independent runs (Magellan variability comes from the Random Forest `random_state`; LLM variability from residual API non-determinism at temperature 0), and I report the mean and standard deviation for all metrics.

4.3 Pipeline Performance Comparison

Table 1 presents the aggregate performance of all three pipelines averaged across five independent runs. The two LLM pipelines reach near-identical F1-score: the single-LLM baseline achieves 0.9065 ± 0.0039 and the multi-agent pipeline achieves 0.8973 ± 0.0048. A pooled McNemar's exact test over the 115 pairs across runs where the two pipelines disagree (Single-LLM correct in 64, Multi-Agent correct in 51) fails to reject the null hypothesis of equal classification accuracy ($p = 0.26$), so the per-pair classification quality of the two pipelines is not statistically distinguishable on this benchmark. A paired t-test on the five per-run F1-scores does detect a small but consistent advantage for the single-LLM pipeline ($\Delta F_1 = 0.0092 \pm 0.0069$, $t = 3.00$, $p = 0.04$); however, this gap is below the run-to-run standard deviation of either pipeline and reflects modest cross-run consistency rather than a substantive per-pair quality difference. Both LLM pipelines substantially outperform the Magellan Random Forest baseline ($F_1 = 0.6066 \pm 0.0144$, roughly 30 points lower); the per-error breakdown is in §4.5.

Table 1: Aggregate performance across five runs (mean ± std). Runtime and token counts are for the full 1,916-pair test split. The best result in each row is shown in bold. For cost metrics (runtime, tokens), lower is better.

Metric	Magellan	Single-LLM	Multi-Agent
Precision	0.6876 ± 0.0186	0.8585 ± 0.0058	**0.8754 ± 0.0075**
Recall	0.5427 ± 0.0147	**0.9602 ± 0.0022**	0.9204 ± 0.0106
F_1	0.6066 ± 0.0161	**0.9065 ± 0.0039**	0.8973 ± 0.0048
Runtime (s)	**10.4 ± 0.1**	2,538.9 ± 304.5	16,878.5 ± 255.0
Total tokens	**0**	1.26M ±560	10.96M ±22,270
Tokens / pair	**0**	656.1 ± 0.3	5,720.0 ± 11.6

In terms of cost and time, the Magellan pipeline is almost negligible compared to the LLM pipelines. Each run takes about 10 seconds with no token costs, since inference relies entirely on local feature computation. The two LLM pipelines take much longer to run due to API latency. The single-LLM baseline takes about 42 minutes, while the multi-agent pipeline takes about 281 minutes, which is approximately 6.6× longer. The multi-agent pipeline also consumes 8.7× more tokens per pair (5,720 vs. 656). For applications that prioritize matching quality over cost, the LLM pipelines are highly effective, but the multi-agent pipeline costs significantly more in time and tokens with no measurable accuracy gain, making the single-LLM pipeline the most attractive plug-and-play option.

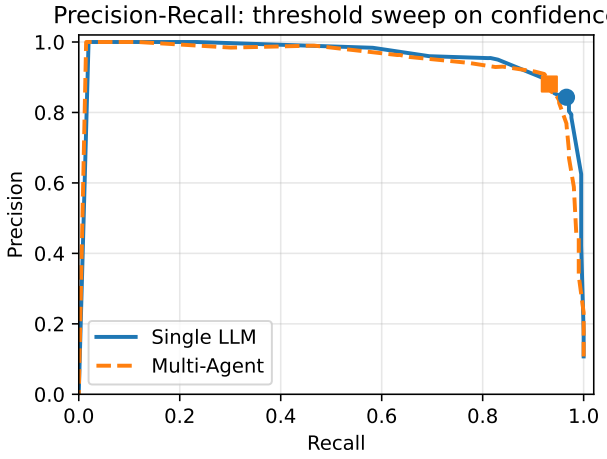

Figure 2: Precision–Recall curves for the two LLM pipelines on Run 1, obtained by sweeping the decision threshold $\tau \in [0, 1]$ over each pipeline's reported confidence. Threshold-optimized maximum F1-score is 0.916 (multi-agent, $\tau = 0.76$) vs. 0.909 (single-LLM, $\tau = 0.79$); both pipelines reach 0.906 at the default $\tau = 0.5$.

Despite this cost asymmetry, the two LLM pipelines are remarkably aligned in their individual decisions: Cohen's Kappa averages $\kappa = 0.942 \pm 0.005$ across the five runs, with 98.7% pair-level agreement, indicating that multi-agent decomposition reaches largely the same verdicts as the single-LLM at a higher cost.

4.4 Precision-Recall Trade-off

Although the two LLM pipelines are tied in F1-score, they occupy different points on the precision-recall curve, and the direction of the trade-off is informative. Across the five runs, the multi-agent pipeline achieves higher precision (0.875 ± 0.007 vs. 0.859 ± 0.005) at the cost of lower recall (0.920 ± 0.010 vs. 0.960 ± 0.002). In absolute terms on a single representative run (Run 1, which is also the run used for all subsequent error analysis), the single-LLM baseline produces 33 false positives and 8 false negatives, while the multi-agent pipeline produces 26 false positives and 14 false negatives. The multi-agent LLM is more conservative when declaring matching records; it does not find as many positives compared to the single-LLM, but as a result, the matches it does identify are more consistently correct. Sweeping the decision threshold over each pipeline's reported confidence (Figure 2) confirms that this rebalancing persists beyond the default operating point: at threshold-optimized cut-offs the multi-agent pipeline reaches its best $F_1 = 0.916$ at $\tau = 0.76$ while the single-LLM reaches $F_1 = 0.909$ at $\tau = 0.79$, a one-point gap that nevertheless leaves the two pipelines in the same neighborhood of the precision-recall plane.

4.5 Error Analysis

I categorized the false positives and false negatives of a representative run (Run 1) by reading each LLM's reasoning field; results

appear in Table 2. Two patterns emerge that are consistent with the architectural differences rather than statistical noise.

Table 2: False positive and false negative error categories for the two LLM pipelines on a single representative run. The lower (better) pipeline total is shown in bold; for error counts lower is better.

	Single-LLM	Multi-Agent
False positives		
Color / finish variant	13 (39%)	8 (31%)
Accessory confusion	8 (24%)	6 (23%)
Model / SKU variant	1 (3%)	3 (12%)
Generic vs. specific listing	1 (3%)	1 (4%)
Form factor	2 (6%)	1 (4%)
Other	8 (24%)	7 (27%)
Total FP	33	**26**
False negatives		
Sparse description	6 (75%)	11 (79%)
Price mismatch	1 (13%)	2 (14%)
Name mismatch	1 (13%)	1 (7%)
Total FN	**8**	14

4.5.1 Single-LLM Errors. The dominant single-LLM false positive occurs when products share a model family but differ on a discriminating attribute, most often color or finish (13 of 33 false positives, about 39%): for example, matching *Canon PowerShot SD1100 IS silver* to *Canon PowerShot SD1100 IS pink*, where the core model number matches but the suffix denotes a true color variant. The LLM appears to weigh the model-number signal heavily and rationalize the color difference as a minor variant. The single-LLM's false negatives are few (8 total) and concentrated on sparse listings (6 of 8) where one side has no description and there is little evidence beyond the name to confirm a match.

4.5.2 Multi-Agent LLM Errors. The multi-agent pipeline produces fewer false positives than the single-LLM (26 vs. 33), with the reduction concentrated exactly where the single-LLM overreached: color variants drop from 13 to 8, and the agents flag more accessory and model-variant cases as distinct. The syntactic agent's edit-distance and BM25 tools surface concrete token-level differences in the model suffixes (e.g., *sd1100is* vs. *sd1100is pink*) that the agent cites when rejecting the match. That same conservatism, however, hurts recall on sparse pairs: 11 of the multi-agent pipeline's 14 false negatives involve at least one product listing with an empty or near-empty description, where the only signal available is the short, partially overlapping name, which embeds to moderate cosine similarity and yields low BM25 scores. The single-LLM baseline, lacking any conflicting evidence from strict tools, matches more aggressively on the model number alone in these cases and incurs fewer false negatives as a result.

Per-agent analysis reinforces this asymmetry: the syntactic agent alone reaches $F_1 = 0.9040$ (within 0.002 of the full orchestrated pipeline), while the semantic agent alone reaches only $F_1 = 0.8454$, with 31 false negatives concentrated on the same sparse-description pairs.

4.5.3 Self-Reported Confidence Implications. The LLM pipelines also return a confidence score with each verdict. The single-LLM's mean confidence drops sharply on its mistakes (TP 0.924, FN 0.764, gap 0.160), so it "knows" when it is uncertain even when it incorrectly declares `NO MATCH`. The multi-agent pipeline shows the same pattern but with a much smaller TP–FN gap of 0.087. Examining the individual agents reveals a wider gap that the orchestrator obscures: the syntactic and semantic agents report mean false-negative confidence of 0.731 and 0.704 (well below their true-positive means of 0.823 and 0.865), but the orchestrator's synthesized false-negative confidence rises to 0.812. Beyond its token overhead, the orchestration layer therefore discards calibration information the agents produce, leaving the multi-agent pipeline with strictly less useful confidence estimates than the single-LLM for downstream uses such as active learning or human-review triage.

4.5.4 Magellan Errors. Magellan's lower F1-score is driven by recall (0.5427) more than precision (0.6876): it is reasonably correct when it predicts a match but misses too many. This still exceeds the DeepMatcher study's $F_1 = 0.436$ for Magellan on Abt-Buy [9] by about 17 points, likely due to the hyperparameter tuning and threshold optimization performed here.

The Magellan feature set encodes string and numeric similarity over names, descriptions, and prices, but it cannot represent world knowledge. For example, it cannot recognize that a *Terk XM mini-tuner home dock with model CNP2000H* is the same physical product as an *Audiovox CNP2000H XM mini-tuner home dock*, even though the *CNP2000H* dock is sold under both brand names. Both LLM pipelines correctly predict `MATCH` with confidence ≥ 0.92, dismissing the brand mismatch as retailer labeling on top of a shared model number, while Magellan predicts `NO MATCH` because its features interpret the brand-token disagreement as overshadowing the shared *CNP2000H* identifier.

5 Discussion

5.1 What Multi-Agent Decomposition Buys and What it Does Not

The central empirical result of this study is that under a knowledge-controlled environment, multi-agent decomposition produces the same F1-score as a single-LLM baseline at roughly nine times the token cost. The two pipelines share a mean Cohen's Kappa of $\kappa = 0.942 \pm 0.005$ across the five runs and agree on 98.7% of pairs. In the 25 verdicts where they disagreed on Run 1, the single-LLM is correct 12 times while the multi-agent is correct 13 times; pooled across all five runs the same near-even pattern holds (single-only correct on 64 pairs vs. multi-only correct on 51, out of 115 discordant pairs). However, this does not mean multi-agent decomposition is useless. The architecture reliably shifts the decision boundary in a specific, interpretable direction toward higher precision and lower recall by forcing agents to attend to tool-based, token-level evidence rather than relying solely on pattern matching. It does mean that the F1-score gains reported for multi-agent ER systems in prior work [1, 12] are unlikely attributable to decomposition itself. When agents are stripped of retrieval, self-annotation, and external knowledge sources as they were in this study, the performance

improvement over a single-LLM effectively disappears, while the increases in token cost and running time remain.

I view these results as a scoping claim about prior work rather than a refutation: multi-agent architectures can and do outperform single-LLM architectures, but on this benchmark the operative component is information augmentation, not the architecture itself. CMAS [12] gains from retrieved demonstrations and type-feature extraction, and Althaf et al. [1] benefit from an explicit RAG layer. The per-agent counterfactual reinforces this: under knowledge isolation the semantic agent reaches only $F_1 = 0.845$ on its own, and its embedding-cosine signal collapses on the same sparse, overlapping-brand-vocabulary pairs where an external knowledge source would have helped. Without such augmentation, decomposition alone is at best a precision-for-recall adjustment, worth the 8.7× token premium only when precision is strictly prioritized.

5.2 LLMs as Zero-Shot Entity Resolvers Versus Established Methods

Beyond the multi-agent question, the results carry a broader implication for the practical adoption of LLMs in ER pipelines. Both LLM pipelines achieve F1-scores near 0.90 on the Abt-Buy dataset, substantially outperforming the Magellan Random Forest baseline ($F_1 = 0.607$) and performing comparably to DITTO, which reports $F_1 = 0.893$ on the same benchmark after task-specific fine-tuning [8]. The single-LLM baseline achieves this with zero labeled training data, no per-dataset feature engineering, and no fine-tuning.

This zero-shot capability has practical significance for organizations that face ER tasks across multiple domains or datasets: traditional ML pipelines require per-schema feature engineering and labeled training data, and DITTO-style approaches still require labeled data and per-dataset GPU fine-tuning, whereas the LLM approach only requires prompt engineering (in my implementation, a system prompt describing what to consider and how to structure the output).

The cost of this convenience is token expenditure. At mid-tier hosted-LLM pricing of roughly \$1–\$15 per million tokens,[2] processing all 16,629 blocked pairs costs approximately \$11–\$164 with the single-LLM versus \$95–\$1,427 with the multi-agent pipeline (8.7× premium). Both are orders of magnitude cheaper than the human labor needed to label training data and maintain a supervised pipeline, but the multi-agent premium offers no measurable performance gain under knowledge isolation, making the single-LLM the more cost-efficient choice. As API costs decline and open-source models proliferate, prompt-based ER may become the default starting point for new ER tasks.

5.3 Confidence-Guided Hybrid Pipelines

Both LLM pipelines produce meaningfully calibrated confidence: the single-LLM's mean confidence drops from 0.924 on true positives to 0.764 on false negatives, and the multi-agent pipeline shows the same pattern at the agent level (though the orchestration layer compresses it). This opens the door to a confidence-guided hybrid in which the LLM handles clear-cut cases at low cost while pairs below a tuned threshold are routed to human review or deterministic rules for known error patterns (color and finish variants alone account for ∼35% of false positives in §4.5).

[2]Public per-token pricing for mid-tier hosted LLMs varied within roughly this range at the time of writing; the absolute numbers are order-of-magnitude estimates.

5.4 Limitations

Single benchmark and two-table scope. All results are on the Abt-Buy product catalog: textual, English-language, two-table, e-commerce-specific, with a 10.8% positive class rate. Conclusions may not transfer to domains with much lower base rates, longer records (e.g., bibliographic), non-textual matching signals, or multi-source/cross-database settings where schema alignment and transitive closure introduce additional challenges. **Single model.** Only `gpt-oss-120b` is evaluated; a stronger frontier model could either widen or close the gap between the single-LLM and multi-agent pipelines, and this study cannot adjudicate that. The midsize model is justified by prior work showing such models are competitive on ER [13]. **Tool design.** The multi-agent tool-set was intentionally restricted to local, computable signals to enforce knowledge isolation, so the findings scope specifically to the no-augmentation regime, not to multi-agent ER in general [12]. **Sequential execution.** Parallelizing the multi-agent pipeline's two agent calls would roughly halve runtime but not change tokens or accuracy.

6 Conclusion

In a knowledge-controlled three-way comparison on Abt-Buy, the multi-agent LLM pipeline is statistically indistinguishable from the single-LLM baseline in F1-score (0.897 vs. 0.907, $\kappa = 0.94$, McNemar $p = 0.26$) at 8.7× the token cost and 6.6× the runtime, while both LLM pipelines clearly outperform Magellan ($F_1 = 0.607$); the single-LLM in particular matches DITTO [8] at $F_1 \approx 0.90$ without any task-specific training or GPU fine-tuning. Under knowledge isolation, the multi-agent architecture's only contribution is a precision-for-recall trade, supporting the interpretation that the F1-score gains reported for multi-agent ER in prior work [1, 12] are best attributed to retrieval and external knowledge rather than to decomposition itself. Future work should extend this evaluation to multi-source and cross-database ER, mix heterogeneous base models within an agent set, and design confidence-guided hybrid pipelines that route uncertain pairs to specialized agents, deterministic rules, or human review.

Acknowledgments

This research project is dedicated to my undergraduate course CIS6930: Data Engineering.

References

[1] Aatif Muhammad Althaf, Muzakkiruddin Ahmed Mohammed, Mariofanna Milanova, John Talburt, and Mert Can Cakmak. 2025. Multi-Agent RAG Framework for Entity Resolution: Advancing Beyond Single-LLM Approaches with Specialized Agent Coordination. *Computers* 14, 12 (2025), 525.

[2] Olivier Binette and Rebecca C Steorts. 2022. (Almost) all of entity resolution. *Science Advances* 8, 12 (2022), eabi8021.

[3] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An overview of end-to-end entity resolution for big data. *ACM Computing Surveys (CSUR)* 53, 6 (2020), 1–42.

[4] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680* (2024).

[5] Mahmoud Mohamed Ashour Hussein. 2025. LLM-Enhanced Entity Matching: Comparative Analysis of traditional and modern techniques. (2025).
[6] 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 (2016), 1197–1208. doi:10.14778/2994509.2994535
[7] Huahang Li, Shuangyin Li, Fei Hao, Chen Jason Zhang, Yuanfeng Song, and Lei Chen. 2024. Booster: leveraging large language models for enhancing entity resolution. In *Companion Proceedings of the ACM Web Conference 2024*. 1043–1046.
[8] 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* (2020).
[9] 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*. 19–34.
[10] Navapat Nananukul, Khanin Sisaengsuwanchai, and Mayank Kejriwal. 2024. Cost-efficient prompt engineering for unsupervised entity resolution in the product matching domain. *Discover Artificial Intelligence* 4, 1 (2024), 56.
[11] 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*. 96–109.
[12] Zihan Wang, Ziqi Zhao, Yougang Lyu, Zhumin Chen, Maarten de Rijke, and Zhaochun Ren. 2025. A cooperative multi-agent framework for zero-shot named entity recognition. In *Proceedings of the ACM on Web Conference 2025*. 4183–4195.
[13] Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. 2025. A Deep Dive Into Cross-Dataset Entity Matching with Large and Small Language Models.. In *EDBT*. 922–934.

A Reproducibility

Code and Data. All source code, prompts, and post-hoc analysis scripts are released at https://github.com/Fritozz-105/cis6930sp26-project (commit `59704b3`). The Abt-Buy benchmark is loaded from HuggingFace at https://huggingface.co/datasets/matchbench/Abt-Buy; the predefined `train`/`valid`/`test` splits are used unmodified, and all final metrics are reported on the 1,916-pair test split. TF-IDF blocking and the cached LLM responses produced by each run are written under `data/cache/` per the released code.

Environment and Hardware. Experiments ran on an Apple M4 Pro (24 GB RAM, macOS) with Python 3.12 managed by `uv`. LLM API calls used `gpt-oss-120b` via the OpenAI Python SDK at temperature 0, issued sequentially. The multi-agent pipeline serves its tools via FastMCP over `stdio` transport as a subprocess, with an isolated response cache per run. Magellan run-to-run variability is controlled by the Random Forest `random_state` (varied across the five runs); LLM run-to-run variability comes from residual API non-determinism at temperature 0.

Commands. After `uv sync`, the full evaluation reproduces the reported metrics:

```
uv run python -m src.main blocking
uv run python -m src.main magellan-tune
uv run python -m src.main evaluate-all --runs 5
```

Individual pipelines can be exercised with `single-llm`, `multi-agent`, or `magellan` subcommands; `–max-pairs N` truncates to a smoke-test slice. Per-pair LLM verdicts and reasoning are preserved in `data/cache/`.

MCP Tools. The multi-agent pipeline exposes five tools. **Edit distance** returns the normalized Levenshtein distance between two product names (raw character edit distance divided by the longer name's length). **Jaccard similarity** returns the word-token Jaccard coefficient. **BM25** returns symmetric BM25 relevance, scoring each name against a two-document corpus and averaging, normalized to $[0, 1]$. **Price comparison** returns the price ratio, absolute difference, and a similarity flag (ratio ≥ 0.8), with `null` fields when prices are missing. **Embedding cosine** returns the cosine similarity between `all-MiniLM-L6-v2` sentence-transformer embeddings of the two names.

System Prompts. All prompts were held constant across the five runs.

Single-LLM: You are an expert at entity resolution. Determine whether two product listings refer to the same real-world product. Analyze names, descriptions, and prices, considering minor name variations (typos, abbreviations, word order), matching specifications (model numbers, capacities), compatible price ranges, and overlapping descriptions. Respond ONLY with a JSON object: `{"verdict": "MATCH"|"NO MATCH", "confidence": 0.0-1.0, "reasoning": ...}`.

Syntactic agent: You are a syntactic similarity analyst. Call ALL three tools (edit distance, Jaccard, BM25) on the product names, then decide. Respond ONLY with the JSON schema above.

Semantic agent: You are a semantic similarity analyst. Call BOTH tools (price comparison, embedding cosine), then decide. Respond ONLY with the JSON schema above.

Orchestrator: You are the orchestrator for a multi-agent product entity resolution system. Two specialized agents have independently analyzed the pair. Review their verdicts, confidence scores, and reasoning. You may agree or override their decision if the reasoning is flawed, inconsistent with the evidence, or not justified by the confidence scores. Respond ONLY with the JSON schema above.

Received 1 May 2026; revised 1 July 2026; accepted 1 August 2026