Effective Entity Augmentation By Querying External Data Sources

Christopher Buss
Oregon State University
bussch@oregonstate.edu

Jasmin Mosavi
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 data sources are accessible only via keyword search interfaces, users usually have to manually formulate keyword queries that extract relevant information for each entity. This 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., each data source may use a different name to refer to the same person). In these 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 discover 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 Mosavi, Mikhail Tokarev, Arash Termehchy, David Maier, and Stefan Lee. Effective Entity Augmentation By Querying External Data Sources. PVLDB, 14(1): XXX-XXX, 2022.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

There is a recognized need to collect and connect information from a variety of data sources [12, 15, 18]. 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 [39]. The main goal of this project is to *repurpose* current drugs to treat or mitigate the symptoms of diseases for which there is no 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 and/or resources, external information often must be retrieved through querying [12, 37]. Many data sources are only accessible via query interfaces/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 drug of interest through query APIs.

However, formulating queries that extract specific information can be troublesome. Different data sources often represent the same concept in distinct forms [11, 13] 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 it elicits no results. They try again using a much more general description of *Zoloft* (i.e., being a *serotonin reuptake inhibitor*). However, their underspecified 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 cohesive representation.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

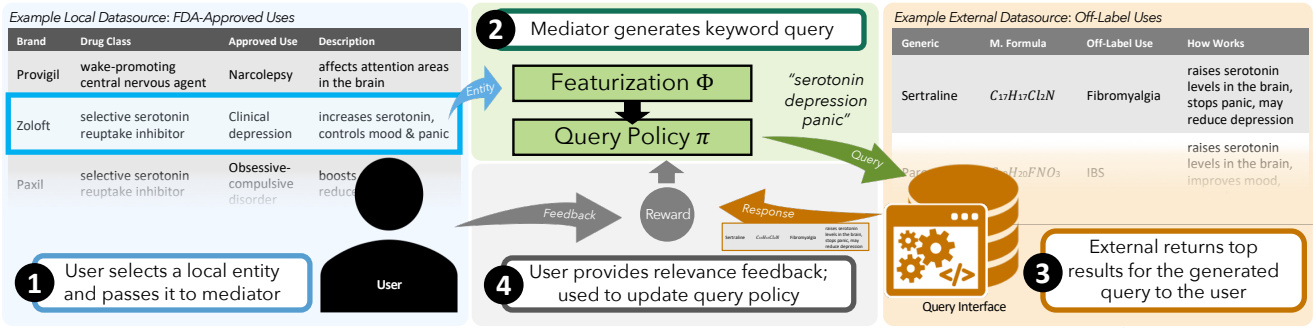


Figure 1: An example of our framework for a single user and single external data source. The user selects (by query, GUI, etc,.) the local entity *Zoloft*. The mediator uses its learned query policy to extract the relevant entity (*Sertraline*) from the external source. The user provides relevance feedback 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 an external source. Most of these rules *cannot* 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 the correct 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 [12]. 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 their querying policy and increase the effectiveness of their 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* [34]. 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 [37] (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 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 effective queries in the *long run* (Section 6).
- We explore whether the broad language understanding capabilities of state-of-the-art deep language models can improve query generation. Namely, we train a small neural network over entity/term features extracted by a large-scale pretrained Longformer model to serve as our query policy (Section 6). We find modest gains in one dataset.
- 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

Before defining the problem of learning querying policies online, we first outline the general framework that our proposed methods operate within. 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. For every local entity $e \in \mathcal{E}$, there exists some relevant entity $X(e)$ (i.e., tuple) in the external dataset. The definition of "relevant entity" depends on the domain. For example, a clinical trial is relevant to the drug that it concerns. $X(e)$ represents the target tuple that the mediator must extract from the external dataset by crafting the correct query for entity e. 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, in order to more accurately evaluate our methods, we assume that $X(e)$ always exists.

Example 1. Tables 1a and 1b show excerpts of local and external datasets, respectively. $\mathcal{E}$ 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. A *querying policy* (or just *policy*) is a mapping $\pi : \mathcal{E} \rightarrow \mathcal{Q}$. An ideal policy would map each local entity $e \in \mathcal{E}$ to a query $q \in \mathcal{Q}$ which extracts $X(e)$ from the external dataset. To the best of our knowledge, these mappings are traditionally written manually. Though $\mathcal{Q}$ is not limited to any specific query language, our remaining discussion will treat $\mathcal{Q}$ as the set of keyword queries accepted by the external data source in order to align with our problem definition.

Extracting Related Information. Formulating a query that extracts $X(e)$ requires overcoming two challenges. First, without knowing the contents of $X(e)$ precisely, the mediator may not know how to express its intent for it. Second, the mediator may not know how the external data source will interpret its query. Keyword queries, are inherently vague [27, 31]. As the ranking method used by the external query interface is often not known, we assume that the mediator does not have any prior knowledge about it. Therefore, an effective policy would account for both representational differences between e and $X(e)$ as well as the way in which the external source quantifies relevance.

Example 2. Given e = *Zoloft*, the mediator must devise a keyword query to extract $X(e)$ = *Sertraline*. One policy might be 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`. Assume this policy maps e (*Zoloft*) to the keyword query *"serotonin depression panic"*.

Query Effectiveness. The mediator's policy is evaluated based on the effectiveness of the queries it produces. There are standard metrics in information retrieval and data management to measure the effectiveness of queries relative to the ranked results they elicit [31]. For example, *Precision@k* is the fraction of relevant answers in the top-k returned results. Another frequently used metric is *Reciprocal Rank* $\frac{1}{r}$ 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 input query in the underlying dataset.

Example 3. The mediator submits its query *"serotonin depression panic"* to the query interface over Table 1b which returns the ranked results {*Paroxetine*, *Sertraline*}. Since $X(e)$ = *Sertraline*, the reciprocal rank of these results would be $\frac{1}{2}$ or 0.5.

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 [12]. 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 [29].

Brand	Drug Class	Approved Use	Description
Provigil	wake-promoting central nervous agent	Narcolepsy	affects attention areas in the brain
Zoloft	selective serotonin reuptake inhibitor	Clinical depression	increases serotonin, controls mood & panic
Paxil	selective serotonin reuptake inhibitor	Obsessive-compulsive disorder	boosts serotonin, reduces stress

(a) FDA-Approved Uses

Generic	M. Formula	Off-Label Use	How Works
Modafinil	$C_{15}H_{15}NO_2S$	Depression	impacts parts of nervous system & brain that control wake-fullness & attention
Sertraline	$C_{17}H_{17}Cl_2N$	Fibromyalgia	raises serotonin levels in the brain, stops panic, may reduce depression
Paroxetine	$C_{19}H_{20}FNO_3$	IBS	raises serotonin levels in the brain, improves mood, controls stress

(b) Off-Label Uses

Table 1: Local database of FDA Drugs and external data source of Off-Label Drugs

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 feedback, we focus on the fundamental question of whether effective querying policies can be learned online and reserve user studies for a future paper. 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 user feedback may be explicit (e.g., click-through [35] or eye movement information [19]) or implicit (e.g., skipping results [26]).

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 bares the full responsibility of training it. Thus, as long as some users provide feedback, the mediator will improve, providing progressively better external results for each local entity, for all users of the system.

3.1 Keyword Query Interface and Results

A keyword query q is a finite string comprised of *terms* (i.e., 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. As explained in Section 1, to save resources, keyword query interfaces might limit the length of input keyword queries. These limits are usually stored in the query interface documentation. We assume that all queries submitted to an external data source have a given fixed length. The particularities of keyword query interfaces could be leveraged to make queries more effective. For example, performance could be improved by incorporating special syntax, such as Boolean operators (ANDs, ORs), or by using advanced search options and faceted search. However, the support for these mechanisms is often interface-specific, so we do not consider them.

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 [24, 31]. Due to the large number of potential answers to many keyword queries, keyword query interfaces often return limited results (e.g., only the top-k). Also, 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, Google.com limits queries to 32 terms, and in our evaluation of dblp.org, queries with as few as 16 terms returned server errors.

3.2 Objective and Challenges

The mediator is involved in a series of *interactions* with the external data source. At interaction t, the mediator is given the local entity $e_t \in \mathcal{E}$ which it maps to a keyword query $\pi(e_t) \in \mathcal{Q}$, where $\pi(e_t)$ represents the output query (i.e., q_t) under the current policy π. The mediator submits q_t to the query interface and receives feedback on q_t based on its ability to extract $X(e_t)$. The mediator uses the feedback to find progressively better policies that more effectively extract relevant entities. Our objective is to develop methods that find an optimal policy quickly to maximize the effectiveness of queries submitted to the external interface.

The effectiveness of query q_t for entity e_t is based on the reward received $r(\pi(e_t), e_t)$. The reward depends both on the unknown qualities of the external data source (how it ranks and returns results relative to q_t) and the method one uses for scoring the returned results relative to e_t and $X(e_t)$. There is at least one query q_t^* which maximizes the reward for entity e_t, $q_t^* = \underset{q \in Q}{argmax}\ r(q, e_t)$.

q_t^* is an optimal query since no other query can be more effective than it. As discussed in Section 2, there are many options for measuring the effectiveness of q_t relative to the external results and $X(e_t)$. We focus on policies that maximize the reciprocal rank (RR) of $X(e_t)$. Thus, we have $r(\pi(e_t), e_t)$ represent RR, as indicated by user feedback, of $X(e_t)$ within the results returned for $\pi(e_t)$.

The reward signal may be noisy as user feedback is often imperfect. For example, users may click on the incorrect tuple or may ignore or miss $X(e_t)$ altogether if it appears at too low of a rank. In this work, we assume that the reward signal contains no noise. Thus, the precise reciprocal rank of $X(e_t)$ is always given by the reward function. In the case where $X(e_t)$ is not contained within the top-k results, the reward is assumed to be zero.

Regret Our formal objective is to find policies that minimize the total regret (thereby maximizing the total reward) across all of the mediator's interactions with the external data source.

$$R_n(\pi) = \mathbb{E}\left[\sum_{j=0}^{n} r(q_j^*, e_j) - r(\pi(e_j), e_j)\right] \tag{1}$$

Regret measures the performance of a policy over an arbitrary amount of interactions n. Thus, it is not enough that our methods eventually find a policy that minimizes future regret: a successful method must also minimize its regret as it searches for said policies.

Balancing 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 possible policies. In order to minimize the regret it incurs, the mediator must search the policy space *intelligently*: if the mediator *exploits* the best query found thus far, it may ignore queries that are more effective; if the mediator 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 (i.e., sending the best queries it has found thus far) and exploring the space of policies to find better queries.

Maintaining Users Engagement Any successful mediator not only must minimize regret but it also must keep users engaged while doing so. 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 the effectiveness of a policy remains relatively low for many interactions, 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 can quickly find a reasonably effective policy in the *short run* while continuing to find more effective policies in the *long run*. The exact definition of a *reasonably effective policy* and *short run* may depend on the number of entities in the local dataset, the amount of information in the external source, the differences in representing information on the entities of interest in the local and external sources, and the complexities of the data items. For instance, it may be unrealistic to aim for a method that finds an optimal (or near optimal) policy for 50% of the local entities in a large local dataset after only 50 interactions with a large external data source whose tuples overlap little with their relevant entities in the local source.

3.3 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 effective. 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 amount of ineffective queries, we limit the terms considered to only those that are *relevant* to 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 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 $L(e)$ together. $L(e)$ may not contain all the terms necessary to form q_t^*, 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 extended to include potentially more effective queries as the mediator interacts with the external data source.

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 which has largely 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 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 reward of including a candidate term $k \in L(e_t)$ within a query for e_t based on the previous queries used for the same entity.

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

where $I(k,j,t)$ is the following indicator function,

$$I(k,j,t) = \begin{cases} 1 & \text{if } k \in q_j, k \in X(e_j), e_j = e_t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

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 3.2, focusing solely on exploiting its current knowledge will prevent the mediator from finding better policies, making greedy term selection a poor approach to minimizing regret.

4.1 A 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 fixed unknown reward distributions, which arm has the largest mean reward [34]. In each round, a MAB algorithm picks an arm and observes the reward sampled from the arm's reward distribution. The success of a MAB algorithm is measured as the expected sum of its losses compared to the algorithm that always chooses the arm with the largest mean reward in n trials. In other words, MAB algorithms are built to minimize regret. Using mild assumptions, it is known that the lower bound of regret for a MAB algorithm, without any prior knowledge on the reward distributions of arms, is logarithmic in the number of trials [34]. There are asymptotically optimal MAB algorithms that estimate confidence bounds on the expected reward of each arm and pick the arm with the largest upper limit of the confidence bound [3, 25]. This approach is called *Upper Confidence Bound* (UCB) [3, 25]. The larger the observed reward of an arm is and the fewer times that arm has been tried, the greater the upper limit of the confidence bound will be. Hence, the UCB approach naturally balances exploitation and exploration in learning. It often chooses the arms with larger observed mean rewards but keeps trying the ones that have not been selected sufficiently frequently due to the relatively large upper limit on their confidence bounds.

Though the UCB algorithm would reduce its regret, it is still 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 to their entities of interest in a local dataset with hundreds of thousands of entities. Thus, the entity-level approach is unlikely to produce effective queries in the *short run* and is impractical over large local datasets.

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 and 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 [34]. 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 provides a low asymptotic expected regret and is commonly applied to contextual bandit problems. LinUCB assumes the reward of each arm to be a linear function of some vector representations of the arm and the current context [9]. The expected regret of LinUCB is of order $\sqrt{Td \ln^3 \frac{KT \ln T}{\delta}}$ with probability $1 - \delta$ where T, d, and K are the number of trials, dimension of the vector representation of arms and contexts, and number of arms, respectively. LinUCB provides an asymptotic regret close to the lowest possible regret, $O(\sqrt{Td})$ [9]. We use LinUCB to learn a shared 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 terms $k \in L(e_t)$ 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 Table 1b). The mediator would quickly learn, irrespective of local entity in Table 1a, 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. Hence, the algorithm explores terms that might not have been tried sufficiently many times in the past. LinUCB needs few updates in its estimated parameter vector in each round making it efficient [9].

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 [32]. 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, like frequencies of terms in entities, to ensure that they are comparable across different entities.

We use NLP practices commonly employed in Information Retrieval to prune candidate terms that are known to be imprecise and redundant [31]. We use the Natural Language Toolkit [6] to remove stop-words (e.g., "the") and apply stemming.

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 help generate 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 output query. These terms may improve the ranking of $X(e)$ in subsequent interactions. If there are multiple external entities relevant to e, (i.e., $X(e)$ has multiple members), these terms may help to retrieve unobserved external entities relevant to e.

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 Table 1b. Unsupervised term borrowing may 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 hyper-parameter to a value between these to extremes (Section 7.5).

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 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. In particular, 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. In this section, we propose methods that can approximate term reward quickly while still having greater representational power, and thus better relative optimal policies, then the dataset-level model.

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 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-level models for entities that the shared model cannot find effective queries for (e.g.,

cannot return any relevant answers) after trying the learned queries by the shared model for those entities sufficiently many times.

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. Because candidate solutions in the entity-level model are terms, it may take too long to learn accurate models for each entity. It cannot also use the knowledge learned by the shared model. Hence, 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, with w_{shared}. This way, the entity-level model may initialize its model weights using the ones learned in the first step. 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.

6.2 Longformer Query Learning Model

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 a large-scale pretrained Longformer [5] model. Contrasted with the linear model, the Longformer model 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. However, 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 this through the Longformer 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 notational convenience, we simply write this process as: $h_1, ..., h_n = \text{Longformer}(s_t)$

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 on to each corresponding representation forming $f_i = [A_t(k_i, e_t), h_i]$ where $[\cdot, \cdot]$ denotes concatenation. f_i is then passed through a small fully-connected neural network to predict reward for each term r_i. In our setting, r_i is an estimate for reciprocal rank (RR) and is bounded between 0 and 1.

Selecting Queries and Updating We apply an ϵ-greedy approach to query formulation [34] – selecting either the next highest scoring term or a random term with probability ϵ 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 holding the last 50 observed terms. We train the model by stochastic gradient descent with batches of 8 samples from buffer at each interaction.

Implementation Details We use the pretrained base longformer model from the Huggingface Transformers library (longformer-base-4096) along with its partner byte-pair-encoding tokenizer. This pretraining gives it strong reasoning capabilities about English words and sentences that are frequently used in our examined entities. The fully-connected nueral network is trained using Pytorch's implementation of Adam with default hyper-parameters. Mean squared error is used as the loss function.

7 EMPIRICAL EVALUATION

7.1 Datasets

We evaluate our methods over a variety of domains using the datasets listed in Table 2. 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. DrugBank contains comprehensive molecular information about drugs, their mechanisms, their interactions, and their targets [38]. DrugCentral uses Drug Central as its external source. Drug Central is similar to Drug Bank in terms of domain, but has a greater focus on regulatory information associated with drugs [4]. On the other hand, ChEBI uses Chemical Entities of Biological Interest (ChEBI), which broadly tracks molecular entities used to intervene in the processes of living organisms [23].

WDC is derived from the non-normalized English *WDC Product corpus*, containing products scraped from many different sites [33]. We include products that look to be successfully scraped and distribute them amongst the local and external sources. In order to make the querying task realistic, we remove shared IDs (ISBNs, SKUs, etc.) that would make extracting relevant entities trivial.

CORD-19 contains records about scientific papers and research related to COVID-19 [36]. We split CORD-19 into two separate sources: one containing abstracts (local) and one containing the remaining attributes (external). There are many overlapping terms due to the entire dataset being COVID-19 related. This makes it more difficult to identify which candidate terms are most effective.

Drugs contains drug reviews from *Drugs.com* (local) [20] and descriptions of the same drugs scraped from *Wikipedia* (external). There is a large diversity of terms within the reviews due to their informal language which is sometimes only loosely related to the drug. Wikipedia articles, on the other hand, are heavily editorialized. For added realism, we include an additional 46K Wikipedia pages that are not relevant to any local drug review.

dataset	source	attributes	avg. terms	entities	#relevant	Perfect MRR (@3)
DrugCentral	Local	name, description, indication, pharmacodynamics, ...	178	3,475	3,457	0.9942
	External	formula, name, fda_labels, drug_class, active_ingredient, ...	279	4,927		
Drugs	Local	drugName, condition, review	108	13,725	413	0.9641
	External	page_title, wikipedia_summary	168	46,976		
News	Local	title, article_summary	42	30,000	30,000	0.9603
	External	article_content	547	30,000		
WDC	Local	category, brand, prod_title, description, ...	67	57,109	55,247	0.8392
	External	category, brand, prod_title, description, ...	72	55,247		
ChEBI	Local	name, description, indication, pharmacodynamics, ...	178	3,475	5,753	0.8575
	External	status, name, definition, charge, formula, mass, ...	73	189,467		
CORD-19	Local	abstract	305	250,575	250,575	0.7945
	External	sha, source_x, paper_title, doi, pmcid, ...	48	340,826		

Table 2: Details of datasets used in our evaluation.

News is derived from a dataset covering 38 major mass-media companies [21]. It contains titles and summaries of articles (local) and the articles themselves (external). Authors drafted the summaries using different techniques, resulting in varying amounts of overlapping terms. The original dataset has three categories based on the number of matching keywords. Our 30,000 record subsample includes articles and summaries from all three categories.

Perfect MRR (@3) in Table 2 indicates the best Mean Reciprocal Rank (MRR) achievable for each dataset when using queries of 3 or fewer terms. 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 only over 5% subsets of each local dataset. Though it is unrealistic to assume that anything but computationally expensive offline algorithms can achieve this performance, we still include it as an indicator of dataset difficulty and term overlap.

7.2 Experimental Setup

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 [8]. 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. As the objective of our models is to maximize Mean Reciprocal Rank (MRR), we also use it to measure their performance. Local entities vary in difficulty, making the performance of models partially dependent on the entities encountered and at what time. To help mitigate this noise, 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 each model 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 and discuss illustrative examples of trends. However, more in-depth results over all datasets can be found in our technical report [7]. We do not include runtime of experiments, but note that much of it is dedicated to external query processing: our models take little time.

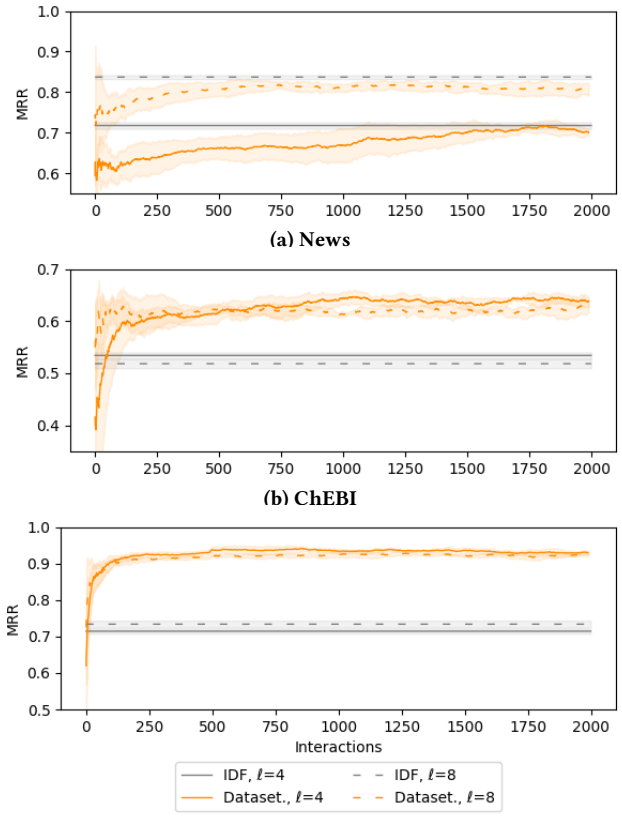
Hyperparameters We treat query length as a hyperparameter and test our models using an ℓ of 4, 8, 16, and 32. These values reflect limits on real interfaces (see Section 3.1) and illustrate how query length affects policy performance. Both *Dataset-Level* (Section 5) and *Hybrid* (Section 6.1) use LinUCB as their exploration strategy while *Longformer* (Section 6.2) uses e-greedy. Both strategies use a hyperparameter to control the extent to which they explore. For LinUCB we use $\alpha = 0.2$ and for e-greedy we use $\epsilon = 0.05$.

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 [24], quantifies how unique a term is to an entity within a dataset. Thus, the IDF of a term may indicate how well the term identifies an entity in the local dataset. As local and external datasets may belong to the same general domain, one might assume that their terms have the same relative IDF (i.e., for terms t_1 and t_2, if $IDF(t_1) > IDF(t_2)$ in the local dataset, then $IDF(t_1) > IDF(t_2)$ in external one). Hence, if a term appears both in the local entity and its relevant external ones and has relatively high IDF in the local dataset, it might also effectively identify relevant entities in external source.

7.3 Dataset-Level Model

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

Dataset-Level* quickly finds policies that outperform *IDF Early on, *IDF* shows 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 is an uncommon quality in datasets. 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).

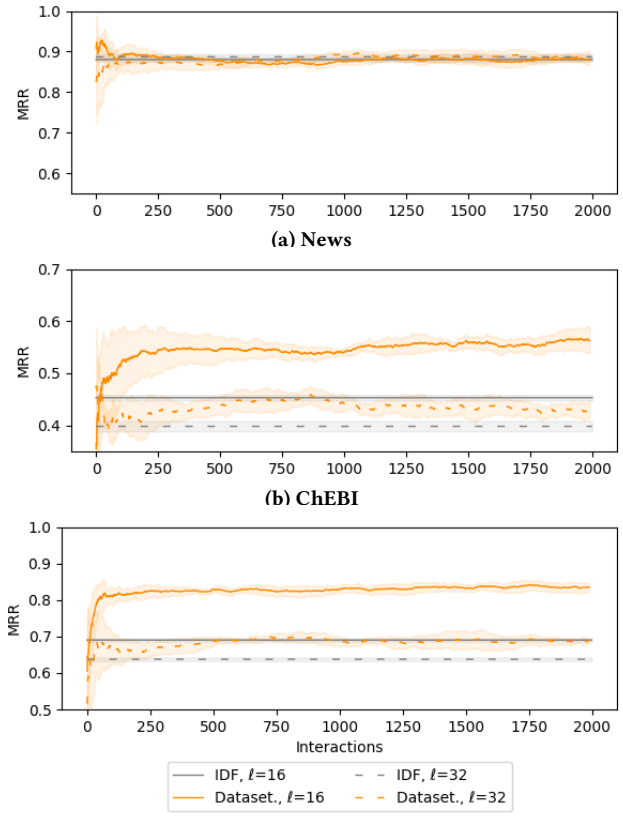


(a) News

(b) ChEBI

(c) DrugCentral

Figure 2: Dataset-Level and IDF (uniform 4/8)



(a) News

(b) ChEBI

(c) DrugCentral

Figure 3: Dataset-Level and IDF (uniform 16/32)

***Dataset-Level* with small ℓ has the most reliable performance** Performance differences between *Dataset-Level* and *IDF* tends 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.

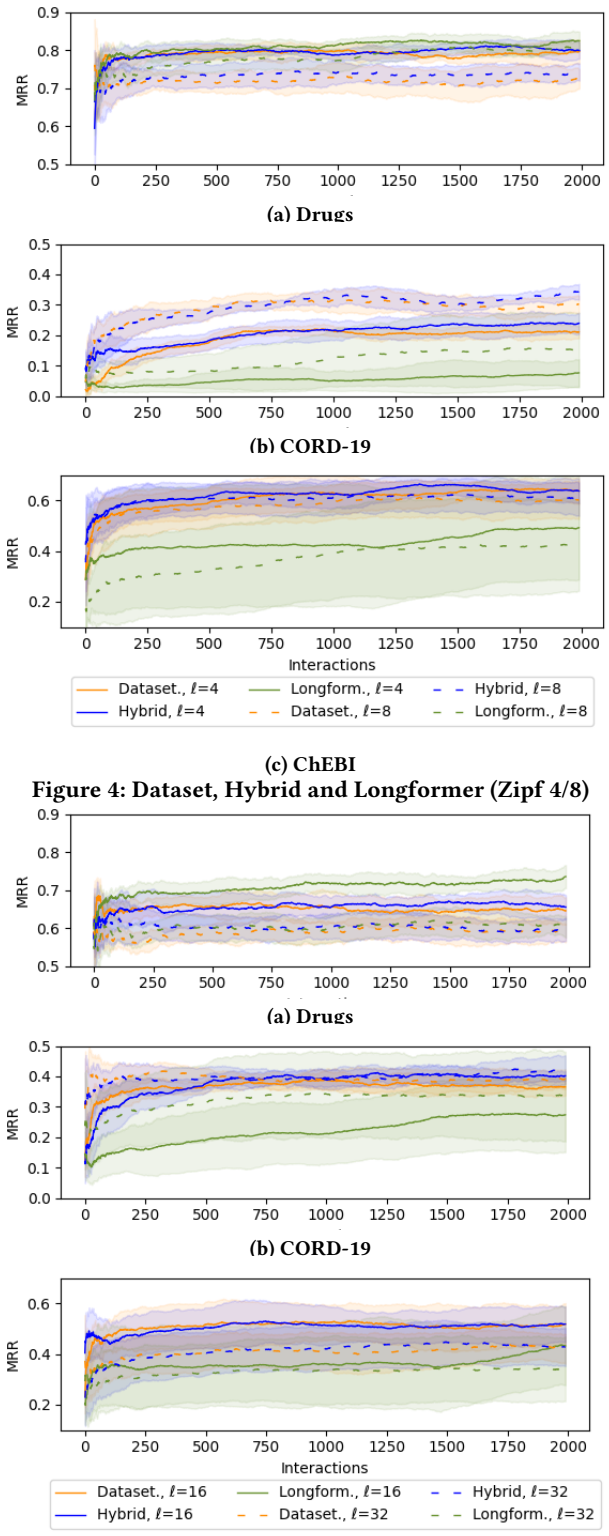
Averaged over all datasets, *Dataset-level* with $\ell = 4$ finds policies that produce an MRR of roughly 0.5 within the first 250 interactions. We consider this to be sufficiently effective performance for the short term and given such little feedback. However, it shows little improvement with additional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

7.4 Overcoming Entity Diversity

Comparing the performance of *Dataset-level's* converged policy with that of *Perfect MRR (@3)* in Table 2, suggests that it is too simple to account for the diversity in local entities. In the following experiments, we compare *Hybrid* (Section 6.1) and *Longformer* (Section 6.2) against *Dataset-Level*. We seek to understand whether our methods can continue to improve their query policies in the *long run*. In order to more accurately measure *long run* performance, we adjust our sampling strategy for local entities.

Studies suggest 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, 10, 17]. This suggests that 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*, *Longformer*, and *Hybrid* under a Zipf sampling of local entities.

***Hybrid* Hyperparameters**. *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 the dataset-level model has historically had poor performance for the local entity (i.e., the previous RR observed for it is less than some threshold β). The hyperparameters n and β are set to 50 and $\frac{1}{15}$ respectively.



(a) Drugs

(b) CORD-19

(c) ChEBI

Figure 4: Dataset, Hybrid and Longformer (Zipf 4/8)

(a) Drugs

(b) CORD-19

(c) ChEBI

Figure 5: Dataset, Hybrid and Longformer (Zipf, 16/32 keys)

***Hybrid* exhibits capacity for improvement over *Dataset-Level*.** *Hybrid* exhibits minor improvement on few datasets, including CORD-19 (Figure 4b), but its performance is otherwise similar to that of *Dataset-Level*. This is likely due to its conservative β: in order to instantiate an entity-specific model, the shared model must be mapping the given local entity to queries that achieve low RR. A small β means the overall policy is unlikely to degrade as entity-specific models are introduced. On the other hand, the few instantiated entity-level models are able to reduce some of the underfitting that the shared model experiences, providing a minor improvement in overall 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 suggests that *Longformer's* richer representation of terms may allow it to better identify the noisy ones and avoid them. On the other hand, *Longformer* also exhibits 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 across its runs.

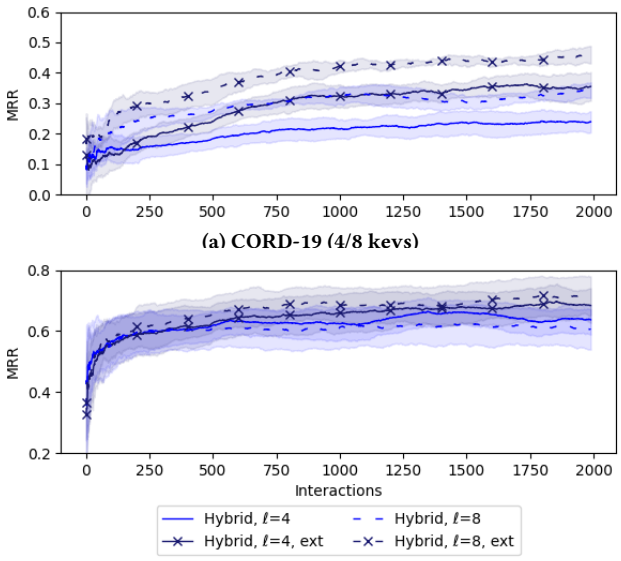
Both *Hybrid* and *Longformer* show signs of improvement beyond the capacity of *Dataset-Level*. Though *Longformer* shows promise in overcoming capacity issues, its less-than-stable performance indicates that it may not always converge to a sufficiently effectively policy quickly (i.e., provide satisfactory performance in the short term). On the other hand, Hybrid has consistent performance and still shows some indication of overcoming the capacity issue without any noticible deterioration in performance.

7.5 External Terms & Features

We seek to answer whether query effectiveness can be improved through the use of external terms and features. We specifically examine our supervised and unsupervised methods for expanding the set of candidate terms for local entities (Section 5.3). We find external terms and features have a similar effect on *Dataset-Level* and *Hybrid*. Thus, for the sake of readability, we only include *Hybrid* in Figures 6 and 7 and omit *Dataset-Level*.

Supervised Term Borrowing improves performance across most datasets. External terms boost performance by varying extents across most datasets over their corresponding versions using the same ℓ (Figure 6a/7a). The one exception to this trend is DrugCentral at $\ell = 4$ and $\ell = 8$ where external terms and features have no affect on performance. This is likely due to the models already achieving high performance even without external terms and features (Figure 2c/3c). By expanding the set of candidate terms with more reliable options (i.e., terms that exist in the relevant entities), *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). This indicates that models significantly benefit from adopting features of the external data source

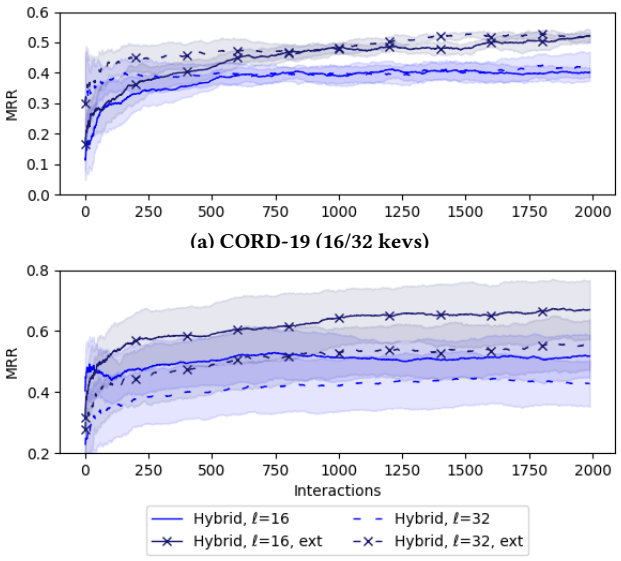
Unsupervised Term Borrowing We adopt term borrowing in an unsupervised setting to determine if we can improve the performance of local entities that have exhausted a percentage of local terms and have yet to extract relevant external entities. **We treat the percentage of exhausted terms as a hyperparameter** that



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

(b) ChEBI (4/8 keys)

Figure 6: Hybrid (external feature comparison) (Zipf 4/8)



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

(b) ChEBI (16/32 keys)

Figure 7: Hybrid (external feature comparison) (Zipf 16/32)

tunes the tradeoff between the certainty that no local terms are relevant to the external entity and how early we can start borrowing terms. If the percentage of exhausted terms is too high (i.e., 100%), it may take several interactions before models start borrowing external terms. On the other hand, if the percentage is too low, models may borrow external terms before sufficiently exploring the local ones (e.g., there might be effective local terms that have not yet been tried). To balance this tradeoff, we set the percentage of exhausted terms to 70% in our experiments.

Unsupervised Term Borrowing can help models extract relevant external entities. Over Drugs, News, WDC, ChEBI, and CORD-19, we have found that unsupervised term borrowing can help to extract relevant external entities that would otherwise not be extracted. We compared local entities whose candidate terms were expanded using unsupervised term borrowing with their counterparts in models that *did not* use unsupervised term borrowing. We found a minor improvement in MRR for the local entities that did have their set of candidate terms expanded. For example, on News with $\ell = 8$, unsupervised term borrowing boosted the target entities MRR from 0 to 0.149 ($\pm$ 0.006).

8 RELATED WORK

Keyword Query Formulation. Researchers have proposed methods to automate keyword query formulation without writing complicated source-specific programs [37]. 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 [27, 31]. 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.

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 [14, 28, 30, 37, 40]). Many Web data sources can be accessed only via form query 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 [14, 30]. 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 [14, 40]. 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.

Entity Resolution. Entity resolution is the problem of finding records that refer to the same real-world entities potentially across different datasets [16, 22]. As opposed to our setting, in entity resolution, all records are available to the algorithms.

9 CONCLUSION

Users often query external sources to find information relevant to entities in local datasets. This is usually done by manually writing programs that given a local entity generate queries that return relevant external entities. This approach takes a long time and a great deal of resources. We proposes novel methods that learn to query external sources online using end-user feedback. Our empirical studies indicate that our methods learn reasonably accurate queries in short run and improve the accuracy of their queries in long run.

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] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc.
[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] Matt Chaput. 2016. Whoosh: Fast, pure-Python full text indexing, search, and spell checking library. https://pypi.org/project/Whoosh/
[9] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual Bandits with Linear Payoff Functions. In *AISTATS*. 208–214.
[10] Carlos Cunha, Azer Bestavros, and Mark Crovella. 1995. *Characteristics of WWW client-based traces*. Technical Report.
[11] Dong Deng et al. 2017. The Data Civilizer System. In *CIDR*.
[12] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of Data Integration* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[13] Xin Luna Dong and Divesh Srivastava. 2013. Big Data Integration. *PVLDB* 6, 11 (2013).
[14] Eduard C. Dragut, Weiyi Meng, and Clement T. Yu. 2012. *Interface Understanding Deep Web Query and Integration*. Morgan & Claypool Publishers.
[15] 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
[16] Lise Getoor and Ashwin Machanavajjhala. 2013. Entity Resolution for Big Data. 1527.
[17] Steven Glassman. 1994. A caching relay for the world wide web. *Computer Networks and ISDN systems* 27, 2 (1994), 165–173.
[18] Behzad Golshan, Alon Y. Halevy, George A. Mihaila, and Wang-Chiew Tan. 2017. Data Integration: After the Teenage Years. In *PODS*.
[19] Laura A. Granka, Thorsten Joachims, and Geri Gay. 2004. Eye-tracking Analysis of User Behavior in WWW Search. In *SIGIR*.
[20] 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.
[21] Max Grusky, Mor Naaman, and Yoav Artzi. 2018. Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies. In *NAACL*.
[22] 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.
[23] 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.
[24] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. 2003. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB*.
[25] 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.
[26] Diane Kelly and Jaime Teevan. 2003. Implicit Feedback for Inferring User Preference: A Bibliography. *SIGIR Forum* 37, 2 (2003).
[27] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.
[28] Weimo Liu, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. 2014. Aggregate Estimation over Dynamic Hidden Web Databases. *PVLDB* 7, 12 (2014), 1107–1118.
[29] 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*.
[30] 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.
[31] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[32] George A Miller. 1998. *WordNet: An electronic lexical database*. MIT press.
[33] 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.
[34] Aleksandrs Slivkins. 2019. Introduction to Multi-Armed Bandits. *Found. Trends Mach. Learn.* 12, 1-2 (2019).
[35] 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*.
[36] 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).
[37] Pei Wang, Ryan Shea, Jiannan Wang, and Eugene Wu. 2019. Progressive Deep Web Crawling Through Keyword Queries For Data Enrichment. In *SIGMOD*. 229–246.
[38] 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*.
[39] 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
[40] Nan Zhang and Gautam Das. 2011. Exploration of Deep Web Repositories. *PVLDB* 4, 12 (2011).