Liberal Entity Matching as a Compound AI Toolchain (Extended Abstract)

Silvery D. Fu[1,2], David Wang[1], Wen Zhang[1], Kathleen Ge[1]
[1]UC Berkeley, [2]System Design Studio

Abstract

As developers increasingly embrace the capabilities of new large language models (LLMs), the focus on AI application development is shifting from only focusing on models to developing compound systems with multiple components to achieve state-of-the-art results [18]. In this paper, we explore the task of *entity matching* (EM) with a compound AI system approach. EM is a fundamental problem in data management and integration, which involves determining whether two descriptions refer to the same real-world entity [7]. For instance, consider determining if product descriptions on Ebay and Amazon (see Fig.1) refer to the same product.

Entity matching has evolved from rule-based and distance-based approaches [15], to machine learning [13, 17], crowdsourcing [13, 16], deep learning [8], and pre-trained language models (PLMs) [7, 9], showing a continuing trend of improved results. Recently, the large language models (LLMs) based approach came to the fore and showed state-of-the-art results over multiple datasets [5, 10, 11] with prompt engineering such as hand-crafted rules and in-context learning (ICL). We refer to this as *solo-AI EM*, which prompts LLMs to perform entity matching in a single model call.

However, important challenges remain with the current solo-AI approach: (1) Solo-AI EM often relies on **hand-tuned** rules and examples as part of prompt engineering. To do this, humans often need to perform trial and error to find the right prompt for each new dataset. For example, an LLM may not know that under a given EM setting, color distinguishes different products and thus requires manual specification or hand-picked few-shot examples for the specific dataset. (2) Solo-AI EM relies on **static** knowledge from training data and thus may fail at entity matching for entities that appear beyond that time. For example, when new products have just been released, LLMs may fail to correctly recognize them for matching. (3) Entity matching is typically a stage in a larger data processing pipeline (*e.g.,* entity resolution) that involves other forms of data processing. Current solo-AI EM follows a **rigid** preprocessing of input data with predefined rules and does not allow for adapting the input data in a format best suited for the model and task. For example, as we will show, for several datasets, simply including schema information in the entity data—which is typically stripped off in existing systems—can improve the matching accuracy.

Further, the solo-AI approach makes it hard to iterate over the system designs. Existing solo-AI EM systems are usually in the form of Python notebooks [10], as opposed to libraries that one can easily incorporate in their applications or APIs to use as a service. Such a lack of system modularity also makes it difficult to configure and optimize the EM system to navigate the trade-offs between accuracy and other performance metrics.

In this paper, we argue that entity matching should be performed *liberally* by AI as opposed to being constrained by (1)-(3), to maximize its accuracy, performance, and ease-of-use. Our key insight is that given that LLMs can not only provide knowledge, but also behaviors such as invoking external tools, an EM system should provide proper *tools* for LLMs so that they can solve the tasks better and even self-improve their performance. To achieve liberal EM, we

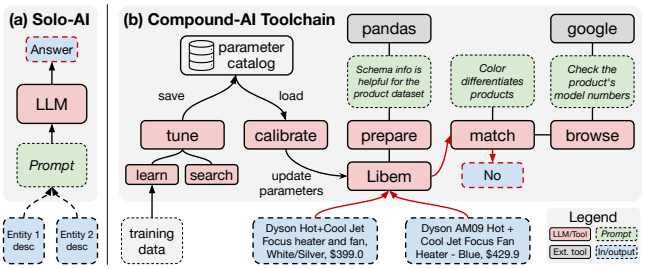

Figure 1: Entity Matching with Solo-AI vs. Compound-AI Toolchain. In Libem: (1) EM is performed liberally with LLM using tools such as data preprocessing ("prepare" tool) and browsing external data sources ("browse" tool); (2) Parameters to configure each tool can be learned when the training data and/or performance metrics are provided.

argue that an EM system should be best designed as a compound AI system [18] that consists of both AI and system components. Specifically, we want an EM *toolchain* that provides:

- **Tool use.** The toolchain should provide relevant tools, such as data processing and information retrieval, so that a model can liberally decide what, when, and how to leverage tools to better perform the EM task.
- **Self-refinement**. The toolchain should adapt to the input dataset and improve its performance without hand tuning when training data is available. Inspired by DSPy [6, 12], we aim for the toolchain to start with simple, general prompts and evolve towards better, more task/dataset specific prompts and parameters that achieve higher matching accuracy.
- **Optimization.** Users should be able to easily configure and optimize (*e.g.,* turn off chain of thought in the browsing step to avoid lengthy reactions for search results) to navigate the trade-offs between performance and cost. Similar to self-refinement, the toolchain should be capable of automatic optimization.

We propose **Libem**, a *compound AI toolchain* that aims to perform entity matching through tool use, self-refinement, and optimization. To achieve this, we made several key design choices in Libem. First, instead of a collection of prompts, we structure Libem as a collection of composable and reusable modules/tools, as shown in Fig.1, where each tool can also be individually invoked for testing or external (re)use. Second, we separate the parameters (*e.g.,* discrete configurations and prompts) from the organization of tools, allowing the parameters to be tuned and configured, as inspired by DSPy [6]. Each Libem tool has its own parameters and may invoke model calls, other Libem tools, or external APIs. For example, the match tool can take an entity pair and return a prediction, and may invoke the browsing tool, while the browsing tool can use Google Search, and the preparation tool can utilize Pandas. Libem can self-refine through training data and can save the optimal parameters for reuse. Third, upon each run of the Libem match, a calibration process takes into account the input dataset and the performance goals to configure Libem (*e.g.,* using the saved parameters) to best perform the incoming EM task.

Dataset	Precision		Recall		F1	
	S	C	S	C	S	C
Abt-Buy	95.1	96.6	95.1	96.1	95.1	**96.4**
Walmart-Amazon	84.3	94.05	94.3	81.87	89.0	87.53
Amazon-Google	63.8	71.1	92.7	98.7	75.6	**82.6**
DBLP-Scholar	89.7	90.6	87.2	96.0	88.4	**93.2**
DBLP-ACM	94.0	96.5	100.0	99.6	96.9	**98.0**

Table 1: EM Accuracy with Solo- (S) vs. Compound-AI in Libem (C).

Design overview. Fig.1 describes the internals and typical workflow of entity matching with Libem. Each tool module has functions, interfaces, parameters, and prompts. The tools are organized in a hierarchy, where the parent provides the interface to access the tool. For example, the *libem.interface* contains match, prepare, and tune tools, and the *libem.match.interface* contains the browse tool.

(1) EM with tool use. When Libem attempts to perform a match, multiple model calls and tool calls may occur. Upon invoking Libem, `libem.match` is activated, and the model decides which tools to use given the tool's parameters. For instance, if the browse tool is selected, Libem will first search the web for the entity information and then use the retrieved data to perform the matching.

(2) Self-refinement. Libem supports self-refinement in the following manner: The `tune` tool invokes `libem.match` on training samples and obtains *rules* or *experiences* (*i.e.,* mistakes to avoid). We implement three simple forms of self-refinement strategies when training data is available: (a) generate rules from successful matches (*e.g.,* "color differentiates entities"); (b) generate experiences from failed matches; (c) search for optimal parameters. Learned rules, experiences, and parameters are saved in a persistent catalog. When input entities are provided, the `libem.calibrate` tool determines which set of parameters to use with the match step. For example, if the input entities are products, Libem loads and calibrates its tools with corresponding parameters from the parameter catalog.

(3) Extending the toolchain. Additional tools can be easily added by defining new tools and adding them to the Libem toolchain. We include a simple code generator with the Libem CLI to facilitate the generation of boilerplate code for new tools.

Libem prototype. Libem is under active development. Our current prototype consists of 2,589 SLOC in Python and includes the following top-level tools: `match`, `browse`, `prepare`, `tune`, `calibrate`, `optimize`, and sub-level tools such as `learn`, `search` in `libem.tune` (Fig.1). Besides, we have added logging, tracing, and telemetry code to enable users to easily track and debug the system's behavior. Libem can be imported as a Python library, invoked via command line, or interacted as a web service.

Early experiments. We evaluate Libem by running entity matching on real-world datasets covering product information and bibliographical data from Abt-Buy, Walmart-Amazon, Amazon-Google, DBLP-Scholar, and DBLP-ACM [7, 11], as used in prior research [5, 10]. We compare Libem to a solo-AI baseline and report precision, recall, and F1 score. We use GPT-4-turbo to process model calls.

Table 1 presents our initial findings. First, we want to point out that these datasets were released before the model was trained, thus our results run the risk of data leakage [3], as was also the case in prior work [5, 10]. Therefore, we are collecting new datasets that were not included in model training, *e.g.,* from Bandai online shops and Bandai Wiki, featuring the newest releases in April 2024, also to test out the capabilities of the browsing tool. Nonetheless, as shown in Table 1, Libem outperforms the solo-AI counterpart in four out of five existing datasets. We observe a 3% increase in the average F1 score across the five datasets, with a maximum of 7% in the Amazon-Google dataset. Due to space limitations, we summarize the following early findings from experimenting with Libem: (1) We can achieve better or comparable performance without manually tuning the prompts, compared to those that involve human-in-the-loop tuning, where self-refinement helps avoid common mistake patterns. (2) A simple choice of enabling schema helps substantially in matching accuracy. (3) Browsing can be highly beneficial when the data sources are recent; however, it often needs to be accompanied by explicit 'chain of thought' prompting to perform well.

Ongoing and future work. We are actively developing Libem with a focus on the following fronts: (i) Better tooling. We are extending and enhancing the tools in the Libem toolchain, such as browsing on user-supplied data sources. (ii) Matching speed and efficiency. Optimizing performance metrics is often as crucial as accuracy. We are working on enhancing per-match latency, throughput (*e.g.,* with batching [5]), and token efficiency (*e.g.,* with caching [1]). We are investigating mechanisms for dynamic trade-offs and optimizations between token usage, accuracy, and speed, such as generating classifiers like random forests from rules learned by Libem to be used in place of model calls during matching. (iii) Refinement strategies. We are exploring alternative strategies and algorithms for self-refinement and calibration, for both prompts and other parameters, including search algorithms such as Bayesian optimization and synthetic data generation [6]. (iv) Practicality. We are investigating how to better deploy and serve the toolchain efficiently, and how to measure and ensure its robustness, *e.g.,* with tracing and new programming primitives like assertions [12]. We plan to conduct large-scale evaluations with more datasets [5] and with open-source models [2, 14].

Finally, we plan to apply the compound AI toolchain approach to broader tasks, such as entity resolution, data cleaning [4], and schema matching and mapping [19], to develop practical compound AI solutions. The Libem library, examples, and benchmarks will be available as an open-source project at https://libem.org.

References

[1] 2024. GPTCache : A Library for Creating Semantic Cache for LLM Queries. https://github.com/zilliztech/GPTCache.

[2] 2024. Introducing DBRX: A New State-of-the-Art Open LLM. https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm.

[3] Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondřej Dušek. 2024. Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source llms. *arXiv preprint arXiv:2402.03927* (2024).

[4] Zui Chen et al. 2023. Seed: Domain-specific data curation with large language models. *arXiv e-prints* (2023), arXiv–2310.

[5] Meihao Fan et al. 2023. Cost-Effective In-Context Learning for Entity Resolution: A Design Space Exploration. *arXiv preprint arXiv:2312.03987* (2023).

[6] Omar Khattab et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714* (2023).

[7] Yuliang Li et al. 2020. Deep entity matching with pre-trained language models. *Proc. VLDB* 14, 1.

[8] Sidharth Mudgal et al. 2018. Deep learning for entity matching: A design space exploration. In *Proc. ACM SIGMOD*.

[9] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB* 16, 4.

[10] Ralph Peeters et al. 2023. Entity matching using large language models. *arXiv preprint arXiv:2310.11244* (2023).

[11] Ralph Peeters et al. 2023. Using chatgpt for entity matching. In *European Conference on Advances in Databases and Information Systems*. Springer.

[12] Arnav Singhvi et al. 2023. DSPy Assertions: Computational Constraints for Self-Refining Language Model Pipelines. *arXiv preprint arXiv:2312.13382* (2023).

[13] Michael Stonebraker et al. 2013. Data curation at scale: the data tamer system.. In *CIDR*, Vol. 2013.

[14] Hugo Touvron et al. 2023. Llama: Open and efficient foundation language models (2023). *arXiv preprint arXiv:2302.13971* (2023).

[15] Jiannan Wang et al. 2011. Entity matching: How similar is similar. *Proc. VLDB*.

[16] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. *Proc. VLDB*.

[17] Pei Wang et al. 2021. Automating entity matching model development. In *Proc. IEEE ICDE*.

[18] Matei Zaharia et al. 2024. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

[19] Yunjia Zhang et al. 2023. Schema matching using pre-trained language models. In *Proc. IEEE ICDE*.