Effective Entity Augmentation By Querying External Data Sources

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 and enrich 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 extract relevant information for each entity. This approach is challenging as many data sources contain numerous tuples, only a small fraction of which may contain entity-relevant information. Furthermore, different datasets may represent the same information in distinct forms and under different terms (e.g., different data source may use different names to refer to the same person). In such cases, it is difficult to formulate a query that precisely retrieves information relevant to an entity. Current methods for information enrichment mainly rely on lengthy and resource-intensive manual effort to formulate queries to discover relevant information. However, in increasingly many settings, it is 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. Our empirical evaluation shows that our approach learns accurate strategies to deliver relevant information quickly.

PVLDB Reference Format:
Christopher Buss, Jasmin Mousavi, Mikhail Tokarev, Arash Termehchy, David Maier, and Stefan Lee. Effective Entity Augmentation By Querying External Data Sources. PVLDB, 16(11): 3404 - 3417, 2023.
doi:10.14778/3611479.3611535

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/bussch/QueryBasedEntityAugmentation.

1 INTRODUCTION

There is a recognized need to collect and connect information from a variety of data sources [14, 18, 23]. As an 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 [49]. The main focus of this project is to *repurpose* current drugs to treat or mitigate the symptoms of diseases for which there is insufficient time or resources to develop effective treatments (e.g., new or rare diseases) [2]. Biomedical researchers often have some local dataset of available drugs (e.g., a dataset of *FDA-approved uses* of drugs). Given a drug in the local dataset, a researcher usually needs to query external data sources to find additional information about the drug (e.g., its *off-label uses*).

Due to a lack of access or resources, external information often must be retrieved through querying [14, 47]. Many data sources 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, as many biomedical data sources are available only via query APIs, 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 [12, 15] such that one needs to tailor their query to specific external data sources. Figure 1 illustrates a case where users have a local dataset of FDA-approved uses of drugs, named *FDA-Approved Uses*, and would like to query an external data source that contains the off-label uses of those drugs, named *Off-Label Uses*. 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*. Because of heterogeneities, one may not know how to query for a specific external entity prior to investigating the content and structure of the data in the external source. Consider a biomedical researcher who seeks additional information about the drug *Zoloft* in their local dataset. Since they are only aware of the structure and content of their local dataset, they query the external data source for *Zoloft*, but that elicits no results. They try again using a much 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 (i.e., contain information about drugs that are not *Zoloft*). 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.

Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611535

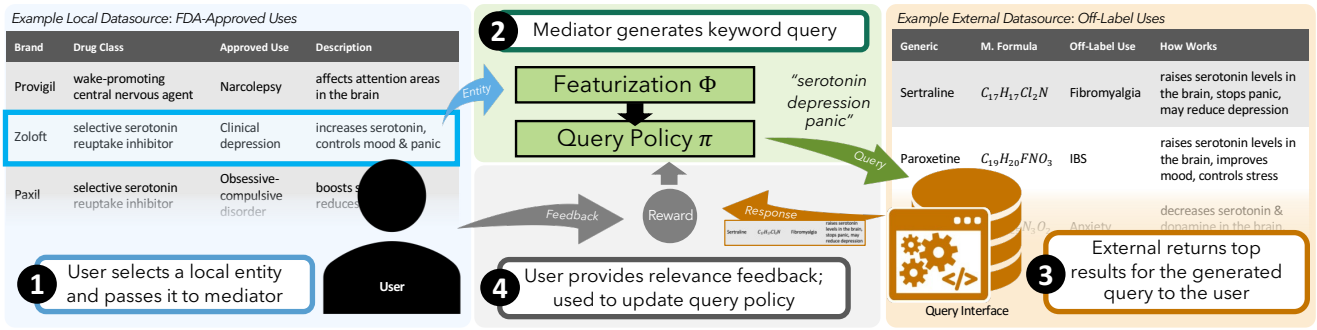


Figure 1: An example of our framework for a single user and single external data source. The user selects (by query, GUI, etc,.) the local entity *Zoloft*. The mediator uses its learned query policy to extract the relevant entity (*Sertraline*) from the external source. The user provides relevance feedback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

Manually querying for specific external entities takes too much time and financial resources. Continuing our example, if the researcher needs additional information for another drug in their local dataset, they will need to repeat the entire process. Moreover, if they need information from additional external data sources, then the work required to query for each drug is greatly exacerbated. Furthermore, any other researcher with a similar information need 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 to retrieve relevant records to a given local entity. Each program implements a set of manually written rules specific to its external source. These rules *cannot* necessarily be reused across data sources. Thus, the mediator requires 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 these programs and investigating the content and structure of every external source to ensure that the programs formulate appropriate queries.

In this paper, we examine methods for *learning 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. Our mediator learns to revise its query policy and improves its performance using the user's feedback.

An alternative to this online learning paradigm is to use offline training data to learn query formulation but collecting and labeling such data still requires considerable manual effort from domain experts [14]. Particularly, it is challenging to gather useful training data from external sources. The data collection/labeling might need to be repeated as the external datasets evolve. In many domains (e.g., drug repurposing for emerging viral diseases), users cannot wait long to prepare offline training data.

Of course, online learning of query policies comes with its own set of challenges. First, the mediator should learn to formulate reasonably accurate queries over external sources early on. We assume the mediator must be effective in the *short run* so users will continue to provide feedback. It is particularly difficult to meet this goal over large local or external datasets as the amount of required feedback for accurate learning generally grows with the number of entities. Second, the mediator should improve its querying policy and increase the effectiveness of its results in the *long run*. Online learning literature indicates that a policy that is effective in the *short run* (i.e., meets the first challenge) might not be accurate in the *long run* as it might become biased to early observations or decisions that do not deliver accurate results in the *long run* [42]. Third, due to lack of prior knowledge about the precise content and structure of the relevant external information, the number of candidate queries for a local entity might be enormous. This *large search space* makes finding effective queries difficult.

Due to the wide-spread use of keyword query interfaces over external sources, we develop online learning methods for formulating keyword queries. There are systems for automatic keyword query formulation, but they assume returned results are always relevant, which is not usually true and is the challenge that we address [47] (see Section 8). Our contributions are as follows:

- We present a framework for on-demand collection of relevant external entities only accessible via query interfaces (Section 2).
- We define the problem of online query-policy learning within the context of the aforementioned framework (Section 3).
- We present a method that learns a separate query policy for each individual local entity. We show that this approach does not scale to large local datasets as it might require a great deal of user feedback (Section 4).
- We propose an entity-conditional method that learns a query policy jointly over all local entities. This approach significantly reduces the amount of user feedback required to learn effective query policies. To overcome representational heterogeneity across the local and external sources, we propose techniques to use features and keywords from the external results in our model and queries, respectively (Section 5).
- If the local dataset contains many diverse entities, it might not be possible to learn effective queries for many entities using a shared model. Hence, we propose an approach that gradually

replaces a shared model with entity-specific ones based on the effectiveness of the shared model. The resulting models will retain the desirable properties of the shared model in the *short run* and learn more effective queries in the *long run* (Section 6).

- We explore whether the broad language understanding capabilities of state-of-the-art large language models can improve query generation. We train a neural network over features extracted by large-scale pretrained Longformer [5] and LLaMA [44] models to serve as our query policy (Section 6).
- We perform extensive empirical studies using six pairs of real-world datasets from different domains, including biology, products, and news. Our studies indicate that our proposed methods learn reasonably effective queries quickly and improve their accuracy in the long run over large datasets (Section 7).

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" depends on the domain. For example, a clinical trial is relevant to the drug that it concerns. For notational convenience, we assume only 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 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 $\mathcal{Q}$. $\mathcal{Q}$ is a subset of the queries accepted by the external query interface. 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. 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 content in `Brand` and `Approved Use` are likely unique to the local dataset. Given this observation, assume the mediator's policy ignores terms from `Brand` and `Approved Use` and prefers terms from `Drug Class` and `Description`. Thus, this policy 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 inverse 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 [37]. 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 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. Unless otherwise noted, we assume that P(.) is uniform.

Optimal Policy. Optimal policy delivers the maximum effectiveness across the entire local dataset. More precisely, the optimal policy π^* for local dataset $\mathcal{E}$ and external dataset D is $\operatorname{argmax}_{\pi} F(\mathcal{E}, D, \pi)$.

Problem Statement. Given local dataset $\mathcal{E}$ and external dataset D, the mediator seeks to find the optimal policy π^* from the set of all possible policies. Finding the optimal policy requires a sufficient understanding of both the representation of entities in $\mathcal{E}$ and how query interface over D answers queries.

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 [14]. 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 LEARNING QUERY POLICY PROGRESSIVELY

In our online approach, the mediator refines its querying policy over time as users provide feedback on the effectiveness of its queries. External relevant information would be presented to the user *on-demand* as they identify entities of interest in the local dataset [35]. The mediator queries external information relevant to the entity of interest using its current policy, presents the results to the user, and collects their feedback on the quality of the results. The mediator may then use the collected feedback to revise and improve its policy to produce progressively better queries and results.

Though our realized system would depend on real-user interaction, we focus on the fundamental question of whether effective querying policies can be learned online and reserve user studies for future work. Thus, we assume a simple user interface and feedback scenario. After the mediator gathers the external results, they are returned to the user. Users can inspect the external results and provide feedback to the mediator. The feedback may be explicit (click-through [45]) or implicit (skipping results [31]).

Our approach is meant to provide users with an additive, non-disruptive experience that improves over time as they provide feedback. Users may interact with the local data source as they normally would and either leverage the external results or ignore them altogether. The mediator learns from the collective feedback of all users, so no single user bears the full responsibility of training it. Thus, as long as some users provide feedback, the mediator can improve, providing progressively better external results for each local entity, for all users of the system.

3.1 Objective and Challenges

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: $\pi_{t+1} \leftarrow update_\pi(f_t)$

Algorithm 1 describes the mediator's general procedure for our online query-policy learning approach. The mediator is involved in a series of *interactions* with the external data source D and a group of local data source users. An interaction at time t is initiated by sampling a local entity for augmentation. How e_t is sampled can reflect one 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 $\mathbf{d}$ as explained in Section 2. Since query interfaces often enforce top-k constraint on their returned results [16, 29], we assume that $|d| \leq k$. The mediator presents $\mathbf{d}$ to the user and evaluates its effectiveness f_t using user's feedback, where f_t depends on the unknown qualities of the external data source (how it ranks and returns results relative to q_t). The mediator uses L_t to update its current policy and find progressively more effective and eventually optimal policies.

Regret. The objective of Algorithm 1 is to find *optimal* policies *quickly*. Thus, it is not enough that Algorithm 1 eventually find the optimal policy: a successful method must also return *reasonably effective* results as it searches for said policy. We use the concept of *regret* to formalize this property. Regret is often used to evaluate the performance of online learning algorithms [42]. Let the policy of Algorithm 1 at time t be π_t. The regret after T interactions is

$$R(T) = T \times F(\mathcal{E}, D, \pi^*) - \mathbb{E}\left[\sum_{t=0}^{T} f(X(e_t), D[\pi_t(e_t)])\right] \quad (1)$$

where expectation is computed over the probability of choosing entities and policies at time t. Regret aggregates the difference between the effectiveness of results delivered by the algorithm's policy and the one of the optimal policy in interactions $0 \leq t \leq T$.

Problem Statement. The problem of online learning of querying policy is to find policy(s) π_t, $0 \leq t \leq T$, that minimize $R(T)$.

Challenges. The sooner the algorithm finds the optimal (or a relatively effective) policy, the less its regret. To meet this challenge, a mediator must accomplish two goals:

1. Balance Exploration and Exploitation. As the mediator neither knows the content of entities in the external source nor the ranking method used by its query interface accurately, finding effective policies requires searching the space of all policies. To minimize regret, the mediator must search the policy space *intelligently*: if it *exploits* the best query found thus far, it may ignore queries that are more effective; if it strictly *explores* until it has found the optimal query for each entity, then it will accumulate a large amount of regret in the process since, many queries are likely ineffective. Thus, we design methods that have the mediator balance both exploiting what it knows and exploring the space of policies for better queries.

2. Maintain Users' Engagement. Policy search requires user feedback, thus the mediator must also keep users engaged while searching. Due to the large number of local entities, large set of possible queries, and the different representations of information in the local and external, it might not be possible to find an effective policy in just a few interactions. Nevertheless, if effectiveness remains relatively low for an extended period, users might become discouraged and abandon the system. It is assumed that users have some tolerance for poor policies during their initial use of the system granted that more effective policies are eventually found. But users may stop providing feedback if the policies continue to perform poorly even after a modest amount of feedback is provided.

Hence, our objective is to design methods that minimize regret across two phases: the *short run* and the *long run*. The short run is the first T_{sr} interactions. Within the short run, the mediator must find a policy such that $R(T_{sr}) \leq R^*(T_{sr}) + \epsilon$ where $R^*(T_{sr})$ is the regret of $\pi*$. Both n and ϵ depend on user tolerance for sub-optimal results. As factors that lead to user abandonment are complicated [13], we do not establish an explicit ϵ range for tolerance and note that it is specific to the domain and the end-users. User tolerance may be high if the alternative to providing feedback is onerous (i.e., hand-crafting the queries themselves in a complex domain). Furthermore, since the mediator leverages feedback from all users, T_{sr} also depends on the number of users. For example, for small ϵ, $T_{sr} = 500$ is likely unreasonable for a system with 5 users; however, it may be reasonable for a system with 50 users. The number of users as well as their expectations may change over time, thus both

T_{sr} and ϵ also change over time. Success in the *short run* requires *expedient sufficiency*: the model must at least meet minimal user expectations to retain usefulness and continue gathering feedback.

In the long run, the goal is to minimize regret as $t \to \infty$. At this stage, user abandonment is no longer the primary concern. Rather, the challenge is to maximize user satisfaction. In this stage, models should be concerned with further minimizing their regret.

Keyword Query Interface and Results. A keyword query q is a finite string comprised of *terms* (keywords). The number of terms in each query is its *length*. We indicate that term k appears in query q with $k \in q$. Where appropriate, we denote the set of queries of length ℓ using Q^{ℓ}. To save resources, query interfaces might limit the number of terms in their input queries. For example, Yelp!'s Fusion API will return no results if more than 8 terms are used and Google.com limits queries to 32 terms. These limits are usually recorded in the query-interface documentation. We assume that all queries submitted to an external data source D have a given fixed length. We explain how to relax this assumption in Section 7.2. Though they are relatively simple, keyword queries present a unique set of challenges. Unlike formal query languages, such as SQL, keyword queries are inherently vague [29, 37]. Moreover, limits on the length of keyword queries often reduce the flexibility of formulating informative queries.

3.2 Managing the Policy Space

The space of potential policies is correlated with the size of $\mathcal{Q}$ (i.e., the co-domain): the larger $\mathcal{Q}$ is, the more ways that local entities can be mapped to queries. Furthermore, treating each query in $\mathcal{Q}$ as producing unique results from the external data source is problematic for two reasons. First, queries containing the same keywords should produce somewhat similar results. Second, assuming each query produces unique results makes evaluating policies more difficult. Under this uniqueness assumption, two policies that send similar queries for any given entity will still be considered *entirely different* if none of their output queries are *exactly* the same. To make our policy space more manageable, we both prune $\mathcal{Q}$ and take a term-centric approach when mapping entities to queries.

Entity-Specific Pruning. For any input local entity, only a small subset of $\mathcal{Q}$ will be useful. Though $\mathcal{Q}$ could be manually pruned using domain expertise, we opt for general methods that do not require this extra attention. In order to remove a large fraction of ineffective queries, we limit the terms considered to only those that *effectively express* the given local entity. We define an entity-dependent co-domain $\mathcal{Q}_e \subseteq \mathcal{Q}$. Let $L(e)$ be the set of terms that make up the content of e. That is, if term k appears in the local entity e, then $k \in L(e)$. For every entity $e \in \mathcal{E}$, $\mathcal{Q}_e$ contains every possible concatenation of distinct terms from $L(e)$. In other words, the mediator maps e to a keyword query q by concatenating a subset of the terms in the local tuple $L(e)$. $L(e)$ might not contain the terms necessary to form an optimal query, but given that relevant entities from related domains often share terms, it is reasonable to believe that an effective query could still be found in many cases. As we will discuss in Section 5.3, $\mathcal{Q}_e$ can be expanded to include other terms and more effective queries during the interaction.

Term Effectiveness. In order to generalize its knowledge across policies, the mediator evaluates the effectiveness of keyword queries based on their content. We take advantage of the fact that many keyword queries overlap with respect to the terms they contain. Intuitively speaking, if a subset of terms is shared across effective queries for some entity e, then it is likely that same subset that has significantly influenced each query's effectiveness. Thus, a desirable policy would map e to queries containing that same subset. Following this logic, we consider methods that track the effectiveness of terms used within keyword queries rather than the effectiveness of whole keyword queries. Furthermore, we assume that terms within keyword queries are independent. This assumption allows our policies to construct output queries term-by-term based on each term's effectiveness. We call the set of terms $k \in L(e)$ the *candidate terms* for e because it consists of all of the possible terms that could be selected one-at-a-time to form $\mathcal{Q}_e$.

4 ENTITY-LEVEL LEARNING

A natural approach to learn queries is to maintain a model for each local entity. The policy for the whole dataset would be the union of each entity-specific model. Precisely, the mediator maps entities $e \in \mathcal{E}$ to queries $q \in \mathcal{Q}_e$ by selecting candidate terms based on their effectiveness in previous queries for the *same* entity.

At time t and given entity e_t, the mediator could calculate the expected accuracy of including a candidate term $k \in L(e_t)$ within a query for e_t over D based on the previous queries used for e_t.

$$\mathbb{E}[k] = \frac{1}{\sum_{j=0}^{t-1} I(k,j,t)} \sum_{j=0}^{t-1} f(X(e_j), D[q_j]) I(k,j,t) \qquad (2)$$

where $I(k,j,t)$ is 1 if $k \in q_j$, $k \in X(e_j)$, $e_j = e_t$, and 0 otherwise. The expected reward for a candidate term $k \in L(e_t)$ would be the mean of the rewards for those queries generated for e_t in previous interactions $[0, t-1]$ where k exists in both the generated query q_j and the content of the relevant external tuple $X(e_t)$. If a term did not appear in $X(e_t)$ then it very likely had no positive affect on extracting $X(e_t)$, thus the reward associated with including k in the query is assumed to have been 0. After calculating the expectation of each candidate term, the mediator could then greedily generate a query with the greatest mean expected reward by selecting ℓ terms with the highest expected rewards: $q_t = \underset{q \in Q_e^{\ell}}{argmax} \frac{1}{\ell} \sum_{k \in q} \mathbb{E}[k]$.

However, since these term estimates are based only on previously sent queries, a mediator that strictly acts in a greedy manner will fail to explore the space of possible queries. As discussed in Section 3.1, focusing solely on exploiting its current knowledge will prevent the mediator from finding better policies.

Multi-Armed Bandit Formulation. Balancing exploration and exploitation of candidate terms online can be modeled as a *Stochastic Multi-Armed Bandit* (MAB) problem. The goal of MAB problems is to learn, from a set of candidate *arms* with unknown reward distributions, the arm with the largest mean reward [42]. In each round, an MAB algorithm picks an arm and observes its reward. MAB algorithms aim at minimizing regret [42]. There are asymptotically optimal algorithms, called *Upper Confidence Bound* (UCB), that estimate confidence bounds on the expected reward of each arm and pick the arm with the largest upper limits [3, 30].

It is, however, challenging to scale the entity-level approach to large datasets with many entities. Because each entity has its

own model, each entity also represents a distinct learning problem. The asymptotic amount of feedback required to learn an effective policy would be approximately linear in the number of entities in the dataset: users may have to wait for thousands if not hundreds of thousands of interactions to get relevant information in a local dataset with hundreds of thousands of entities.

5 DATASET-LEVEL LEARNING

To reduce the amount of feedback required to find an effective query policy, we consider an entity-conditional model of query quality that is learned jointly over all entities. We share learning across entities while recognizing the distinct characteristics of each entity for generating its queries. Like the entity-level model, each *arm* in this approach is a candidate term. However, the reward (i.e., effectiveness) of using each term varies based on the local entity (i.e., context) for which the term is used. Since the reward of each arm depends on its context, we cast our online query formulation problem as a *contextual multi-armed bandit* (contextual bandit) problem [42]. The input of a contextual bandit problem is a finite set of arms *and contexts*, where at each round only one of the contexts is active. An arm might be used in different contexts. The reward of each arm depends on the context in which it is used.

5.1 A Linear Bandit Approach

LinUCB extends the idea of UCB to the contextual bandit problem. It assumes the reward of each arm to be a linear function of some vector representations of the arm and the current context [10]. The regret of LinUCB is of order $\sqrt{Td\ln^3 \frac{KT\ln T}{\delta}}$ where T, d, and K are the number of trials, dimension of the vector representation of arms and contexts, and number of arms, respectively. It provides an asymptotic regret close to the lowest possible one, $O(\sqrt{Td})$ [10]. We use LinUCB to learn the query model.

More precisely, we assume that the expected reward for each term $k \in q_t$ is a linear function f_t parameterized by an unknown weight vector $w^* \in \mathbb{R}^d$ as $A_t(k, e_t) \cdot w^* + \epsilon_t$, where $A_t(k, e_t) \in \mathbb{R}^d$ is a vectorized representation of term k and entity e_t, and ϵ_t is Gaussian noise with mean 0 and variance 1 (i.e., $\epsilon_t \sim \mathcal{N}(0, 1)$). Our goal is to learn the weight vector w^* online. In this approach, feedback on the effectiveness of each query is used to update the parameters of the reward function of all terms of all queries. Hence, the learned function can also be used to estimate the reward of never-before-used candidate terms. Training this model amounts to leveraging user feedback to find a set of weights w that most accurately model the true reward function.

Example 4. Assume the vectorized representation of a term $k \in L(e_t)$ indicates the attribute(s) for which k appears in the content of e_t. Since terms from `Brand` are unlikely to yield any matches from a dataset that only knows drugs by their generic names (i.e., the external data source in Figure 1), the mediator would quickly learn, irrespective of local entity, that terms from `Brand` do not produce a sufficient enough reward to be used in queries.

Like the entity-level model, the dataset-level model uses a term selection strategy that balances exploration and exploitation. Let W be the set of all possible weight vectors in $\mathbb{R}^d$. At interaction t, LinUCB constructs a confidence region $C_t \subset W$ that contains w^* with (high) probability of $1-\delta$ using the information from previous interactions. It then picks a candidate term with the largest possible reward over C_t. The larger the observed average reward of a term is and the fewer times it has been tried up to round $t-1$, the larger its maximum possible reward over C_t will be. The degree of exploration is controlled by input parameter $\alpha \geq 0$.

5.2 Representations of Terms & Entities

We represent $A_t(k, e)$ using *lexical*, *distributional*, and *schematic* features of terms. Lexical features are based on a term's word type (e.g., noun or adjective) as indicated by WordNet [38]. The distributional features of terms are based on the properties of terms over the entire local dataset. For example, let *Dataset Frequency* (DF) of a term denote the fraction of entities in the local dataset in which the term appears. *Inverse Dataset Frequency* (IDF) of a term is the inverse of its DF. The IDF of a term quantifies how well that term identifies the entity within the dataset and we use it as a distributional feature in our model. We use a combination of domain-specific (e.g., IDF of a term in the local dataset) and non-domain-specific (e.g., word types from WordNet) features. The non-domain-specific features are meant to capture the general characteristics of terms that are not biased to their domain-specific representations. To capture the context (i.e., the local entity for which a term appears) we include entity-specific features of terms, such as the frequency of k in the content of e and the attribute(s) for which k appears in the content of e. We normalize features, such as frequencies of terms in entities, to ensure that they are comparable across different entities.

5.3 Using External Terms & Features

A local entity and its relevant external entities might share few to no terms. Hence, policies that only consider queries formed from the content of a given local entity may lack the ability to build effective queries for that entity. To address this problem, we propose two methods for expanding the set of candidate terms for certain local entities by *borrowing* terms from entities appearing in external results. We distinguish these two methods based on whether external terms are borrowed based on user feedback and external results (supervised) or just external results (unsupervised).

Supervised Term Borrowing. For a keyword query to extract $X(e)$ from the external dataset, it must contain at least some terms that appear in the content of $X(e)$. Thus, expanding the set of candidate terms for e to include those terms in $X(e)$ would allow for queries that more effectively extract $X(e)$. After the user identifies $X(e)$ within the returned results, the mediator adds the terms in $X(e)$ to its set of candidate terms for entity e. In future interactions, when the mediator is asked to map e to a query, it can use these additional terms to increase the effectiveness of its query. These terms may improve the ranking of $X(e)$ in subsequent interactions.

Unsupervised Term Borrowing. When the mediator lacks the candidate terms to retrieve $X(e)$, we must expand the set of candidate terms with something other than the content of $X(e)$. The added terms should have some relation to the local entity while also reflecting the term distribution of the external dataset. Since the mediator's policy is trained to map e to queries that extract external entities *related* to e, those same external entities may be transitively related to $X(e)$. Thus, these related entities may reveal

additional terms that can be used to retrieve $X(e)$. For example, similar drugs may have similar biological effects, meaning similar terms in attributes like `How Works` in Figure 1. Unsupervised term borrowing could saturate the candidate set with unrelated terms. Thus, we take a conservative approach and only borrow terms from the external entity in the top position of the returned results. Furthermore, we only apply unsupervised term borrowing to local entities that meet the following criteria: 1) $X(e)$ has not been extracted yet, and 2) *a sufficiently large fraction* of candidate terms from the content of e have been tried. Setting the aforementioned fraction to a large value (e.g., 100%) might delay borrowing and deliver ineffective results for a relatively long time. Using smaller values for this fraction might lead to borrowing terms too quickly and before the mediator policy has collected enough information about potentially related entities to $X(e)$. In our experiments, we set this to a value between the two extremes (70%) (Section 7.4).

External Features. For our candidate terms, we use *external features* that reflect how effectively those terms can pinpoint entities over the external source. For example, the frequency with which a term appears in an external entity might indicate how effectively this term can pinpoint the entity in the external source. Since the mediator does not have access to the entire external dataset, we use only the external features that can be computed during querying the external source using the returned results (e.g., frequency of terms in the returned (relevant) entities). We use external features for both borrowed external and local terms.

6 OVERCOMING ENTITY DIVERSITY

The dataset-level model learns a linear approximation of term effectiveness over all local terms using relatively few features. Thus, it should converge to one of its most effective policies with few interactions. However, the dataset-level model may lack the power to represent the more nuanced properties of terms that determine their effectiveness. Thus, its most effective policies will likely be less effective than the entity-level model's. As large datasets often contain many diverse entities, the dataset-level model may lack the capacity to sufficiently estimate rewards of candidate terms for all entities. Thus, we propose methods that can approximate term reward quickly while having greater representational power.

6.1 From Dataset-Level to Entity-Level Learning

In contrast to the dataset-level approach (Section 5), the entity-level approach (Section 4) would eventually result in a (near-)optimal policy given *many* interactions. To combine the strengths of these methods, we introduce a two-stage approach called *Hybrid* that quickly learns a shared model and then leverages this model to warm-start entity-specific learning. This method combines the benefits of shared query learning (i.e., keeping users engaged by learning a relatively effective model quickly) and the entity-level query-learning models (i.e., learning an effective model for each entity in the long run). It starts with learning the shared query model using the approach explained in Section 5. It then switches to entity-specific models for entities that the shared model cannot find effective queries for (e.g., cannot return any relevant answers) after trying the queries learned by the shared model for those entities sufficiently many times. Dividing the input space of local entities across different models can increase overall performance by reducing under-fitting. By introducing entity-specific models, *Hybrid* not only removes outliers that its shared model cannot fit to but also fits dedicated entity-specific models to those same outliers.

We, however, modify the entity-level method proposed in Section 4 to 1) speed up its learning and 2) enable it to leverage the available information in the learned shared model. As candidate solutions in the entity-level model are terms, it may take too long to learn effective policies for each entity. Thus, instead of using the entity-level model, we use LinUCB to find accurate queries for each selected entity in entity-level learning. We represent each term in the entity as a vector of features used to train the shared model. We train a weight vector w_{shared} until some point and then initialize the space of solutions for each entity-specific model for entity e, w_e, based on all previous feedback on queries sent for e. Additionally, we subtract all previous feedback on queries sent for e from w_{shared}. This way, we warm-start w_e with only the most relevant feedback and reduce the effect that outlier e had on w_{shared}. One might use additional entity-specific features (e.g., the frequencies of a term appearing in the relevant or non-relevant results for the entity) in the feature vector for each entity-specific model.

Transition Details. *Hybrid* starts with a dataset-level model for all entities and keeps track of two metrics: (1) the MRR in the last two windows of n interactions each, and (2) the last RR observed for individual local entities. For a given local entity, *Hybrid* will switch to an entity-specific model if the dataset-level model has reached capacity (i.e., MRR has not increased between the two windows) and has shown poor performance over the local entity (i.e., the previous RR observed for it is less than threshold β). Hyper-parameter n determines both the amount of feedback the dataset-level model must receive before entity-specific models can be instantiated and the sample size used to assess the dataset-level model's recent performance. If n is too small, then the mediator will instantiate models for local entities that may have performed well under the dataset-model. Optimally, n should be set according to the amount of feedback it takes for the dataset-level model to reach its capacity. The threshold β can be thought of as the lower-bound on acceptable performance. As n shrinks (grows) and β grows (shrinks), *Hybrid* approaches pure entity-level (dataset-level) learning.

6.2 Language Model Based Query Learning

The simplicity of a linear model is attractive in online learning since it treats estimation as a convex problem, and if the features used are good predictors of term effectiveness, then a linear model should perform well even with little feedback. However, the linear model's simplicity is also a limitation on its ability to discover more nuanced predictors of a terms's quality. Hence, we also explore using a model that leverages state-of-the-art transformers for richer representations of tuples and keywords. Specifically, we consider large-scale pretrained language models (LMs) LLaMA [44] and Longformer [5]. In contrast to linear models, LMs can encode entire tuples jointly such that individual term quality can be predicted based on keyword representations contextualized on high-dimensional representations of the entire entity. This flexibility comes at the cost of significantly more parameters and non-convex optimization.

Encoding Tuples and Scoring Terms. Given an entity e_t, we concatenate all terms $k_i \in L(e_t)$ into a single string s_t and pass it

through the model after standard byte-pair-encoding tokenization. The output of the model provides a contextualized representation h_i for each input token. Note that the byte-pair-encoding may break candidate terms into multiple inputs or candidate 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: $h_1, ..., h_n = \text{LM}(s_t)$

These representations capture information about each term given the context of the entity. To further enrich these features with dataset-level information, the feature vector from the linear model $A_t(k_i, e_t)$ is concatenated onto each corresponding representation forming $f_i = [A_t(k_i, e_t), h_i]$ where $[\cdot, \cdot]$ denotes concatenation. Vector f_i is then passed through a small fully connected neural network to predict reward r_i for each term. In our setting, r_i is an estimate for reciprocal rank and bounded between 0 and 1.

Selecting Queries and Updating. We apply an ϵ-greedy approach to query formulation [42] — selecting either the next-highest-scoring term or, with probability ϵ, a random term until the desired query length is achieved. Once user feedback is received, the RR for the query is calculated and used to supervise the network. Specifically, the observed RR is recorded 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 50 observed terms. We train the model by stochastic gradient descent with batches of 8 samples from the buffer at each interaction.

Implementation Details. LLaMA consists of 32 layers, 4096 hidden representation size, 32 attention heads, and 7 billion parameters. Longformer consists of 12 layers, 768 hidden representation size, 12 attention heads, and 125 million parameters. They both use pretrained models from the Huggingface Transformers library, utilizing their respective byte-pair-encoding tokenizers. Through pretraining, they acquire strong reasoning capabilities about English words and sentences that are frequently used in our examined entities. To train the fully-connected neural network, we use Pytorch's implementation of Adam with default hyper-parameters. We use mean squared error as the loss function.

7 EMPIRICAL EVALUATION

7.1 Experimental Setup

Datasets. We evaluate our methods over a variety of domains using the datasets listed in Table 1. Each dataset contains a local dataset and an external dataset. We include the entity count for each source along with the average number of terms per entity. Every entity in a local dataset has at least one relevant entity in its external dataset, but some external datasets have additional irrelevant entities that can appear in results. For this reason, we also specify the number of external entities that are relevant to at least one local entity.

Both **DrugCentral** and **ChEBI** are derived from datasources used in the NIH project discussed in Section 1. For both, we use *DrugBank* as the local source, which contains comprehensive molecular information about drugs [48]. DrugCentral uses *Drug Central* as its external source. Drug Central stores regulatory information associated with drugs [4]. *ChEBI* broadly tracks molecular entities used to intervene in the processes of living organisms [27].

WDC is derived from the non-normalized English *WDC Product corpus*, containing products scraped from many different sites [39]. **CORD-19** contains records about scientific papers and research related to COVID-19 [46]. We split CORD-19 into two separate sources: one containing abstracts (local) and one containing the remaining attributes (external). **Drugs** contains drug reviews from *Drugs.com* (local) [24] and descriptions of the same drugs scraped from *Wikipedia* (external). **News** is derived from a dataset covering 38 major mass-media companies [25]. It contains titles and summaries of articles (local) and the articles themselves (external).

Perfect MRR ($\ell = 4$) in Table 1 indicates the best Mean Reciprocal Rank (MRR) achievable for each dataset when using queries of length 4. This metric was calculated offline by searching the entire space of queries for each local entity and keeping track of the highest achievable RR. Due to the runtime required, we have calculated this metric over 5% subsets of each local dataset except for CORD-19. Due to CORD-19's large size and high number of candidate terms per local entity, we could only calculate this metric over a 544 local entity sample. Though it is unrealistic to assume that anything but computationally expensive offline algorithms can achieve this performance, we include it as an indicator of dataset difficulty and term overlap.

Interactions. We simulate a series of interactions between a mediator and a query interface. Each interaction is initiated by sampling a local entity. Given the local entity, the mediator generates a query of length ℓ and submits it to the external data source. The external data source returns its top-20 results based on a static ranking function (BM25), which we implement using the Whoosh package [9]. The query is assigned a reward based on the reciprocal rank of the top-relevant result using simulated feedback (i.e., ground truth).

Evaluation Metric. To mitigate variations across entities, we compute MRR as a sliding average over the previous 500 interactions. Each graph plots this average against the current interaction. We report MRR for our methods as the average of five runs each comprising 2000 interactions. Error bands are included around each average (line) to show a 95% interval for standard error across the runs. Due to lack of space, we report illustrative examples of trends and complete results are in [7]. We omit the runtime of experiments, but note that much of it is dedicated to external query processing: our models take relatively little time to execute and update.

Hyperparameters. Unless specified otherwise, we treat query length as a hyperparameter and evaluate our methods using $\ell \in \{4, 8, 16, 32\}$. These values reflect limits on real interfaces (see Section 3.1) and illustrate how query length affects policy performance. *Dataset-Level* (Section 5) and *Hybrid* (Section 6.1) use LinUCB as their exploration strategy while*LM-Based models* (Section 6.2) use ϵ-greedy. Both strategies use a hyperparameter to control the extent to which they explore. For LinUCB we use $\alpha = 0.2$ and for ϵ-greedy we use $\epsilon = 0.05$. We evaluate how the degree of exploration affects performance in Sections 7.2 and 7.3.

Static IDF Benchmark. To help contextualize the performance of our methods, we present a naive policy for comparison. *Static IDF* always produces queries using the top-ℓ terms in the content of e based on their *Inverse Dataset Frequency* (IDF). As explained in Section 5.2, IDF, a common measure of term specificity [29], quantifies how unique a term is to an entity within a dataset.

Table 1: Details of datasets used in our evaluation, sorted according to the number of entities within the external source.

dataset	source	attributes	avg. terms	entities	#relevant	Perfect MRR ($\ell = 4$)
DrugCentral	Local	name, description, indication, pharmacodynamics, ...	178	3,475	3,457	0.9971
	External	formula, name, fda_labels, drug_class, active_ingredient, ...	279	4,927		
Drugs	Local	drugName, condition, review	108	13,725	413	0.9822
	External	page_title, wikipedia_summary	168	46,976		
News	Local	title, article_summary	42	30,000	30,000	0.9763
	External	article_content	547	30,000		
WDC	Local	category, brand, prod_title, description, ...	67	57,109	55,247	0.8697
	External	category, brand, prod_title, description, ...	72	55,247		
ChEBI	Local	name, description, indication, pharmacodynamics, ...	178	5,483	5,753	0.8953
	External	status, name, definition, charge, formula, mass, ...	73	189,467		
CORD-19	Local	abstract	305	250,575	250,575	0.8325
	External	sha, source_x, paper_title, doi, pmcid, ...	48	340,826		

7.2 Dataset-Level Model

Our first questions of interest are (1) can *Dataset-Level* outperform the *Static IDF* benchmark?, (2) can *Dataset-Level* find a sufficiently effective policy in the *short run*?, and (3) how does query length affect performance and how can we set it apriori? A defining challenge of a newly deployed mediator is that it will have no experience generating queries for most local entities. To simulate this challenge, we sample local entities uniformly. This setup mimics a difficult learning task where the mediator must generalize what it has learned to novel entities. Figures 2 and 3 show the performance of *Dataset-Level* relative to the *Static IDF Benchmark* (*IDF*). For both models, local entities are sampled uniformly at each interaction. However, since *IDF's* policy does not change, we calculate its MRR over all interactions and present it as a single vertical line.

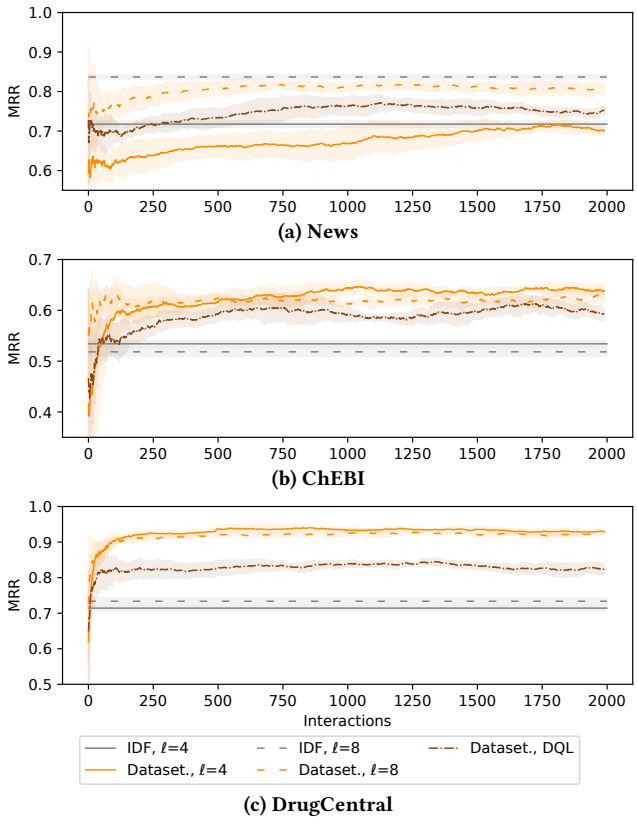
Exploration. We evaluate *Dataset-level* with varying degrees of exploration $\alpha \in \{0, .2, .5, 1, 2\}$ and find that a small degree of exploration (i.e., $\alpha = .2$) provides the most consistent performance, balancing the mixed effects that exploration can have over different datasets. For some datasets, a small degree of exploration provides marginally better performance than no exploration; for other datasets, performance slightly degrades as the degree of exploration increases. The negative effect of exploration is at least partially determined by the structure of the local source. For example, the local source for both CHeBI and DrugCentral has the most attributes of any dataset, resulting in many features per candidate term. However, few features tend to be positively correlated with query effectiveness. As ℓ increases, we find that the effect of exploration is neutralized. This is likely due to the reduced effect of exploratory terms themselves: a set of exploratory terms will have a greater impact on the outcome of a short query than a long query. Furthermore, a higher ℓ forces the mediator to select a greater diversity of terms even when no exploration is used.

***Dataset-Level* quickly finds policies that outperform *IDF*.** Early on, *IDF* achieves higher MRR on most datasets. However, *Dataset-Level* eventually surpasses it, often within the first 100 interactions. One exception to this trend is News (Figures 2a/3a) where the local IDF of terms is correlated with their effectiveness: both the input local entity and its relevant external entity tend to share distinguishing terms. This result is not too surprising, as titles of news articles often share specific terms with their respective content. However, our experiments show that this trait is uncommon. Thus, methods that learn to adapt to particular external sources are likely to have better policies than models that stick to one heuristic of term effectiveness (e.g., IDF of local terms).

***Dataset-Level* with small ℓ has the most reliable performance.** Performance differences between *Dataset-Level* and *IDF* tend to be greatest when ℓ is small. However, there is no general correlation between query length and model performance. In some instances, increasing ℓ improves the performance of both models to the point of convergence (Figure 3a). In other instances, increasing ℓ reduces the performance of both models (Figures 3b/3c). When ℓ is small, models use only those top few keywords with the highest predicted effectiveness; but as ℓ grows, there is a higher liklihood that the models will include noisy terms that reduce the rank of the top relevant entity. Furthermore, long keyword queries may not be accepted by external data sources. Given these findings, more discriminating polices that send few terms are preferable since they neither run the risk of degrading performance nor having their queries outright rejected by the external source.

Dynamic Query Length (DQL). As the best value for ℓ tends to be dataset-specific, we present a method for dynamically altering ℓ based on a model's estimate of candidate terms. Instead of selecting the top ℓ terms, we use nucleus sampling which has been shown to be an effective strategy in natural language generation [28]. The initial candidate term estimates are normalized using the softmax function $\overline{k_i} = \exp(\mathbb{E}[k_i]) / \sum_{k_j \in L(e)} \exp(\mathbb{E}[k_j])$. The top $\overline{k}$ terms are then selected based on a threshold $P \in (0, 1]$ which determines the total probability mass of terms to select. For example, if $P = .8$, then the minimal number of top terms would be selected such that their probability mass exceeds .8. To align with other experiments, we set 32 terms as a hard cutoff point for query lengths.

For reasonable P DQL tends towards optimal query lengths. We evaluate $P \in \{.1, .2, .4, .6, .8\}$ and find that DQL adjusts query size according to input local entities in all cases. Of the values tested for P, we find $[.2, .6]$ to be a reasonable range, thus we include DQL with $P = .4$ within Figures 2 and 3. Intuitively, if the threshold is too low (.1), then informative terms will be excluded; if the threshold is too high (.8), then many noisy terms will be included. Furthermore, it is likely that only a relatively small portion of candidate terms increase query effectiveness. For $P \in \{.2, .4, .6\}$, DQL tends towards the optimal query lengths. For example, for $P = .4$, of all queries sent for CORD-19 (where a large ℓ leads to better performance), at least 50% had a length within $[27, 32]$. Similarly, of all queries sent for ChEBI (where a small ℓ leads to better performance), at least 50% had a length within $[10, 32]$.

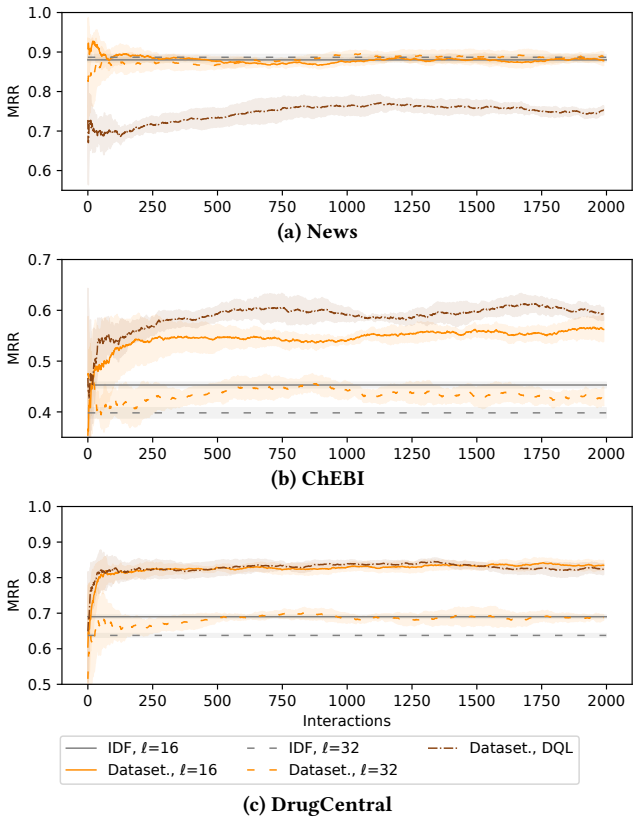


(a) News

(b) ChEBI

(c) DrugCentral

Figure 2: Dataset-Level and IDF (Uniform, 4/8 keys)



(a) News

(b) ChEBI

(c) DrugCentral

Figure 3: Dataset-Level and IDF (Uniform, 16/32 keys)

For reasonable P, DQL is robust to the choice of hyperparameter. Even as P is pushed to the boundaries of the range $[.2, .6]$, we find that DQL maintains the observed behavior. For $P = .2$ and $P = .4$, 50% of queries sent for CORD-19 are within the range of $[13, 21]$ and $[32, 32]$ respectively. For ChEBI, we observe $[4, 28]$ and $[16, 32]$. This indicates that DQL with $P \in [.2, .6]$ is a reasonable choice when we are unsure of the optimal query length.

Averaged over all datasets, *Dataset-level* with $\ell = 4$ finds policies that produce an MRR of roughly 0.5 within the first 250 interactions. We consider this performance to be sufficiently effective in the short run, given such limited feedback. However, it shows little improvement with additional feedback. Thus, it is quick to hit its capacity to account for local entity diversity.

7.3 Overcoming Entity Diversity

In the following experiments, we compare *Hybrid* (Section 6.1) and *LM-Based* (Section 6.2) against *Dataset-Level*. We seek to understand whether our methods can continue to improve their query policies in the *long run*. To more accurately measure *long-run* performance, we adjust our sampling strategy for local entities.

Studies suggest that entity preference follows a near-Zipf distribution $1/i^s$ where i is the rank of the $i'th$ most popular entity and $s \approx 1$ [1, 11, 22]. Thus, users request the $i'th$ most popular entity approximately twice as often as the $(i+1)'th$ most popular entity. Following this evidence, we simulate user preference by sampling local entities from a Zipf distribution ($s = 1$). Since our datasets do not indicate entity popularity, we randomly assign the order of popularity, which is held constant across different models. Figures 4 and 5 show the performance of *Dataset-Level*, *LM-Based*, and *Hybrid* under a Zipf sampling of local entities.

***Hybrid* Hyperparameters**. To avoid the risk of instantiating too many entity-specific models too soon, we use $n = 50$ and $\beta = \frac{1}{15}$.

***Hybrid* improves MRR over *Dataset-Level* in the long run**. To better understand how the addition of entity-specific models affect performance in the long run, we evaluate *Dataset-Level* and *Hybrid* over the same stream of local entities for 10,000 interactions. We compare the MRR of the two methods over three sets of local entities: 1) those that *Hybrid* has initialized *entity-specific* models for thus far (MRR_{es}), 2) those that *Hybrid* uses its *dataset-level* model for thus far (MRR_{dl}), and 3) all local entities encountered thus far (MRR_{all}). Generally, we find that *Hybrid* meets or exceeds *Dataset-level's* MRR_{all}, with the most dramatic improvement on CORD-19 with $\ell = 4$. Within 3,000 interactions, we observe a 1.35 performance increase with *Hybrid* at 0.2443 ($\pm$ 0.03) MRR_{all} and *Dataset-level* at 0.1808 ($\pm$ 0.0179) MRR_{all}. By 10,000 interactions, the performance increase is 1.55, with *Hybrid* at 0.2896 ($\pm$ 0.0396) MRR_{all} and *Dataset-level* at 0.1861 ($\pm$ 0.018) MRR_{all}. Furthermore, we observe a 2.12 performance increase in MRR_{es} and a 1.2 performance increase in MRR_{dl} with *Hybrid* at 0.2306 ($\pm$ 0.0683) MRR_{es} and *Dataset-level* at 0.1087 ($\pm$ 0.0365) MRR_{es} and *Hybrid* at 0.2858 ($\pm$ 0.0089) MRR_{dl} and *Dataset-level* at 0.2388 ($\pm$ 0.0232) MRR_{dl} respectively. This supports our claim that entity-specific models can increase the performance of the dataset-level model by reducing under-fitting through the elimination of outlying local entities.

Exploration. We evaluate *Hybrid* with varying degrees of exploration $\alpha \in \{0, .2, .5, 1, 2\}$ and find that a small degree of exploration

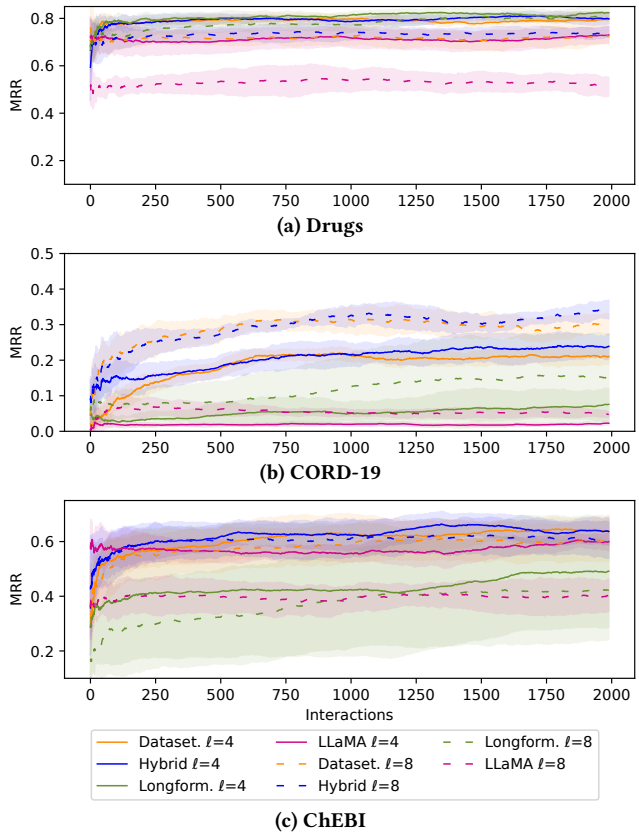


Figure 4: Dataset, Hybrid, and LM-based (Zipf, 4/8 keys)

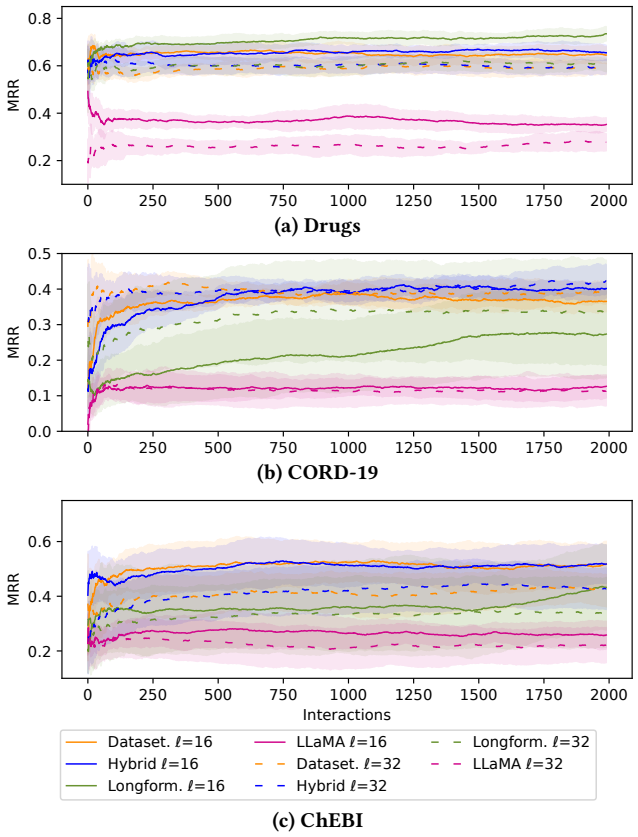


Figure 5: Dataset, Hybrid, and LM-based (Zipf, 16/32 keys)

tends to benefit *Hybrid* more than *Dataset-level*. LinUCB shrinks its upper confidence bounds on features by trying terms with those features sufficiently enough. Thus, models with little feedback (i.e., newly instantiated entity-specific models) will explore to a high degree. We find the most significant benefit on CORD-19 and Products, where $\alpha = 0.2$ results in higher MRR in the long run. Results over ChEBI further suggest that high degrees of exploration (i.e., $\alpha = 1$ and $\alpha = 2$) should be avoided, as the negative effects of exploration over this dataset tend to be exacerbated by entity-specific models. For News, DrugCentral, and Drugs, we find that entity-specifc models do not significantly affect the relationship between degree of exploration and performance.

***Longformer's* rich representations increases its predictive power but also make its performance inconsistent**. Though *Longformer* shows a minor improvement in performance over *Hybrid* in few instances, it also exhibits a more dramatic improvement over Drugs with $\ell = 32$ (Figures 5a). Interestingly, Drugs is another dataset where model performance degrades as ℓ increases. This trend suggests that *Longformer's* richer representation of terms may allow it to better identify and avoid noisy terms. On the other hand, *Longformer* also exhibits some of the worst performance of any of our models on CORD-19 (Figures 4b/5b) and ChEBI (Figures 4c/5c). Furthermore, *Longformer* exhibits a large standard error over both datasets, implying a high variance in performance.

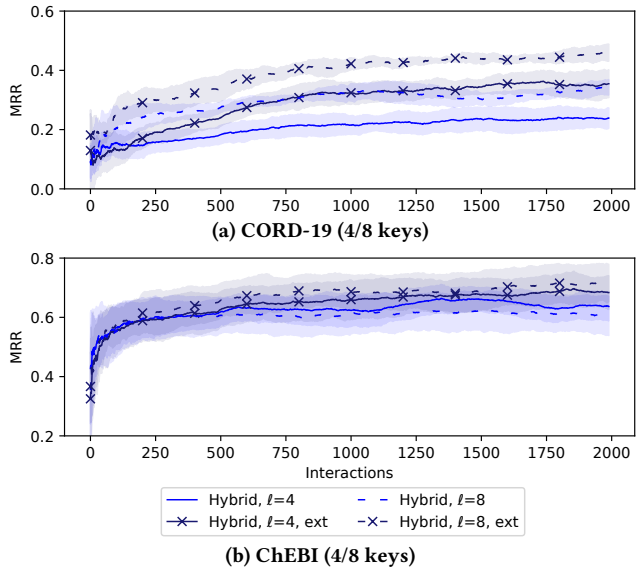
***LLaMA's* extensive model size enhances its representational power; however, it leads to decreases in performance.** Our findings indicate both *Longformer* and *Dataset-level* perform better than *LLaMA* on all datasets, except for ChEBI with $\ell = 4$ (Figure 4c). The intricate nature of *LLaMA's* features and representations necessitates a complex non-linear function to fully exploit its benefits. It is challenging to fit such a function online. While *Longformer's* smaller model size results in lower representational power, it compensates by reducing complexity. This enables *Longformer* to extract valuable information and outperform *LLaMA*.

Both *Hybrid* and *Longformer* show signs of improvement beyond the capacity of *Dataset-Level*. Though *Longformer* shows promise in overcoming capacity issues, its less-than-stable performance indicates that it may not always converge to a sufficiently effectively policy quickly. On the other hand, *Hybrid* shows some promise for overcoming the capacity issue without any noticeable deterioration in performance compared to *Dataset-level*.

7.4 External Terms & Features

We evaluate whether we can improve query effectiveness through the use of external terms and features. We find external terms and features have similar effects on *Dataset-Level* and *Hybrid*. Thus, for the sake of readability, we include only the results of *Hybrid*.

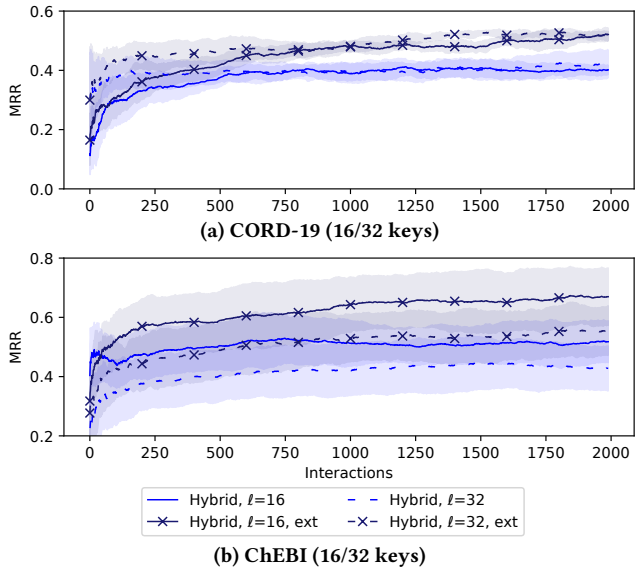
Supervised Term Borrowing improves performance for most datasets. External terms boost performance to varying extents for most datasets (Figure 6a/7a). The one exception is DrugCentral at $\ell = 4, 8$, where external terms and features have no affect on performance. This result is likely due to the models already achieving high performance (Figure 2c/3c). By expanding the set of candidate



(a) CORD-19 (4/8 keys)

(b) ChEBI (4/8 keys)

Figure 6: External feature comparison (Zipf, 4/8 keys)



(a) CORD-19 (16/32 keys)

(b) ChEBI (16/32 keys)

Figure 7: External feature comparison (Zipf, 16/32 keys)

terms with more reliable options, *Hybrid* achieves the overall best performance on some datasets (Figure 6a/7a) and uses fewer noisy terms at high ℓ values on other datasets (Figure 6b/7b).

Unsupervised Term Borrowing helps extract relevant external entities. Over Drugs, News, WDC, ChEBI, and CORD-19, we find minor improvements in MRR for the local entities with expanded candidate term sets. For example, on News with $\ell = 8$, unsupervised term borrowing boosts MRR from 0 to 0.149 ($\pm$ 0.006). Thus, unsupervised term borrowing can help extract relevant external entities that could not be extracted using only those terms that appear in the content of the local entities.

8 RELATED WORK

Pay-as-you-go Data Integration. Researchers have proposed pay-as-you-go integration systems that rely on user feedback [19, 35, 51]. Some systems use pay-as-you-go methods to construct a unified schema and query interface over multiple databases [51]. 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.

ML for Data Integration. Researchers have used ML methods for some data integrating tasks (e.g., schema mapping and entity matching) [6, 14, 21, 23, 26, 43]. As opposed to our setting, these systems require access to the entire content of integrated datasets during training. Moreover, they usually use offline ML methods. Some use active learning for schema mapping and entity matching where the system selects and shows training examples (e.g.. matching candidates) to users for labeling [26, 40, 43]. This method is different from ours as it does not provide any results during training. Also, users do not choose entities.

Data Discovery and Augmentation. Given a query table as an input, data discovery methods seek to find related tables within a large pool of tables (crawled from web, data lakes, companies with many disparate tables across multiple data sources) quickly [8, 17, 20, 41, 50, 53]. Our system, however, aims at finding external information relevant to each local entity. They often preprocess candidate corpora, often by building indexes across (external) tables, which requires access to the entire meta-data and content of those tables. Such accesses do not exist in our setting.

Data Acquisition. Researchers have proposed methods to acquire training examples from external sources to train an ML model [32]. As opposed to our setting, they assume that every query over the external source returns accurate results.

Deep Web Crawling & Querying. Web crawlers aim at extracting the entire information stored in external data sources to organize it for future use (e.g., search [16, 34, 36, 47, 52]). Many Web data sources can be accessed only via form interfaces (i.e., *deep Web*). Researchers have proposed techniques that find a minimal set of queries to crawl all accessible tuples of these data sources [16, 36]. 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-form query interfaces so users can query multiple sources via a single interface [16, 52]. They preprocess query forms to translate the queries over the unified interface to the ones over external query interfaces. Our system, however, finds 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 [47]. 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 [33, 37]. 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] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana De Oliveira. 1996. Characterizing reference locality in the WWW. In *Fourth International Conference on Parallel and Distributed Information Systems*. IEEE, 92–103.

[2] 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.

[3] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32, 1 (2002).

[4] Sorin Avram, Thomas B Wilson, Ramona Curpan, Liliana Halip, Ana Borota, Alina Bora, Cristian G Bologa, Jayme Holmes, Jeffrey Knockel, Jeremy J Yang, and Tudor I Oprea. 2022. DrugCentral 2023 extends human clinical data and integrates veterinary drugs. *Nucleic Acids Research* 51, D1 (12 2022), D1276–D1287. https://doi.org/10.1093/nar/gkac1085 arXiv:https://academic.oup.com/nar/article-pdf/51/D1/D1276/48441389/gkac1085.pdf

[5] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. *arXiv:2004.05150* (2020).

[6] Angela Bonifati, Radu Ciucanu, and Sławek Staworko. 2014. Interactive Join Query Inference with JIM. *Proc. VLDB Endow.* 7, 13 (aug 2014), 1541–1544. https://doi.org/10.14778/2733004.2733025

[7] Christopher Buss, Jasmin Mosavi, Mikhail Tokarev, Arash Termehchy, Maier David, and Stefan Lee. 2023. *Effective Entity Augmentation By Querying External Data Sources*. Technical Report. https://web.engr.oregonstate.edu/~termehca/papers/entityarg.pdf

[8] 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

[9] Matt Chaput. 2016. Whoosh: Fast, pure-Python full text indexing, search, and spell checking library. https://pypi.org/project/Whoosh/

[10] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual Bandits with Linear Payoff Functions. In *AISTATS*. 208–214.

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

[12] Dong Deng et al. 2017. The Data Civilizer System. In *CIDR*.

[13] Abdigani Diriye, Ryen White, Georg Buscher, and Susan Dumais. 2012. Leaving so soon? Understanding and predicting web search abandonment rationales. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 1025–1034.

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

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

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

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

[18] 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

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

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

[21] Lise Getoor and Ashwin Machanavajjhala. 2013. Entity Resolution for Big Data. 1527.

[22] Steven Glassman. 1994. A caching relay for the world wide web. *Computer Networks and ISDN systems* 27, 2 (1994), 165–173.

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

[24] 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.

[25] Max Grusky, Mor Naaman, and Yoav Artzi. 2018. Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies. In *NAACL*.

[26] Sairam Gurajada, Lucian Popa, Kun Qian, and Prithviraj Sen. 2019. Learning-Based Methods with Human-in-the-Loop for Entity Resolution. In *CIKM*. 2969–2970.

[27] 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.

[28] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The Curious Case of Neural Text Degeneration. *International Conference on Learning Representations* (2019).

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

[30] Kevin Jamieson, Matthew Malloy, Robert Nowak, and Sébastien Bubeck. 2014. lil' UCB : An Optimal Exploration Algorithm for Multi-Armed Bandits. In *Proceedings of The 27th Conference on Learning Theory*, Vol. 35. 423–439.

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

[32] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data Acquisition for Improving Machine Learning Models. *Proc. VLDB Endow.* 14, 10 (jun 2021), 1832–1844. https://doi.org/10.14778/3467861.3467872

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

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

[35] 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*.

[36] 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.

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

[38] George A Miller. 1998. *WordNet: An electronic lexical database*. MIT press.

[39] 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.

[40] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active Learning for Large-Scale Entity Resolution. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17)*. Association for Computing Machinery, New York, NY, USA, 1379–1388. https://doi.org/10.1145/3132847.3132949

[41] 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 (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1531–1544. https://doi.org/10.1145/3448016.3458456

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

[43] Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. 2018. Active Learning of GAV Schema Mappings. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '18)*. Association for Computing Machinery, New York, NY, USA, 355–368. https://doi.org/10.1145/3196959.3196974

[44] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[45] 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*.

[46] 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).

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

[48] 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*.

[49] 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

[50] 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 (SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 97–108. https://doi.org/10.1145/2213836.2213848

[51] 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).

[52] Nan Zhang and Gautam Das. 2011. Exploration of Deep Web Repositories. *PVLDB* 4, 12 (2011).
[53] 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 (SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 847–864. https://doi.org/10.1145/3299869.3300065