Generating Data Augmentation Queries Using Large Language Models*

Christopher Buss
Oregon State University
bussch@oregonstate.edu

Jasmin Mousavi
Oregon State University
mousavij@oregonstate.edu

Mikhail Tokarev
Oregon State University
tokarevm@oregonstate.edu

Arash Termehchy
Oregon State University
termehca@oregonstate.edu

David Maier
Portland State University
maier@pdx.edu

Stefan Lee
Oregon State University
leestef@oregonstate.edu

Abstract

Users often want to augment entities in their datasets with relevant information from external data sources. As many external sources are accessible only via keyword-search interfaces, a user usually has to manually formulate a keyword query that extracts relevant information for each entity. This is challenging as many data sources contain numerous tuples, only a small fraction of which may be relevant. Moreover, different datasets may represent the same information in distinct forms and under different terms. In such cases, it is difficult to formulate a query that precisely retrieves information relevant to a *specific* entity. Current methods for information enrichment mainly rely on resource-intensive manual effort to formulate queries to discover relevant information. However, it is often important for users to get initial answers quickly and without substantial investment in resources (such as human attention). We propose a progressive approach to discovering entity-relevant information from external sources with minimal expert intervention. It leverages end users' feedback to progressively learn how to retrieve information relevant to each entity in a dataset from external data sources. To bootstrap performance, we use a pre-trained large language model (LLM) to produce rich representations of entities. We evaluate the use of parameter efficient techniques for aligning the LLM's representations with our downstream task of online query policy learning and find that even lightweight fine-tuning methods can effectively adapt encodings to domain-specific data.

1 Introduction

There is a recognized need to collect and connect information from a variety of data sources [5, 13, 10]. For example, we have recently worked in a large-scale NIH-funded project to augment the information of biomedical entities by querying other biomedical data sources [32]. The focus of this project is to *repurpose* current drugs to treat the symptoms of diseases for which there is insufficient time or resources to develop new treatments [1]. Biomedical researchers often have some local dataset

*Presented at the 1st International Workshop on Databases and Large Language Models at VLDB 2023

Table Representation Learning Workshop at NeurIPS 2023.

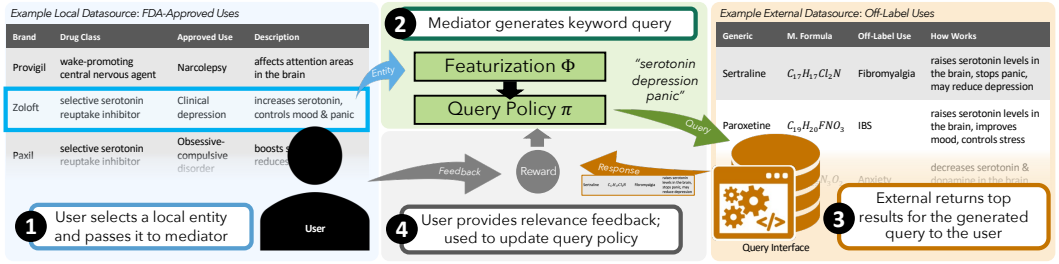


Figure 1: An example of our framework for a single user and single external data source. The user selects (by query, GUI, etc,.) the local entity *Zoloft*. The mediator uses its learned query policy to extract the relevant entity (*Sertraline*) from the external source. The user provides relevance feedback on the results, which is then used to further refine the mediator's querying policy.

of available drugs. Given a drug in their dataset, a researcher usually needs to query external data sources to find additional information about the drug.

Due to a lack of access or resources, external information often must be retrieved through querying [5, 30]. Many external datasets are only accessible via query interfaces or APIs. Even with access, it may require too much of a resource (e.g., storage space, time) to download and maintain an up-to-date copy of the external dataset. Thus, information relevant to some local entity must often be gathered on a as-needed basis by querying external data sources. For example, the users of the aforementioned drug repurposing data collection system must often query the information relevant to their current drug of interest through query APIs.

However, formulating a query that extracts specific information can be troublesome. Different data sources often represent the same concept in distinct forms [6]. Thus, one needs to tailor their query to specific data sources. Figure 1 illustrates a case where users have a local dataset of FDA-approved uses of drugs and would like to query an external data source that contains the off-label uses of those drugs. A drug that is identified by one of its brand names (e.g., *Zoloft*) in *FDA-Approved Uses* is referred to by its generic name (e.g., *Sertraline*) in *Off-Label Uses*. Due to heterogeneities, one may not know how to query for a specific external entity prior to investigating the content and structure of data in the external source. Consider a biomedical researcher who seeks more information about the drug *Zoloft*. Since they are only aware of the structure and content of their local dataset, they query the external data source for *Zoloft*, but this elicits no results. They try again using a more general description of *Zoloft* (i.e., being a *serotonin reuptake inhibitor*). However, their under-specified query produces many results, most of which are irrelevant. After additional trial-and-error, they find a query that retrieves *Sertraline*. More work is required to then merge the local and external entities into one coherent representation.

Manually querying for specific external entities requires too much work. Continuing our example, if the researcher needs more information for another drug in their local dataset, they will need to repeat the process. Moreover, if they need information from multiple external data sources, then the work required to query for each drug is exacerbated. Furthermore, other researchers with similar information needs must *repeat the same such work themselves*.

To alleviate the burden, one can use a **shared system that automates query formulation**. This **mediator system** acts as a go-between for users and external data sources: a user specifies a local entity (e.g., *Zoloft*) perhaps through a query or a graphical user interface, and the mediator maps the local entity to queries that retrieve the relevant external entities (e.g., *Sertraline*) from their respective external sources.

To the best of our knowledge, such mediators are currently created by *manually writing programs* that generate queries for specific external sources. These programs consist of rules that *cannot* necessarily be reused across data sources. Thus, they require a significant amount of labor and expert attention to build and maintain. Instead of conducting their own research, biomedical researchers in our NIH-funded project spend most of their time writing and maintaining these programs.

In this paper, we *learn the mediator's query policy online* through user interaction. As illustrated in Figure 1, after the user specifies a local entity, the mediator formulates a query to retrieve records from an external source according to its *query policy* and shows the returned external records to the

user. The user then provides feedback on the relevance of the returned records to the local entity. The mediator then uses this feedback to improve its query policy.

Of course, online learning of query policies has its own set of challenges. First, the mediator must learn a sufficiently effective policy in the *short run* so users will continue providing feedback. This challenge is easier to meet when the users' only alternative is tiresome (i.e., manually submitting queries for many local entities) or there are many users providing feedback. Second, the mediator should continue leveraging user feedback to find increasingly effective policies in the *long run* (i.e., it should not be prone to under-fitting to local entities). To help overcome these challenges, we use a pretrained large language model (LLM) to extract features from local entities and terms. Through pretraining, LLMs encode linguistic knowledge within the rich representations of their outputs. However, to get the most out of an LLM, its output representations should be adjusted to suit the specific task and domain. This is commonly done through finetuning, where the weights of the LLM are trained jointly with the task-specific model. However, finetuning is resource-intensive and may overwrite the LLM's knowledge [24]. Thus, in this paper, we evaluate more parameter efficient techniques for our online setting.

We present a framework for on-demand collection of relevant external entities only accessible via query interfaces (Section 2) and define the problem of online query-policy learning within the context of the aforementioned framework (Section 2.1). Due to the wide-spread use of keyword query interfaces over external sources, we use an online learning method for formulating *keyword queries*. We evaluate prefix tuning and attribute encoding as parameter efficient techniques for boosting the performance of an LLM-based query policy learner over four pairs of real-world datasets. We find that even lightweight fine-tuning methods, specifically prefix tuning, can effectively adapt encodings to domain-specific data.

2 General Framework

The **mediator** wraps the local dataset and the query interface over the external data source. We assume the mediator has full access to the local dataset, but can only access external datasets through their query interfaces. Given a user-specified entity from the local dataset, the mediator must devise and submit a query to the interface to extract external entities relevant to the given local entity. This framework is not tied to a particular method by which a user specifies the local entity (e.g., through query or GUI).

Local Dataset. To simplify our exposition, we assume the local dataset is a single relational table where each tuple stores information about a distinct entity. One may extend our approach to multi-relational datasets by defining an entity as the join of its related tuples. We denote the set of local dataset entities as $\mathcal{E}$.

External Dataset. Like the local dataset, we model the external dataset as a set of entities (i.e., tuples). Given local entity e and external dataset D, $X(e) \in D$ represents the external entity that is relevant to the local one. The definition of "relevant entity" is domain-dependent (e.g., a clinical trial is relevant to the drug that it concerns). For notational convenience, we assume one relevant external entity exists for each local entity, however, in the case of more than one, we can easily extend $X(e)$ to be the set of all relevant entities. If no relevant entities exist, then extracting $X(e)$ is impossible regardless of the method used. Thus, to accurately evaluate our methods, we assume that $X(e)$ always exists.

Example 2.1. Figure 1 shows excerpts of a local (left) and an external (right) dataset. $\mathcal{E}$ consists of all drugs in *FDA-Approved Uses*. If e is *Zoloft* then the relevant tuple $X(e)$ in *Off-Label Uses* is *Sertraline*. We show the content of $X(e)$ for explanation's sake. In a real setting, the content of $X(e)$ would not be known a priori.

Querying Policy. We call the queries submitted by the mediator to the external data source *mediator queries*. We denote the set of *all possible* mediator queries as $\mathcal{Q}$. The precise definition of $\mathcal{Q}$ varies based on the characteristics and capabilities of the external query interfaces. A *querying policy* (*policy*) is a mapping $\pi : \mathcal{E} \rightarrow \mathcal{Q}$. To our knowledge, policies are traditionally written manually.

Example 2.2. Given e = *Zoloft*, the mediator must devise a keyword query to extract $X(e)$ = *Sertraline*. One policy is to use the content of the input entity (*Zoloft*) within the output mediator query. However, the terms in `Brand` are likely unique to the local dataset. Given this observation, assume the policy ignores terms in `Brand` and maps e (*Zoloft*) to the keyword query *"serotonin depression panic"*.

Query Result. External query interfaces usually return results of query q as a list of entities inversely sorted based on the degree by which the query interface deems the entities relevant to q. More precisely, the *result* of query $q \in \mathcal{Q}$, $D[q]$, is a list of entities in D.

Query Effectiveness. Ideally, we would like the mediator query q submitted for local entity e to return the external entity relevant to e (i.e., $X(e)$ is placed in a relatively high position in $D[q]$). Given mediator query q for local entity e, we define the effectiveness of q over external dataset D as a real-valued function $f(X(e), D[q])$ whose range is in $[0, 1]$. The precise mechanics of f depends on the domain. For instance, there are standard metrics in information retrieval and data management to measure how effectively queries achieve this goal given their returned results [23]. For example, *precision@*k is the fraction of relevant answers in the top-k returned results. Another frequently used metric is *reciprocal rank* (RR) $\frac{1}{r}$ where r is the position of the first relevant answer. One metric may be more appropriate than another for a specific setting. For instance, reciprocal rank may be a better indication of effectiveness than precision@k if there are at most a couple relevant answers to the query. One can choose f based on the domain. In this paper, we use reciprocal rank.

Example 2.3. The mediator submits $q =$ *"serotonin depression panic"* to the query interface over the external dataset in Figure 1, which returns the ranked results $D[q] =$ (*Paroxetine*, *Sertraline*). Since $X(e) =$ *Sertraline*, the reciprocal rank of these results would be $\frac{1}{2}$.

Effectiveness of Policy. A mediator's policy is evaluated based on the effectiveness of the queries it produces. More formally, the effectiveness of policy π for local dataset $\mathcal{E}$ and external dataset D is $F(\mathcal{E}, D, \pi) = \sum_{e \in \mathcal{E}} P(e)\ f(X(e), D[\pi(e)])$ where $P(.)$ is the prior probability of choosing local entities for augmentation by users.

2.1 Learning Query Policy Progressively

We propose a method for learning this querying policy progressively. In our online approach, external relevant information would be presented to the user *on-demand* as they identify entities of interest in the local dataset [21].

Algorithm 1 describes the mediator's procedure for online query-policy learning. The mediator is involved in a series of *interactions* with the external data source D and a group of local data source users. The $t'th$ interaction is initiated by sampling a local entity for augmentation. How e_t is sampled can reflect a local user's preference for augmenting that specific entity at time t. The mediator uses its current policy π_t to map e_t to query q_t. It then submits this query to the external data source to obtain a ordered list of results **d**. Since query interfaces often enforce a top-k constraint on their results [16, 7], we assume that $|d| \leq k$. The mediator presents **d** to the user and evaluates its effectiveness f_t per the user's feedback. The feedback may be explicit (click-through [28]) or implicit (skipping results [17]). The mediator uses f_t to update its policy and find progressively more effective policies over time.

Algorithm 1 Mediator (Online Query Policy Learning)

1: **for** t = 1, 2, 3,...,T **do**
2: Observe local entity e_t sampled from $\mathcal{E}$
3: $q_t \leftarrow \pi_t(e_t)$
4: $\mathbf{d} \leftarrow D[q_t]$ ▷ Results over external data source D
5: Present results to user.
6: Observe degree of effectiveness $f_t \leftarrow f(X(e_t), \mathbf{d})$
7: $\pi_{t+1} \leftarrow update_\pi(f_t)$

Objective Function. The objective of the mediator is to find policies that optimize some function $B(.)$. There are many options for $B(.)$, such as regret [27]. In this work, we seek to balance the short- and long-run effectiveness as measured by reciprocal rank. Thus, it is not enough that the mediator eventually find an effective policy: it must also produce *reasonably effective* results along the way.

Challenges. To balance short- and long run-effectiveness, a mediator must overcome two challenges. First, it must balance *exploration and exploitation*. If it *exploits* the best queries found thus far, it may ignore more effective queries; if it strictly *explores* until it has found the optimal queries then it will likely formulate many ineffective queries in the process. Second, it must maintain user engagement.

It might not be possible to find an effective policy in few interactions. Nonetheless, users may stop providing feedback if the policies perform poorly even after a modest amount of feedback is provided.

Keyword Query Interface and Results. A keyword query q is a finite string comprised of *terms*. The number of terms in a query is its *length* ℓ. To save resources, query interfaces might limit the number of terms in their input queries. For example, Yelp!'s Fusion API and Google.com have limits of 8 and 32 terms respectively. We assume that all queries submitted to an external data source D have a given fixed length. Unlike formal query languages, such as SQL, keyword queries are inherently vague [23, 16]. Thus, formulating precise keyword queries can be challenging.

Managing the Policy Space. The space of potential policies is correlated with the size of $\mathcal{Q}$: the larger $\mathcal{Q}$ is, the more ways that local entities can be mapped to queries. To make our policy space more manageable for our online setting, we both prune $\mathcal{Q}$ and take a term-centric approach when mapping entities to queries.

For any input local entity, only a small subset of $\mathcal{Q}$ will be useful. In order to remove many ineffective queries, we consider only those queries that *express* the input local entity. We define an entity-dependent co-domain $\mathcal{Q}_e \subseteq \mathcal{Q}$. Let $L(e)$ be the set of terms within the content of e. For every entity $e \in \mathcal{E}$, $\mathcal{Q}_e$ contains every possible concatenation of distinct terms $k \in L(e)$. $\mathcal{Q}_e$ might not contain a maximally effective query, but given that relevant entities from related domains often share terms, a reasonably effective query may still be found in many cases.

We also leverage the fact that many queries overlap with respect to their contents. Intuitively speaking, if a subset of terms is shared across effective queries for an entity e, then it is likely that same subset that has contributed to each query's effectiveness. Following this logic, our methods track the effectiveness of terms used within queries. We assume that terms influence query effectiveness independently. This assumption allows our policies to construct queries term-by-term based on each individual term's effectiveness.

Merging Local and External Information. One might have to merge local data with its relevant external data by performing other steps of data integration, such as schema matching [5]. However, it takes more than one paper to investigate all steps of data integration. Thus, we assume that in these settings, users leverage existing data integration tools to create the final dataset and focus on the task of collecting information from external sources effectively.

3 LLM-Based Query Learning

Figure 1 illustrates a single *interaction* of online query policy learning. The mediator's policy is refined progressively over many interactions with the objective of maximizing the mean reciprocal rank (MRR) of its queries. As discussed in Section 1, an optimal method would overcome two major obstacles. First, it would maintain user engagement by producing effective queries in the short run. Second, it would have the capacity to improve its policy in the long run.

We use a pretrained large language model (LLM) to help meet the aforementioned challenges. The model may benefit from the LLM's rich representations of tuples and terms, boosting the model's early performance while also allowing it to fit to the diversity of local entities over time.

Encoding Tuples and Scoring Terms. Given an entity e, we concatenate its terms into a single string s and pass it through an LLM after standard byte-pair-encoding tokenization. The LLM produces a sentence-contextualized representation h_i for each input token. Note that the byte-pair-encoding may break terms into multiple inputs or terms may appear multiple times in the entity, so to produce feature h_i corresponding to term k_i, the output encodings of all these instance are averaged. For convenience, we write this process as: $h_1, ..., h_n = \text{LM}(s)$. These representations capture information about each term given the context of all terms within the entity. However, they lack contextual information about the local data source. Thus, we add this information post-encoding.

We define a feature vector $A_t(k_i, e)$, which contains *distributional* and *schematic* features of terms relative to the local source. One such feature is *Inverse document Frequency* (IDF). Let *Dataset Frequency* (DF) of a term denote the fraction of entities in the local dataset in which the term appears. IDF of a term is the inverse of its DF, and it quantifies how well that term identifies the entity within the dataset. $A_t(k_i, e_t)$ is concatenated onto each corresponding representation forming $c_i = [A_t(k_i, e_t), h_i]$ where $[\cdot, \cdot]$ denotes concatenation. Vector c_i is then passed through a small fully connected layer to predict reciprocal rank r_i for each term.

Table 1: Details of datasets used in our evaluation.

dataset	source	attributes	avg. terms	entities	#relevant
Drugs	Local	drugName, condition, review	108	13,725	413
	External	page_title, wikipedia_summary	168	46,976	
WDC	Local	category, brand, prod_title, description, ...	67	57,109	55,247
	External	category, brand, prod_title, description, ...	72	55,247	
ChEBI	Local	name, description, indication, pharmacodynamics, ...	178	5,483	5,753
	External	status, name, definition, charge, formula, mass, ...	73	189,467	
CORD-19	Local	abstract	305	250,575	250,575
	External	sha, source_x, paper_title, doi, pmcid, ...	48	340,826	

As discussed in Section 1, we desire parameter efficient methods for adjusting the output of the LLM to our specific task and data. We consider two such methods: *prefix tuning* and *attribute embeddings*.

Prefix Tuning. We use prefix tuning as an alternative to updating all weights of the LLM [18]. Before passing the base encoding of entity e (i.e., s) through the LLM, we prepend a prompt consisting of d vectors onto s. This contextualizes the output of all tokens in s on this continuous prompt. Feedback is propagated back to these d vectors to update them, resulting in downstream representations that are increasingly more aligned with our objective.

Attribute Embeddings. To inject the structural information of local entity e within its downstream representation, we adjust the base encoding of s prior to passing it through the LLM [8]. Each attribute (column) within the local dataset is encoded as a vector. These vectors are then added to tokens to provide attribute information. As in prefix tuning, these encodings are updated based on feedback.

Selecting Queries and Updating. To encourage exploration, we apply an ϵ-greedy approach to query formulation [27] — selecting either the next-highest-scoring term or, with probability ϵ, a random term until the desired query length is achieved. User feedback (i.e., reciprocal rank or RR) is used as a prediction target for all query terms appearing in the returned external matches. Unobserved terms have targets of 0 assigned. These term-entity-RR tuples are added to a first-in-first-out buffer of examples for the last 30 observed queries. We train the model by stochastic gradient descent with batches of 8 samples from the buffer at each interaction.

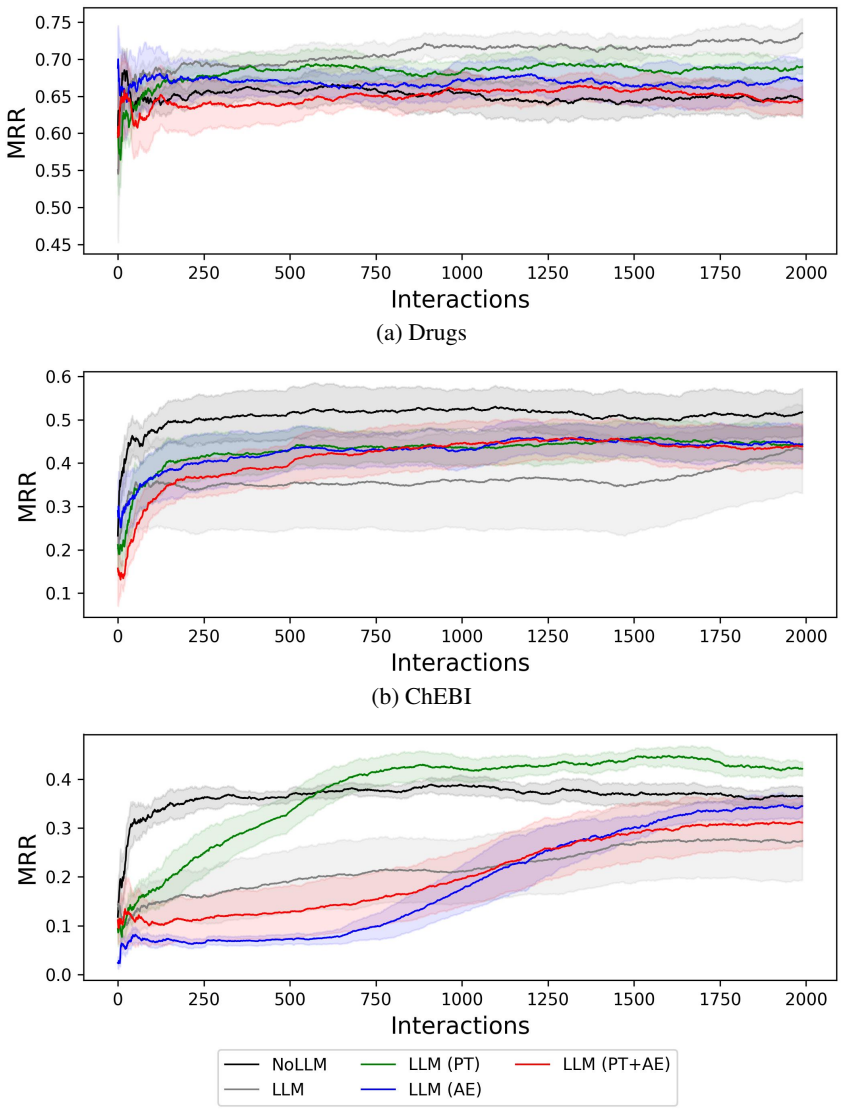
We use a pretrained Longformer model from the Huggingface Transformers library. Parameters are trained using Pytorch's implementation of Adam with default hyper-parameters.

4 Empirical Evaluation

We evaluate our modles over the datasets listed in Table 1. Each one contains a local and an external source. We include the entity count and the average number of terms per entity. Each local entity has at least one relevant external entity, but some external sources have additional irrelevant entities that can appear in results. Thus, we also specify the number of relevant external entities. **ChEBI** is derived from sources used in the NIH project discussed in Section 1. The local source uses *DrugBank* data, which contains molecular information about drugs [31]. The external source uses *ChEBI* data, which contains molecular entities used to intervene in the processes of organisms [15]. **WDC** is derived from the English *WDC Product corpus*, containing products scraped from many sites [25]. **CORD-19** contains research records related to COVID-19 [29]. We split CORD-19 into two sources: one containing abstracts (local) and one containing the remaining attributes (external). **Drugs** contains reviews from *Drugs.com* (local) [14] and descriptions of the same drugs in *Wikipedia* (external).

Interactions. We simulate a series of interactions. Each interaction is initiated by sampling a local entity. Given the entity, the mediator generates a query of length ℓ and submits it to the external source, which returns its top-20 results using BM25. The query is then scored based on simulated feedback (i.e., ground truth).

Sampling. Entity preference tends to follow a Zipf distribution $1/i^s$ where i popularity rank and $s \approx 1$ [4]. Thus, users request the $i'th$ most popular entity approximately twice as often as the $(i+1)'th$ most popular entity. We simulate user preference by sampling local entities from a Zipf distribution ($s = 1$). We randomly assign popularity, which is held constant across methods.



(a) Drugs

(b) ChEBI

(c) CORD-19

Figure 2: Comparison of different configurations of the LLM model and *NoLLM* for $\ell = 16$. *LLM* is the base model whereas *LLM (PT)* uses prefix tuning, *LLM (AE)* uses attribute encoding, and *LLM (AE+PT)* uses both techniques.

Evaluation Metric. We compute mean RR (MRR) as a sliding average over the previous 500 interactions. We report the average of five runs each comprising 2000 interactions. We plot this average against the current interaction. We include error bands around each line to show an 80% interval for standard error across runs.

Hyperparameters. We treat query length as a hyperparameter and use $\ell \in \{4, 16\}$. We use $d = 5$ prefix tokens for prefix tuning along with a moderate amount of exploration ($\epsilon = 0.05$).

Dataset-Level (NoLLM). We include the Dataset-Level model discussed in our previous work as a baseline [2]. This model approximates the rewards of terms as a linear function of their features, which includes the *distributional* and *schematic* discussed in Section 3.

Code and datasets can be found in our Github repository: `https://github.com/bussch/QueryBasedEntityAugmentation`.

4.1 Results

We seek to understand whether prefix tuning and attribute encoding lead to more effective query policies. Figure 2 compares different configurations of the LLM-based model along with *NoLMM* for $\ell = 16$ over three datasets. The full set of results can be found in the appendix.

Though, no single configuration produces the best performance over all datasets, *LLM (PT)* is the most consistently effective configuration. Over CORD-19, it outperforms all other configurations in both the short- and long-runs. In other cases, its performance is similar to that of *LLM*. Compare this with *LLM (PT+AE)* and *LLM (AE)* which often perform worse than either *LLM* or *LLM (PT)*. This suggests that prefix tuning is the more robust technique: it can improve performance over certain datasets and does not significantly degrade performance over others.

One contributing factor to attribute encoding's poor performance on most datasets may be the structure of the datasets themselves. Since ChEBI has 21 attributes in total, it may benefit the use of attribute encodings. On the other hand, we observe attribute encoding performing worse on CORD-19 and Drugs. In contrast to ChEBI, the local sources for both Drugs and CORD-19 contain one long textual field with few to no other attributes. Thus, the addition of attribute encoding devolves into simply adding the same continuous vector onto most term embeddings.

Overall, our results illustrate that even light-weight fine-tuning methods can effectively adapt encodings to domain-specific data. Since both ChEBI and CORD-19 are concerned with specific biomedical domains, they likely contain vocabulary which was rarely seen during the LLM's pre-training. We find that prefix tuning enhances performance over both of these datasets. Despite this, *NoLLM* still outperforms *LLM (PT)* over ChEBI. This may imply that more aggressive fine-tuning strategies are required. However, these strategies might require too much feedback. Given the online nature of our problem, it would likely be more beneficial to combine prefix tuning with other unsupervised methods (e.g., using LLMs pre-trained over domain-relevant corpuses).

5 Related Work

Pay-as-you-go Data Integration. Researchers have proposed pay-as-you-go integration systems that rely on user feedback [11, 34, 21]. Some systems use pay-as-you-go methods to construct a unified schema and query interface over multiple databases [34]. The developer of this system uses available schema-mapping and record-linking tools to explore the schema and content of datasets, which requires access to the entire content of external dataset. We, however, do *not* have access to the entire content of external dataset.

Data Discovery and Augmentation. Given a query table as input, data discovery methods seek related tables within a large pool of tables (crawled from web, data lakes, companies with many tables across multiple data sources) quickly [33, 36, 3, 9, 12, 26]. We, however, seek external information relevant to each local entity. They often preprocess candidate corpora by building indexes across (external) tables. This requires access to the full meta-data and content of external tables, which we do not have in our setting.

Deep Web Crawling & Querying. Web crawlers aim at extracting information stored in external data sources to organize it for future use (e.g., search [22, 30, 35, 7, 20]). Many Web data sources can be accessed only via forms (i.e., *deep Web*). Researchers have proposed techniques that find a minimal set of queries to crawl all tuples in these data sources [22, 7]. As opposed to our setting, they do not consider the notion of relevance to a given entity. Some systems provide a unified query interface over multiple Web forms so users can query these sources via a single interface [35, 7]. They must preprocess these forms to translate queries across them. Our system, however, find information relevant to local entities over keyword query interfaces. It also does not perform any preprocessing to understand the query answering methods of the external sources.

Keyword Query Formulation. Researchers have proposed methods to automate keyword query formulation without writing complicated source-specific programs [30]. However, these methods assume that the external query interface is perfectly accurate and does not return any non-relevant answers, which is not usually true [23, 19]. They do not consider the issue of data heterogeneity and thus lack the ability to adjust their query formulation to account for it. The goal of these methods are also different as they aim to find information related to an entire dataset rather than to an entity.

References

[1] Ted T. Ashburn and Karl B. Thor. 2004. Drug repositioning: identifying and developing new uses for existing drugs. *Nature Reviews Drug Discovery* 3, 8 (2004), 673–683.

[2] Christopher Buss, Jasmin Mousavi, Mikhail Tokarev, Arash Termehchy, David Maier, and Stefan Lee. 2023. Effective Entity Augmentation By Querying External Data Sources. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3404–3417.

[3] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 1001–1012. `https://doi.org/10.1109/ICDE.2018.00094`

[4] Carlos Cunha, Azer Bestavros, and Mark Crovella. 1995. *Characteristics of WWW client-based traces*. Technical Report.

[5] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of Data Integration* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[6] Xin Luna Dong and Divesh Srivastava. 2013. Big Data Integration. *PVLDB* 6, 11 (2013).

[7] Eduard C. Dragut, Weiyi Meng, and Clement T. Yu. 2012. *Interface Understanding Deep Web Query and Integration*. Morgan & Claypool Publishers.

[8] Philipp Dufter, Martin Schmitt, and Hinrich Schütze. 2022. Position information in transformers: An overview. *Computational Linguistics* 48, 3 (2022), 733–763.

[9] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2021. COCOA: COrrelation COefficient-Aware Data Augmentation. In *International Conference on Extending Database Technology*.

[10] National Science Foundation and National Institutes of Health. 2021. Smart Health and Biomedical Research in the Era of Artificial Intelligence and Advanced Data Science (SCH). `https://www.nsf.gov/pubs/2021/nsf21530/nsf21530.htm`

[11] Michael J. Franklin, Alon Y. Halevy, and David Maier. 2008. A first tutorial on dataspaces. *PVLDB* 1, 2 (2008).

[12] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. METAM: Goal-Oriented Data Discovery. In *IEEE 39th International Conference on Data Engineering (ICDE)*.

[13] Behzad Golshan, Alon Y. Halevy, George A. Mihaila, and Wang-Chiew Tan. 2017. Data Integration: After the Teenage Years. In *PODS*.

[14] Felix Gräßer, Surya Kallumadi, Hagen Malberg, and Sebastian Zaunseder. 2018. Aspect-based sentiment analysis of drug reviews applying cross-domain and cross-data learning. In *International Conference on Digital Health*. 121–125.

[15] Janna Hastings, Gareth Owen, Adriano Dekker, Marcus Ennis, Namrata Kale, Venkatesh Muthukrishnan, Steve Turner, Neil Swainston, Pedro Mendes, and Christoph Steinbeck. 2016. ChEBI in 2016: Improved services and an expanding collection of metabolites. *Nucleic acids research* 44, D1 (2016), D1214–D1219.

[16] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. 2003. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB*.

[17] Diane Kelly and Jaime Teevan. 2003. Implicit Feedback for Inferring User Preference: A Bibliography. *SIGIR Forum* 37, 2 (2003).

[18] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 4582–4597.

[19] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[20] Weimo Liu, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. 2014. Aggregate Estimation over Dynamic Hidden Web Databases. *PVLDB* 7, 12 (2014), 1107–1118.

[21] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (Luna) Dong, David Ko, Cong Yu, and Alon Halevy. 2007. Web-scale Data Integration: You can only afford to Pay As You Go. In *CIDR*.

[22] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. 2008. Google's Deep Web Crawl. *PVLDB* 1, 2 (2008), 1241–1252.

[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[24] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[25] Anna Primpeli, Ralph Peeters, and Christian Bizer. 2019. The WDC training dataset and gold standard for large-scale product matching. In *Companion Proceedings of The 2019 World Wide Web Conference*. 381–386.

[26] Aécio Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. 2021. Correlation Sketches for Approximate Join-Correlation Queries. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1531–1544. `https://doi.org/10.1145/3448016.3458456`

[27] Aleksandrs Slivkins. 2019. Introduction to Multi-Armed Bandits. *Found. Trends Mach. Learn.* 12, 1-2 (2019).

[28] Aleksandr Vorobev, Damien Lefortier, Gleb Gusev, and Pavel Serdyukov. 2015. Gathering additional feedback on search results by multi-armed bandits with respect to production ranking. In *WWW*.

[29] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Michael Kinney, Ziyang Liu, William Merrill, P. Mooney, D. Murdick, Devvret Rishi, J. Sheehan, Zhihong Shen, Brandon Stilson, Alex D Wade, Kuansan Wang, Christopher Wilhelm, Boya Xie, Douglas A. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. 2020. CORD-19: The COVID-19 Open Research Dataset. *ArXiv* (2020).

[30] Pei Wang, Ryan Shea, Jiannan Wang, and Eugene Wu. 2019. Progressive Deep Web Crawling Through Keyword Queries For Data Enrichment. In *SIGMOD*. 229–246.

[31] DS Wishart, YD Feunang, AC Guo, EJ Lo, A Marcu, JR Grant, T Sajed, D Johnson, C Li, Z Sayeeda, et al. 2017. DrugBank 5.0: a Major Update to the DrugBank Database for 2018. In *Nucleic Acids res. 2017 Nov 8*.

[32] E. C. Wood, Amy K. Glen, Lindsey G. Kvarfordt, Finn Womack, Liliana Acevedo, Timothy S. Yoon, Chunyu Ma, Veronica Flores, Meghamala Sinha, Yodsawalai Chodpathumwan, Arash Termehchy, Jared C. Roach, Luis Mendoza, Andrew S. Hoffman, Eric W. Deutsch, David Koslicki, and Stephen A. Ramsey. 2021. RTX-KG2: a system for building a semantically standardized knowledge graph for translational biomedicine. *bioRxiv* (2021). `https://www.biorxiv.org/content/early/2021/11/01/2021.10.17.464747`

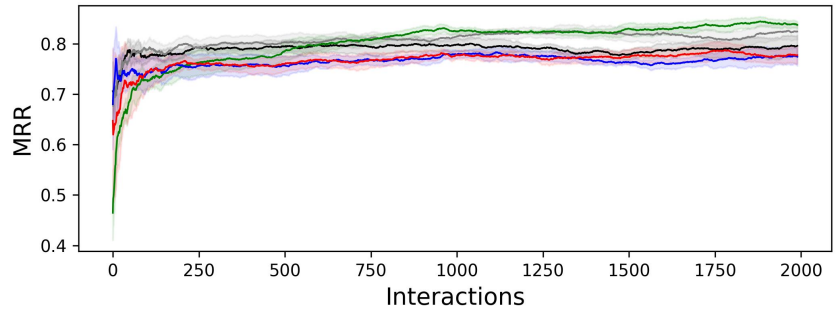
[33] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 97–108. `https://doi.org/10.1145/2213836.2213848`

[34] Zhepeng Yan, Nan Zheng, Zachary G Ives, Partha Pratim Talukdar, and Cong Yu. 2013. Actively soliciting feedback for query answers in keyword search-based data integration. *PVLDB* 6, 3 (2013).

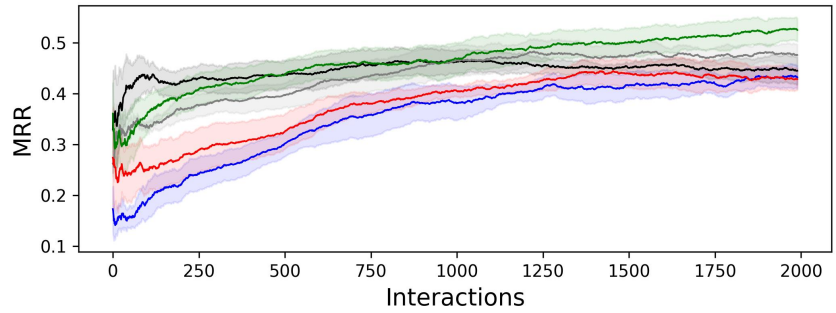
[35] Nan Zhang and Gautam Das. 2011. Exploration of Deep Web Repositories. *PVLDB* 4, 12 (2011).

[36] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 847–864. `https://doi.org/10.1145/3299869.3300065`

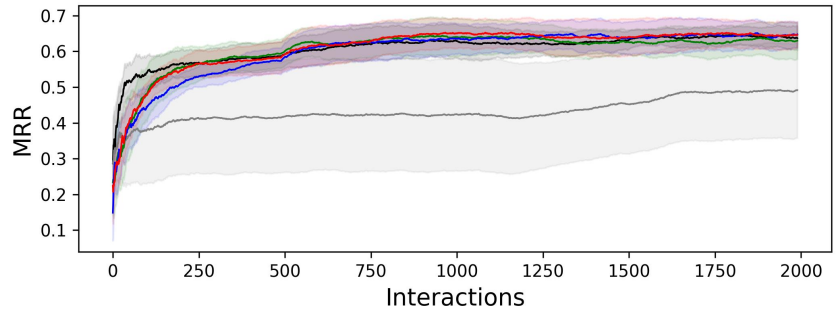
Appendix A Comparison with four keywords



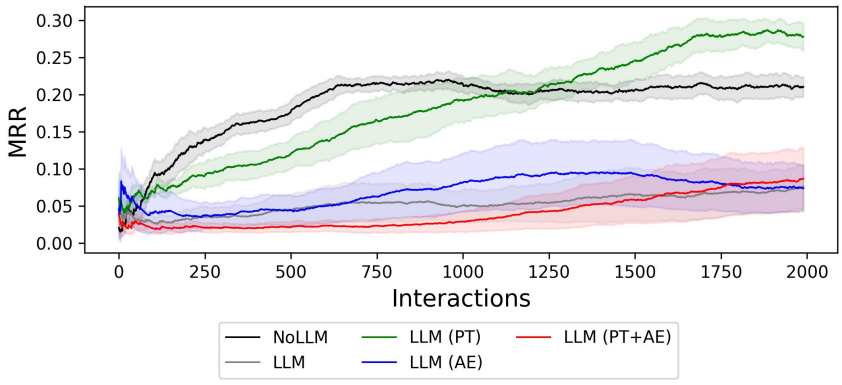
(a) Drugs



(b) WDC



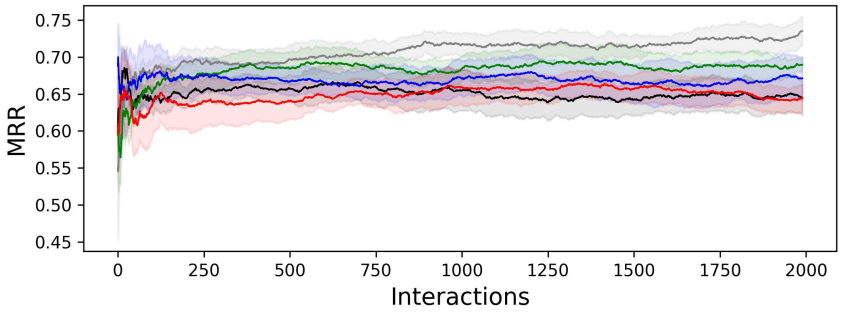
(c) ChEBI



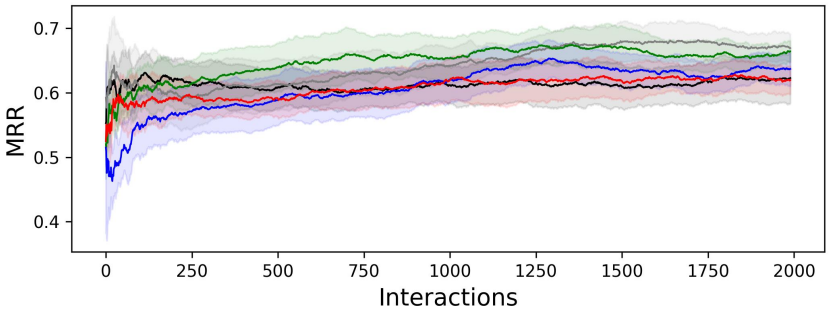
(d) CORD-19

Figure 3: Comparison of different configurations of the LLM model and *NoLLM* for $\ell = 4$. *LLM* is the base model whereas *LLM (PT)* uses prefix tuning, *LLM (AE)* uses attribute encoding, and *LLM (AE+PT)* uses both techniques.

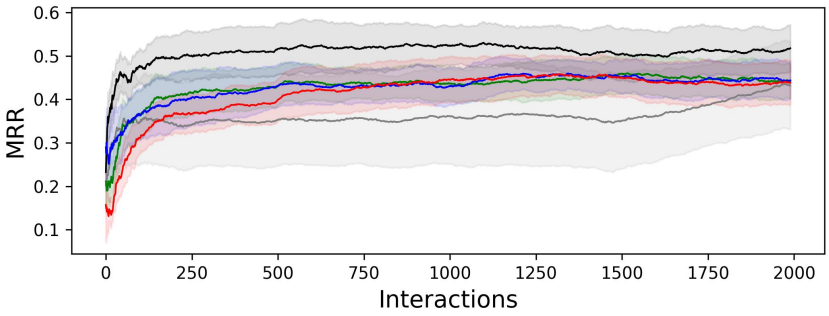
Appendix B Comparison with sixteen keywords



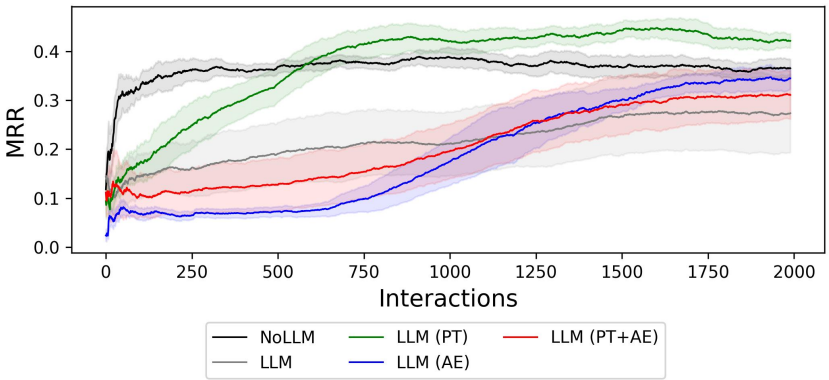
(a) Drugs



(b) WDC



(c) ChEBI



(d) CORD-19

Figure 4: Comparison of different configurations of the LLM model and *NoLLM* for $\ell = 16$. *LLM* is the base model whereas *LLM (PT)* uses prefix tuning, *LLM (AE)* uses attribute encoding, and *LLM (AE+PT)* uses both techniques.