Rows from Many Sources: Enriching row completions from Wikidata with a pre-trained Language Model

Carina Negreanu[*]
Alperen Karaoglu[*]
cnegreanu@microsoft.com
Microsoft Research
Cambridge, UK

Jack Williams
Microsoft Research
Cambridge, UK

Shuang Chen
Harbin Institute of Technology
Harbin, China

Daniel Fabian
Microsoft Research
Cambridge, UK

Andrew Gordon
Microsoft Research
Cambridge, UK

Chin-Yew Lin
Microsoft Research
Beijing, China

ABSTRACT

Row completion is the task of augmenting a given table of text and numbers with additional, relevant rows. The task divides into two steps: subject suggestion, the task of populating the main column; and gap filling, the task of populating the remaining columns. We present state-of-the-art results for subject suggestion and gap filling measured on a standard benchmark (WikiTables).

Our idea is to solve this task by harmoniously combining knowledge base table interpretation and free text generation. We interpret the table using the knowledge base to suggest new rows and generate metadata like headers through property linking. To improve candidate diversity, we synthesize additional rows using free text generation via GPT-3, and crucially, we exploit the metadata we interpret to produce better prompts for text generation. Finally, we verify that the additional synthesized content can be linked to the knowledge base or a trusted web source such as Wikipedia.

CCS CONCEPTS

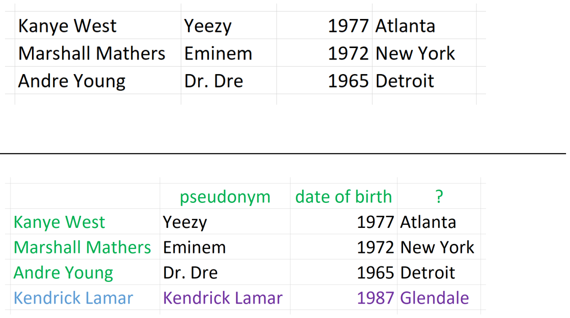
• **Information systems → Language models**.

KEYWORDS

knowledge base linking, natural language applications, language models, semantic knowledge, free text generation, tabular data

ACM Reference Format:
Carina Negreanu, Alperen Karaoglu, Jack Williams, Shuang Chen, Daniel Fabian, Andrew Gordon, and Chin-Yew Lin. 2022. Rows from Many Sources: Enriching row completions from Wikidata with a pre-trained Language Model. In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3487553.3524923

[*]Both authors contributed equally to this research.



Kanye West	Yeezy	1977	Atlanta
Marshall Mathers	Eminem	1972	New York
Andre Young	Dr. Dre	1965	Detroit

	pseudonym	date of birth	?
Kanye West	Yeezy	1977	Atlanta
Marshall Mathers	Eminem	1972	New York
Andre Young	Dr. Dre	1965	Detroit
Kendrick Lamar	Kendrick Lamar	1987	Glendale

Figure 1: Row Completion in a Nutshell

1 INTRODUCTION

Row completion is the task of suggesting new complete rows for a given table. Its purpose is to help users gain confidence in the completeness of their data. Row completion is a table enhancement that is particularly important for spreadsheet intelligence, because it is a step towards unlocking a multitude of other intelligent features that require good quality data, such as data insights and chart recommendations.

Row completion divides into three standard steps that we illustrate using the example in Figure 1. To the best of our knowledge, the overall task of row completion is new, although each of its three steps has been studied independently. The top table is the input table; the bottom table is the linked table with a suggested row.

Step 1: Table Interpretation We link the data in the input table to a knowledge base (KB), such as DBpedia [1] or Wikidata [28]. Table interpretation itself decomposes into three sub-tasks as proposed in [2]: *column type identification*, *entity linking* and *property linking*. The figure shows in green the entities and properties from Wikidata (using the English labels as proxies for the unique numeric identifiers, and "?" denotes an unlinked property).

Step 2: Subject Suggestion We predict additional primary entities described in the table, a task we refer to as *subject suggestion*, since it determines the primary subject of the row and not the complete row.

Our figure shows in blue a single suggested subject: the entity "Kendrick Lamar". In practice, subject suggestion generates an ordered set of entities.

Step 3: Gap Filling We fill in the remainder of the row, a particular case of *gap filling*. Our figure shows in purple the properties of "Kendrick Lamar" filled by our algorithm.

The goal of this paper is to dramatically expand the scope of subject suggestion and gap filling by sourcing entities and properties from relatively uncurated web sources. Our target is Wikidata, the largest, fastest growing, and most up to date open knowledge base.

To achieve this goal, we need to go beyond methods aimed at structured datasets like DBpedia or WikiTables, which inherit structure from their source Wikipedia, highly curated by humans.

Our key new idea is to rely on prompting an existing pre-trained language model during subject suggestion (Step 2) and gap filling (Step 3) to source entity and property candidates by free text generation.

Challenges for Step 2: Subject Suggestion with Wikidata. Subject suggestion first involves generating a set of potential candidates and then ranking them. EntiTables [36], the current state of the art in candidate generation for subject suggestion, sources candidates from DBpedia and WikiTables. Although highly valuable, both sources are extracted from Wikipedia limiting the diversity.

To suggest from Wikidata we need to address two significant challenges. The first is that *Wikidata has a weak type ontology.*

Existing approaches to subject suggestion, such as Entitables [36], rely on a well-defined type ontology to construct search indexes to find candidates. Unfortunately, Wikidata lacks such a well-defined type ontology, and does not require users to provide complete information when entering a new entity, leading to a significant number of missing properties [14]. For example, the type of Cat and Lion in DBpedia is "Mammal", while in Wikidata, a Cat is an instance of "organisms known by a particular common name" (analogous to its type), and a Lion is an instance of "taxon". We take a new approach where we start from multi-relational entity embeddings (such as Pytorch BigGraph (PBG) embeddings [17]) of the seeds and create a candidate set from their nearest neighbours that share a subset of properties linked from the table. In doing so, we can handle Wikidata's steep growth by avoiding expensive computations over the whole entity space. Furthermore, PBG embeddings can be updated efficiently which makes them an ideal candidate for data that evolves rapidly over time.

The second challenge for subject suggestion from Wikidata is that *directly using KB embeddings for subject suggestion can have low recall.* While Wikidata is vast, it remains incomplete which can lead to sparse regions in the embedding space as PBG trains on an input graph by ingesting its list of edges, each identified by its source and target entities and, possibly, a relation type. Thus, for some entities our initial approach might not have sufficient recall and we diversify our candidates by using free text generation. Generative language models like GPT-3 [4] or Codex [5] are trained on enormous datasets extracted from the web, and capture a rich knowledge of co-occurrence information. Augmenting knowledge base subject suggestion using a language model is an enticing way to find new candidates, however prompting GPT-3 using the input table alone delivers poor results. Our approach is to exploit the table metadata we interpret in Step 1 to craft structured prompts. For example, in Figure 1 we use the linked properties to create the prompt "Kanye West has pseudonym Yeezy and has date of birth 1977". Thus, we treat the large language model as a source of knowledge (that needs to be verified) which we query by synthesizing the tabular context. Our approach shows that good quality queries can be constructed for relational tables so we do not have to fine-tune the language model to understand the information in the table.

Turning to candidate ranking for subject suggestion, prior art, such as Table2Vec [35] or more recently Tabbie [12], ranks via entity similarity (between seed entities and candidates) using knowledge base and table similarity statistics. They make use of tabular representations (static embeddings [35] or BERT-based representations [12]) which require significant training and/or fine-tuning (e.g. [12] estimates total emissions at 300kg of CO_2 for this task). We propose building a classifier over features that are generated with metadata already available from the base models (e.g. GPT-3 and PBG) without further training or fine-tuning. As we have a large set of candidates relative to the number of suggestions we want to offer we classify using outlier detectors.

Challenges for Step 3: Gap Filling with Wikidata. In Step 3, we want to fill in the remainder of the row once we have a subject suggestion. Recent state of the art for this task is achieved by TURL [6], a different table representation approach that learns contextualized representations on relational tables by masked entity retrieval. It sources fills from other WikiTables. In our work we fill gaps by using other sources such as Wikidata. Unfortunately, KB sparsity directly affects gap filling as either the information we would like to retrieve doesn't exist (the entity's property value is missing) or there is not enough information to link the column to a property. To overcome this challenge we extend gap filling to include information from multiple sources such as news, Wikipedia articles or any other reliable web sources. Similarly to subject suggestion we use GPT-3 to generate potential candidates. The difference between the two approaches is that instead of trying to verify the candidates by linking back to Wikidata we attempt to ground the generations in a broader range of sources. Assigning provenance to a generation is insufficient to assess whether the generation should be a fill. We need to also use the context from the source to determine if the generation is consistent with the table. For example, if we had a table about athletes we could use GPT-3 to fill in the missing eye-colour information. If one of the generations is "brown" we need to verify that the colour refers to eye colour and not hair colour.

The overall contribution of the paper is to establish a new state-of-the-art for row completion, and its key steps of subject suggestion and gap filling, by using language models in those two steps.

2 RELATED WORK

Various aspects of Row Completion have been previously studied and in this section we will provide an overview. For candidate generation previous work has proposed sourcing candidates from a Knowledge Base such as DBpedia or Freebase ([23], [24], [41], [47], [36]), or use free text ([25], [26], [22]) or use structured text such as web tables ([29], [6], [36]) or web queries ([9], [10], [30]). As several approaches we focus on multi source candidates and our work is closest to [36] and [35].

When extracting candidates from KBs prior art favours DBpedia where comparing types and categories between all entities leads to good candidate generation. This approach works well for well curated KBs, but in the case of Wikidata we found that it is less effective considering its flexible ontology. To this point, recent work [39] proposes category generation for sets of entities.

For ranking candidates, previous work uses various model classes, for example Machine Learning approaches ([31], [6], [15], [42], [32], [12]), probabilistic approaches ([9], [29], [30], [26], [27], [36], [8]), graph theory approaches ([43]), and human-in-the-loop approaches ([21], [7]). Our pipeline uses Machine Learning components as we include free text generation and use the output of an unsupervised outlier detector as a ranking function. Unlike previous work we do not train or fine-tune the language model or the embeddings on tabular data. Due to the lack of a large corpus of varied data (the datasets are primarily synthetic or carefully curated) we are wary of generalization. Nonetheless recent deep learning tabular representation models (such as [12], [6] or [33]) show improvements by better encoding the tabular context.

A related area of study is *set expansion*, an instance of subject suggestion specialized to one column tables without headers, or lists. Relevant work in this space that tackles ambiguity and semantic drift includes [10], [24], [26], [34], [41], [25], [35] and [22].

For the gap filling task [37] aims to fill in gaps in tables with multi-source candidates (from DBpedia and tables) and their algorithms for tabular sources get improved upon by recent work [6]. Our work extends to unstructured text sources (like Wikipedia or news articles) for significant recall and precision gains.

Tangential emerging research areas that are relevant to our work are knowledge acquisition via pre-trained language models and prompt-engineering. Prior work (such as [11], [16] or [48]) uses the knowledge within pre-trained language models for QA, fact checking or truthful generation. Significant efforts have focused on building better prompts and a representative collection can be found in a recent survey [19]. To our knowledge we are the first to focus on prompt engineering for tabular data.

3 TASKS

In this section, we formally define the tabular data interpretation and augmentation problem. We start by describing the tabular data and knowledge base (KB), then formally define the task settings.

Let T range over relational tables of the form $\{t_{1,1}, ..., t_{m,n}\}$, where t ranges over text values. A table is a matrix of text cells of m rows and n columns. Write $T_{[i,*]}$ for row i in table T; write $T_{[*,j]}$ for column j in table T. We do not assume availability of metadata beyond the table contents, such as column headers or types.

The knowledge base (KB) studied in our work follows RDF (Resource Description Framework) standard which consists of a terminological component (TBox) and an assertion component (ABox). Let $\mathcal{E} = \{e_1, ..., e_{|\mathcal{E}|}\}$ be the set of all entities in the KB. The TBox defines the schema structure of the KB including type ontology $\mathcal{T}$ and properties set $\mathcal{P}$. The type ontology $\mathcal{T} = \{\tau_1, ..., \tau_{|\mathcal{T}|}\} \subseteq \mathcal{E}$ consists of type definitions and a type hierarchy constructed with the *subclass of* relation. An entity $e \in \mathcal{E}$ can belong to one or more types of the type ontology. Specifically, we write $type(e) = \tau$ to denote an entity e belongs to a type $\tau \in \mathcal{T}$. The properties set $\mathcal{P} = \{p_1, ..., p_{|\mathcal{P}|}\}$ defines the set of possible properties used to describe key attributes of an entity. The ABox is the instantiation of KB which is composed of a set of RDF triples $\langle s, p, o \rangle$, where s denotes a subject (an entity $e \in \mathcal{E}$), $p \in \mathcal{P}$ is a property (also known as predicate or relation) and o denotes an object (an entity e, or a data value, e.g. number, time, string etc.). We write $p(e)$ for *property lookup* that returns o when $\langle s, p, o \rangle$ exists in the KB, and $\perp$ otherwise. We implicitly assume that $p(e)$ maps to zero or one target. Although tailored for our target KB, Wikidata[1], the same notation applies to other knowledge bases like DBpedia and Freebase, etc.

Definition 1. Given table T and entity set $\mathcal{E}$, *table entity linking* aims to link entity mention m_k in a specific cell $t_{i,j}$ of table T to its referent entity in $\mathcal{E}$, or predict there is no corresponding entity in the KB, denoted $\perp$. In Figure 2, 1b displays linked entities.

Definition 2. Given table T and property set $\mathcal{P}$, *property linking* associates a pair of columns, indexed by s and o, with a property $p \in \mathcal{P}$ such that property p relates $T_{[*,s]}$ and $T_{[*,o]}$ component-wise: $p(t_{i,s}) = t_{i,o}$ for all i. In Figure 2, 1c displays linked properties.

Definition 3. Let $E = (e_{1,1}, ..., e_{n,1})$ be the list of entities corresponding to the left most table column, which we refer to as the *main column. Subject suggestion* generates a ranked list of entities E_{new} to be added to E. In Figure 2, 2 displays suggested subjects.

Definition 4. Given table T we extend the main column with the entity set E_{new}. For each $e_{i,1} \in E_{new}$ *gap filling* returns $o_{i,j}$ where $o_{i,j}$ is either sourced from the KB via a triple $\langle e_{i,1}, p_j, o_{i,j} \rangle$, or via a verified web source. In Figure 2, 3 displays gap suggestions.

Definition 5. *Row completion* is the task of returning a complete new row, that is, *Row completion = Subject suggestion + Gap Filling*.

4 ALGORITHMS

Given a set of candidate entities for the main column we start by trying to find the most likely mapping between relational columns and Wikidata properties. We take a similar approach to [40] with the exception of two problems that we address differently: property sparseness and numerical matching. Full details of this extension can be found in Appendix A and in this section we will focus on the algorithms for Subject Suggestion and Gap Filling.

At the end of this phase our algorithm has produced the meta-data shown in Step 1 in Figure 2. We find the in-table properties *pseudonym* and *date of birth* and fail to link the third column. We manage to link all entities in the main column and identify possible types for them (such as *human*).

4.1 Subject suggestion

Algorithm 1 Subject suggestion for table T with seeds $\mathcal{L}$ (sketch)

```
$suggest(T, \mathcal{L})$ =
    $C \leftarrow \{e \mid \forall i < m, e \in \mathcal{E}.\ distance(\mathcal{L}_i, e) < threshold\}$ 2a
    $P \leftarrow \{link(GPT3(toPrompt(T_{[i,*]}))) \mid \forall i < m\}$ 2b
    return $rank(features(T, \mathcal{L}, C \cup P))$
```

Algorithm 1 presents the high-level algorithm for subject suggestion which is represented by Step 2 in Figure 2. Subject suggestion takes as input an (m, n)-table T and the linked main column $\mathcal{L}$, and

[1]https://www.wikidata.org/

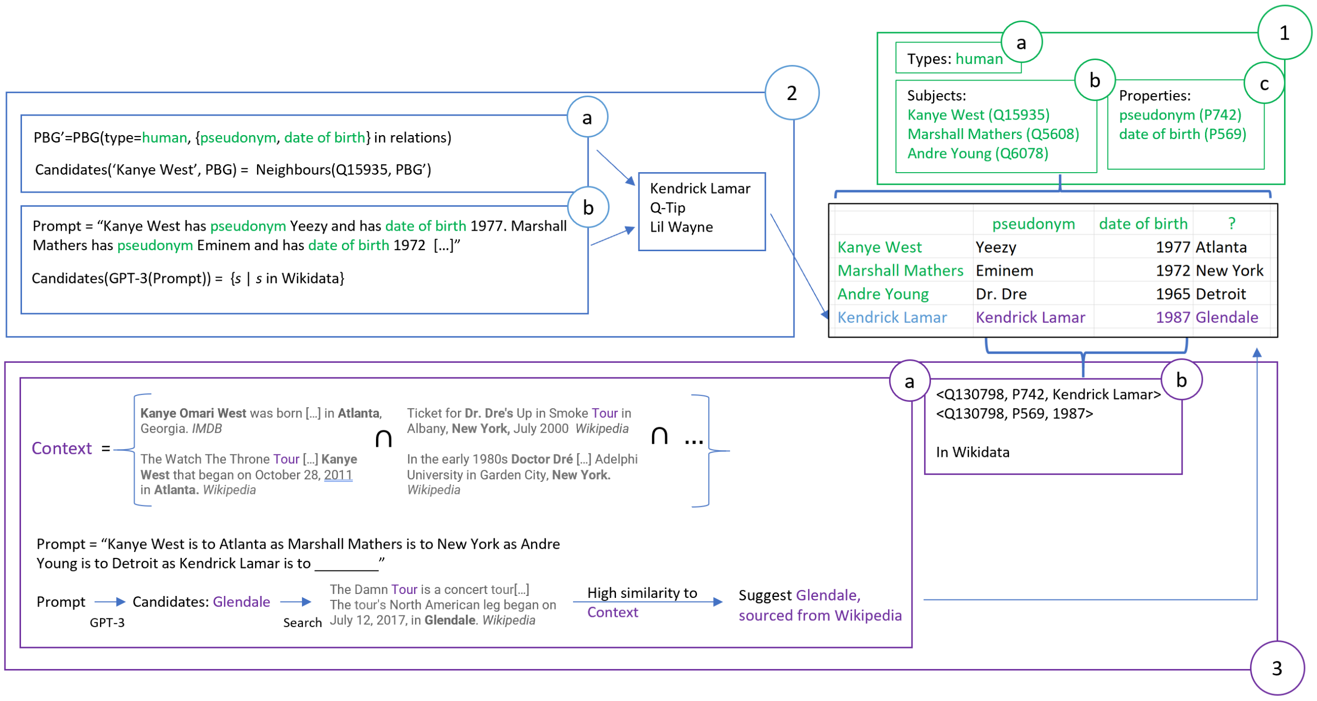

1
a
Types: human
b
Subjects:
Kanye West (Q15935)
Marshall Mathers (Q5608)
Andre Young (Q6078)
c
Properties:
pseudonym (P742)
date of birth (P569)
pseudonym
date of birth
?
Kanye West Yeezy 1977 Atlanta
Marshall Mathers Eminem 1972 New York
Andre Young Dr. Dre 1965 Detroit
Kendrick Lamar Kendrick Lamar 1987 Glendale
2
a
PBG'=PBG(type=human, {pseudonym, date of birth} in relations)
Candidates('Kanye West', PBG) = Neighbours(Q15935, PBG')
b
Prompt = "Kanye West has pseudonym Yeezy and has date of birth 1977. Marshall Mathers has pseudonym Eminem and has date of birth 1972 […]"
Candidates(GPT-3(Prompt)) = {s | s in Wikidata}
Kendrick Lamar
Q-Tip
Lil Wayne
3
a
Context =
Kanye Omari West was born […] in Atlanta, Georgia. IMDB
The Watch The Throne Tour […] Kanye West that began on October 28, 2011 in Atlanta. Wikipedia
∩
Ticket for Dr. Dre's Up in Smoke Tour in Albany, New York, July 2000 Wikipedia
In the early 1980s Doctor Dré […] Adelphi University in Garden City, New York. Wikipedia
∩ …
Prompt = "Kanye West is to Atlanta as Marshall Mathers is to New York as Andre Young is to Detroit as Kendrick Lamar is to ________"
Prompt
GPT-3
Candidates: Glendale
Search
The Damn Tour is a concert tour[…] The tour's North American leg began on July 12, 2017, in Glendale. Wikipedia
High similarity to Context
Suggest Glendale, sourced from Wikipedia
b
<Q130798, P742, Kendrick Lamar>
<Q130798, P569, 1987>
In Wikidata

Figure 2: Row completion pipeline. In Step 1 we link the table to Wikidata, find the in-table properties (Step 1c), and the possible types (Step 1a) and corresponding Q-numbers for the subject entities (Step 1b). To retrieve other subject entities (*Kendrick Lamar*) we rank candidates generated via PBG' (Step 2a) and GPT-3 (Step 2b). To fill in the remaining relevant information we either suggest GPT-3 verified candidates (Step 3a) or we retrieve directly from Wikidata (Step 3b).

returns a ranked list of subjects to extend the main column. The algorithm for subject suggestion comprises two sub-tasks: candidate generation and candidate ranking.

Candidate generation. Candidate generation is an important task as ranking all candidates in Wikidata has a high computational cost and at the current rate of growth it cannot scale even with the most efficient ranking algorithms. Our novel method to create a diverse candidate set is shown in Figure 2, Step 2.

First, we source KB candidates using the PBG embedding space (Step 2a). Similar to *Word2Vec*, PBG generates multi-relation embeddings by maximizing the (transformed) similarity score for existing RDF triples while minimizing it for non-existent ones. The similarity is measured by cosine distance (TransE) or dot product distance (RESCAL, DistMult, ComplEx) over transformed vectors for the source and object. The transformations considered are linear transformations (RESCAL), translations (TransE) and complex multiplication (DistMult, ComplEx). A viable candidate is a neighbor of at least one of the seeds present, shares at least one type in common with the seeds, and has some of the in-table properties.

A key assumption in our work is that using nearest neighbors in PBG is meaningful for generation via similarity. We have found that requiring full property overlap is too restrictive for our choice of KB considering the property sparseness in Wikidata. In some cases (for example when the type is human) we need to further limit the number of candidate generations so we constrain by demanding all in-table properties to be present. If we have a small number of seeds (our current setup), we search for the nearest neighbors of each seed. We can extend this approach for very large tables when we might want to keep extra seeds by checking if we can find clusters in the seed space. Using PBG embeddings has a few important strong points. We did not have to train a new set of embeddings on relevant tabular data to build a space that is representative for our setup (as proposed in [35]) because the embedding space is already constructed based on the relations between entities. We also do not have to worry about restricting our search to a fixed number of hops. By transitivity if the RDFs $\langle s, p, o\rangle$ and $\langle s, p, o'\rangle$ exist then o and o' will likely be neighbors (and similarly for $\langle s, p, o\rangle$ and $\langle s', p, o\rangle$) as described in [17]. As a final point, PBG updates efficiently with extra data, which is important as we want to have up-to-date suggestions.

The second approach for candidate generation is to enhance the candidate pool with predictions obtained from GPT-3 free text generation (as shown in Step 2b of Figure 2). GPT-3 is an autoregressive language model with 175 billion parameters that has shown promising performance in the few-shot setting. Input table T is converted to a prompt and used to sample GPT-3, from which candidates are generated. To create the prompt for GPT-3 we extractively summarize the seed rows given the properties identified. We create a template by concatenating the transformed $\langle s, p, o\rangle \longrightarrow$ *"s has p o"* that we identify on each row and we provide the text generator the template. The text generator then fills in new templates and we extract the entities from the templates. If we can link the entities to Wikidata we add them to the candidate set.

Candidate ranking. After we create a set of candidates from PBG ($\mathcal{C}$) and GPT-3 ($\mathcal{P}$) we want to create a ranked list. In practice, to maintain high levels of recall, the number of candidates we generate is considerably larger than the desired number of completions.

In order to address the large class imbalance we rank by using the output of an outlier detector that returns the probability that a given candidate is part of the suggestion set. The assumption that we make is that *given a set of well curated features the relevant suggestions will appear as outliers in the candidate set.* Our models make use of the following features:

(1) the distance to the closest seed in PBG (the choice of metric depends on the way the graph is generated e.g cosine similarity for TransE, dot product for ComplexE),
(2) the percentage of properties not in the table in common with seeds,
(3) minimum normalized Levenshtein distance between candidate entity label and seed entity labels,
(4) the minimum string embedding (in FastText space [3]) cosine distance between the candidate entity label and seed entity labels,
(5) the percentage of types shared between seeds and candidate,
(6) the percentage of seeds that have the candidate as a neighbor,
(7) the candidate generator (GPT-3, PBG or both),
(8) the GPT-3 generation score (if available).

We start from the toolbox proposed in [46] for outlier detection algorithms. For semi-supervised and supervised systems we first use a kNN algorithm on the training data to assign table-clusters based on our feature space (and then map the test tables to a given cluster). As a supervised system we use XGBOD [44], an ensemble system that combines unsupervised outlier mining algorithms to extract representations to improve the feature space on top of which runs a supervised classifier. For each cluster we train a VAE [13] on non-outliers (incorrect predictions) and find outliers using the reconstruction error (the hypothesis being that the higher the reconstruction error, the most likely a candidate is to be a viable suggestions). We also consider unsupervised methods that are proximity-based (e.g. LOF) or neural networks (e.g. MO-GAAL [20]).

4.2 Gap filling

Once we link the table to the KB and find good suggestions to append to the main column we can in principle easily fill in the remaining information in the table from the KB. Unfortunately, there are two cases where we cannot retrieve the information: when we cannot identify the property the column represents (for example, when the property does not exist in the KB, as shown in column four in Figure 2) and when we can identify the property but the entity we are trying to fill the value for does not have the property in the KB (for example, an athlete's eye colour is not recorded).

In order to address these challenging situations we query GPT-3 for potential fills. To mitigate hallucinations we want to link the generation to at least one web source (as GPT-3 is trained on web data). Unfortunately, just linking is insufficient as we could find that although the fill is accurate it is not relevant for the table, for example GPT-3 could retrieve the correct hair colour of the athlete but we are interested in their eye colour. Our approach compares the context from the linked web source with a synthesized context that links the seed entities and their relevant values to decide if the fill is relevant. Another benefit of linking is that it provides a way to mitigate the fact that GPT-3 might suggest out of date information (as the language model is unlikely to update frequently). For example, GPT-3 could retrieve the population of the US, but the data might not be from the most recent census. When linking to a web source we could warn the user the information is out of date from the metadata of the web source.

Algorithm 2 Ranked Gap Filling for cell (i, j) in linked table $\mathcal{L}$.

```
rankValues$(s, j, \mathcal{L}, p?)$ =
  $context \leftarrow$ contextOfSeeds$(\{$bing$(\mathcal{L}_{[i,1]}, p, \mathcal{L}_{[i,j]}) \mid i \leq$ rows$(\mathcal{L})\})$
  $scoredValues \leftarrow \emptyset$
  for $i$ in $sample$
    $o \leftarrow$ GPT3(toPrompt$(\mathcal{L}_{[*,1]} \cdot \mathcal{L}_{[*,j]})$)
    $snippets \leftarrow$ bing$(s, p, o)$
    $best \leftarrow$ argmax$_{x \in snippets}$ score$(x, context)$
    if $best_{score} > threshold$ then
      $scoredValues \leftarrow scoredValues \cup \{(o, best_{text})\}$
  return $scoredValues$

gapFill$(i, j, \mathcal{L})$ =
  $s \leftarrow \mathcal{L}_{[i,1]}, p \leftarrow$ propertyLinks$(\mathcal{L}, j)$
  if$\langle s, p, o \rangle \in$ KB then return $\{(o, \langle s, p, o \rangle)\}$ 3b
  return rankValues$(s, j, \mathcal{L}, p)$ 3a
```

Algorithm 2 presents the algorithm for ranked gap filling which is represented by Step 3 in Figure 2.

Formally, gap filling takes as input a target cell (i, j) and a *linked* (m, n)-table $\mathcal{L}$, and returns a set of ranked values with provenance. Provenance is either an RDF-triple or a sourced text snippet.

Gap filling proceeds by obtaining the subject s for the row containing the gap, and the linked property p for the column containing the gap. Property p can be undefined when property linking fails, for example because the underlying KB lacks the relation. When the KB contains a triple with the corresponding subject and property we immediately return this triple as the value to populate the gap, as shown in Step 3b in Figure 2. If a triple does not exist (because the property or value is missing) we query GPT-3 to retrieve a set of candidates, corresponding to function rankValues.

The function rankValues takes as input a subject s, a column j, a *linked* (m, n)-table $\mathcal{L}$, and an optional property p denoted by p?. First we define *context*, a loose context given the seed information. For each seed row we query Bing for representative sentences using the seed subject, the value in column j of the row, and optional property p as keywords by using the *bing* function. We then use sentence transformers to encode the information into a context. To find the most likely context shared among the seeds we compute the (cosine) similarity across encodings. All the sentences that are most similar (with similarity above a learned threshold) become the context. This process is defined by *contextOfSeeds*.

The next step is to iteratively sample GPT-3 for gap suggestions using a prompt created by concatenating prompts for every seed row. When the property p is missing we use an analogy-based prompt to retrieve a set of candidates, as shown in Step 3a in Figure 2. For a subject s and value o we construct prompt "s is to o as". When the property p is linked we construct a prompt of the form "s has p o". The function *GPT3* returns the GPT-3 suggestions that are obtain by using the prompt generated by *toPrompt*. We prune incorrect GPT-3 candidates by verifying against information from trustworthy sources, producing *snippets*, a set of sourced text snippets. Sourced snippets are obtained by linking candidate $\langle s, p, o \rangle$ triple to Wikipedia or news articles using a web-search engine (like the Bing API). Each source snippet is scored using (cosine) similarity to *context*, and if the best snippet has sufficient score, the value and source is added to *scoredValues*.

Unlike prior approaches, our solution does not require having a two step process to first determine if a cell should be filled. We learn a rough threshold on the validation set such that candidates with scores less than the given threshold are not proposed which can lead to blank cells.

5 EXPERIMENTAL EVALUATION

In this section we present our experimental results. By convention, in every table of results we indicate prior work using a citation, and all other entries are results of this work.

5.1 Details for reproducibility

Datasets. In this work we run extensive experiments on the standard benchmark proposed in [36], a dataset of 1000 real tables curated by humans. Most importantly this dataset is suitable for evaluation for candidates sourced either from DBpedia or Wikidata as the tables are extracted from Wikipedia (so either source should be able to retrieve all relevant suggestions). To our knowledge there are currently no benchmarks that are more extensive.

As the WikiTables dataset was curated in 2014 as part of TabEL [2], the set of ground truth suggestions can be incomplete (for example, the table about US presidents does not include Donald Trump), or some of the values are out of date (for example, the data is from a previous census). Unfortunately, extending the ground truth tables can be fairly noisy so we chose a more conservative approach: we created a set of rough table extensions by scraping Wikipedia and we checked if any of the extra entities are among the highly ranked candidates in our model. After manual validation we removed them when reporting results. We have chosen this approach in order to fairly compare with prior art which uses the data from the 2016 version of DBpedia. This problem affects at least 67 out of 1000 tables.

A second issue with the curated test set is that the main assumption in the task definition does not necessarily hold for this dataset. A significant number of tables in the test set are not strictly relational (as defined in [38]). We estimate that at least 83 tables in our dataset are not strictly relational, but a join of several relational tables. To align with prior art and the standard definition of subject suggestion, we report results relative to the first column even though more properties relate to another column.

For rapid development we created a pipeline to download Wikidata dumps and load them into a Spark distributed cluster in DataBricks that accepts between 2-11 workers (Standard DS4 v2 with 28GB Memory, 8 Cores, 1.5 DBU). The full pipeline can efficiently run in Databricks and the runtime upper-bound for the full test set is 14 minutes for entity-property linking, 52 minutes for candidate generation, 11 minutes for feature generation, 6 minutes for ranking and 58 minutes for gap filling. We believe the runtime can be significantly improved as we have not focused on optimizing the pipeline. For reproducibility we use the pre-trained PBG embeddings for Wikidata[2] (which includes 78 million entities), the 300 dimensional, 1 million FastText Embeddings[3] trained with subword information on Wikipedia 2017, UMBC web base corpus and statmt.org news dataset and the Hugging Face sentence transformer with bert-base-nli-mean-tokens[4].

Particulars to reproduce Candidate Generation Experiments. In order to generate GPT-3 candidates we call the API[5] 100 times (with 1 sentence completions) and use temperature 0.7 (we trade stability for diversity). We generate 100 raw candidates per table out of which, on average, 73 can be linked to a Wikidata entity.

For PBG generation imposing type constraints reduces the search space by a factor of 50 at the cost of limiting maximum average recall to 93.8%. Introducing the in-table property constraint reduces the space further by a factor of 2 and improves recall slightly.

Particulars to reproduce Ranking Experiments. In our work we have experimented with the available detectors in [46]. In general we found that ensembles give a performance boost of 5%-7% and in particular XGBOD [44] and LSCP [45] perform best. We set the contamination level between 1%-6% and as we use the output just for ranking (and not classifying), we are looking for methods that are tolerant to changes in the contamination hyperparameter. For proximity based methods we are also interested in the number of neighbours and for all the methods we have tried, the results did not change drastically by changing this hyperparameter.

For unsupervised methods, MO-GAAL[20] outperformed other neural network approaches, but we found that it was not particularly stable for our set of features (level of contamination and number of subgenerators had a fair impact).

Particulars to reproduce Gap Filling Experiments. For row completion we first try to retrieve the relevant RDF triples. If that is not possible we query GPT-3 under the setup previously described for candidate generation. Using the output of the gap filling generation we proceed to try to verify by linking to web sources. We call the Bing API[6] restricted to news and Wikipedia articles and we retain the top 10 matches. We construct the description of the page by concatenating the *name* and *snippet* fields. In our experiments we found that using the ranking of the search engine was not fruitful. To extend the search one can use the *deep links* option to return related webpages that Bing found on the webpage's website or the *relatedSearch* option to return a list of most related queries made

[2]https://github.com/facebookresearch/PyTorch-BigGraph
[3]https://fasttext.cc/docs/en/english-vectors.html
[4]https://huggingface.co/sentence-transformers/bert-base-nli-mean-tokens
[5]https://beta.openai.com/
[6]https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

Average Recall per nr candidates generated			
Method	Anchor	# Candidates	
		1000	10000
EntiTables KB [36]	DBpedia	64.1	78.8
Entitables [36]	+ tables	78.4	94.0
PBG TransE/ComplexE	Wikidata	72.2/72.4	88.1/88.2
+ GPT-3 (Our method)	+ free text	87.4/87.5	94.5/94.7

Table 1: Candidate generation for subject suggestion. Average recall for generations from top 3 seeds. For 1000 generations we observe a significant gain over prior art. The difference between PBG versions is not significant.

by other users. For our dataset this extension did not improve results. We threshold our results by showing a fill only if the source similarity score is above 0.05.

5.2 Experimental results

Candidate generation results. Unlike previous approaches we generate candidates from Wikidata and unfortunately the prior generation algorithms are not suitable for this KB. Categories play an important role in previous approaches [36], but for the vast majority of Wikidata entities categories are not provided and thus we cannot fairly compare approaches by a direct implementation of prior art. Recent work extends categories to Wikidata [39] and in our future work we will investigate leveraging this approach.

We provide a first comparison in Table 1 by looking at the performance of our approach versus the current state of the art in candidate generation, EntiTables[36]. For KB generation we consistently improve over prior art despite searching in a significantly larger space. When introducing other sources (tables versus free text) we perform significantly better for small numbers of candidates, but our advantage reduces as we increase the number of generations. We hypothesize that the discrepancy in KB size is relevant when generating many candidates. In practice, ranking significantly more than 1000 candidates is not practical so our goal is to have quality generations within 1000 candidates.

Previous approaches only report results by generating suggestions when they consider the top-k rows as seeds. We argue that unless we treat rows as an ordered collection the generation should be order agnostic. We report average recall for 3 or 4 seeds in Table 2 and generate 5000 candidates. We have included only 3 and 4 seeds as previous models have peak performance in this case and GPT-3's prompt requires a fair amount of information. The error margin is relatively small for PBG and prior art, but significant for GPT-3. In our study we observed that GPT-3 performs very well for tables when there is an apparent order (for example, consecutive years). Our best hypothesis is that when the seeds have a certain pattern GPT-3 confidently reproduces the pattern, which is consistent with previous studies [4]. By improving the prompt we have gained a significant performance boost while shrinking the variation. When we tried the same approach with GPT-2[7] we did not achieve satisfactory results. We have ran secondary experiments on an annotated 100 instances, where we noticed that GPT-3 is resilient to badly worded prompts (i.e. that are not fluent), but GPT-2 predictions improved (by 8%). Based on our experimentation around various prompt modifications we conclude that this approach does not generalize well for less performant language models. We also explored using Codex[8], a descendant of GPT-3 for code generation. We encoded the table as a Pandas DataFrame and included it in Codex's prompt together with the ask to add more examples. The relevance of the generations decreased, but the structure quality improved (i.e. more suggestions had the correct type). We hypothesize that as Codex is trained on a significant amount of data from Github, and most examples contain toy datasets, it loses some of its capability to *directly* retrieve factual information.

[7]https://huggingface.co/gpt2

Stability study		
	3 seeds	4 seeds
Entitable KB [36]	73.1 +/- 0.1	74.9 +/- 0.08
Entitables [36]	91.6 +/- 0.08	92.5 +/- 0.07
PBG	76.2 +/- 0.09	77.6 +/- 0.08
GPT-3 base	43.8 +/- 2.1	46.5 +/- 3.2
GPT-3 best	59.7 +/- 1.5	61.4 +/- 1.7
GPT-2	31.1 +/- 3.6	32.9 +/- 3.3
Codex	49.2 +/- 2.0	48.4 +/- 1.9
PBG + GPT-3 best (Our method)	**94.1 +/- 0.5**	**94.3 +/- 0.4**

Table 2: Subject suggestion. Average recall performance with error-bars by choosing every 3-seed/4-seed combination in top 5 rows. We limit to an average of 5000/6000 candidates per table to recover the EntiTables generations from [35]. GPT-3 base uses as prompt the table as is, GPT-3 best uses our improved prompt as shown in Figure 1, GPT-2 uses the same prompt as GPT-3 best and Codex uses as prompt the table as a Pandas DataFrame alongside a simple ask.

Ranking results. For the task of table completion we test the capabilities of the ranking function by using the candidates previously generated. When comparing the generation results between the two methods we find that in the case where we restrict candidate numbers to 5000 we have only 1122 candidates in common (on average). As we want to test the ranking function we run our outlier detection algorithms over both candidate sets in Table 3. As baselines we include Table2Vec [35] which augments the approach from EntiTables [36] with static table embeddings, as well as a new enhancement of EntiTables with a new, deep learning tabular representation TABBIE[9] [12].

In our work we report results only for 3 or 4 seeds as we found that the results for 2 and 5 seeds are very similar (in line with the findings of prior art). The first two rows in the table show that by extending the baseline with TABBIE's table representation we can improve upon Table2Vec. On its own, TABBIE ranks 300k entities and has a performance of 43.4/44.1 for 3/4 row seeds. When coupling it to our approach or EntiTables we restrict the classifier to the set of candidates we are considering.

In our work we have experimented with several outlier detectors including VAEs[13], MO-GAAL[20] and LSCP+LOF[45], but found they perform 5-10% worse than XGBOD[44]. One common result across all detectors is that they significantly over perform for our

[8]https://openai.com/blog/openai-codex/
[9]using the implementation found at https://github.com/SFIG611/tabbie

Mean average precision for subject suggestion				
METHOD	OUR CANDIDATES		ENTITABLES	
	3	4	3	4
Table2Vec [35]	55.8	56.7	64.0	65.2
EntiTables [36] + TABBIE [12]	63.2	63.7	**66.0**	**66.2**
XGBOD (Our method)	72.4	72.6	59.9	60.2
XGBOD + TABBIE [12]	**72.8**	**72.9**	64.9	64.9

Table 3: Ranking for subject suggestion. Mean average precision (MAP) for subject suggestion by taking the top 3/4 rows as seeds. We report performance on the whole ground truth set and inherit a handicap from candidate generation (by comparing performance from our candidates and EntiTable's).

candidate set compared to the one from EntiTables. We hypothesize this is caused by the way we constructed our feature space to take advantage of PBG's/GPT-3 structure which does not translate as well for DBpedia entities mapped in the Wikidata space.

As shown in Table 3, we achieve best results when we include TABBIE's classifier scores for our candidates as an extra feature for XGBOD. We observe that the performance increase is not significant as the added feature is highly correlated with GPT-3's generation score (a Pearson Coefficient of 0.89) and medium correlated with the distance to closest seed in PBG (a Pearson Coefficient of 0.72).

Thus, we conclude that for subject suggestion using a language model to generate candidates has a significantly bigger impact than using it to encode a table when combined with KB mining. On tables that are not relational this conclusion is unlikely to hold and we leave it to future research to investigate other types of tables. The choice of language model can be impactful and our experiments suggest that the capacity of the model, and the data the model is trained on are important.

Gap Filling. For gap filling we compare against various completion methods as shown in Table 4. TURL [6] improves gap filling with tabular data over prior art CellAutocomplete [37] and we use it as a baseline. Unlike CellAutocomplete we first try to link to a KB and if that is not possible we use auxiliary methods. By including GPT-3 suggestions our method significantly improves over the direct KB retrieval baseline. Unsurprisingly, as the tables are sourced from Wikipedia we find that 82% of GPT-3 suggestions are grounded in Wikipedia but we also find that 37% can come from other sources as well (for example, BBC). As the dataset is fairly simple we find very few cases (5%) when GPT-3 generates more than 2 viable completions and it mainly generates a single viable completion (78%). An interesting case where the GPT-3 completion fails to return reasonable candidates is for numerical cells. The recall for numerical cells is 38.4% in contrast with strings for which the recall is 82.9%. By looking at the suggestions the language model returns numerical values that are of the right type (for example, it returns years when appropriate) but are hallucinations. In this dataset most numerical cells were linked by the KB, but we think that including table candidates would be beneficial in other datasets.

To link a GPT-3 completion to an article we must firt generate a loose context from the seed rows. To do so we have experimented with bag of words overlap and standard sentence embeddings as well as sentence transformers. The first two methods had similar performance and led to 78% precision and we obtained a slight improvement by introducing transformers (79.3%).

Precision/Recall Gap Filling				
METHOD	ANCHOR	P @ 1	R @ 1	P @ 3
KB	Wikidata (Wd)	**97.8**	37.2	**97.8**
TURL [6]	tables	49.2	41.1	66.3
TURL + KB [6]	Wd + tables	63.8	55.3	76.1
GPT-3 + KB (Our)	Wd + free text	79.3	**70.2**	82.2

Table 4: Gap Filling. We extend prior methods by including candidates that are verified in news or Wikipedia articles and improvements in both recall and precision are significant (around 15%). The test set includes all cells, except the subject column, for non-seed rows in the 1000 WikiTables dataset.

6 CONCLUSIONS AND FUTURE WORK

The main goal of our work was to provide a system that successfully suggests new, complete rows from large, diverse sources like Wikidata or trustworthy websources.

We showed that despite metadata scarcity (i.e no headers or captions) we can start from a few examples and generate the relevant metadata to a quality that is good enough to exploit by pre-trained systems such as GPT-3 or PBG.

From our studies we conclude that candidate generation is an extremely important step and language models can play a key role in improving this space. For relational tables the gains from having a rich tabular representation (that comes at a significant cost) can be surpassed by generating a high-quality set of candidates.

Limitations. Although our work has shown great potential there are still limitations that should be addressed. The current scope, relational single subject tables, is quite restrictive and further research needs to be conducted to first extend to multi subject tables and then to mixed tables (not strictly relational). As discussed in the results section we have noticed performance drops significantly for numerical properties (including dates) and we need to improve the system for such cases. Finally the system is tailored for Wikidata, but it should be extended to other KBs as well.

Future versions. An interesting direction for future work is to create a system that holistically improves subject suggestion and row completion simultaneously, for example by creating a feedback loop. We plan to investigate this option in our system's second version. Before releasing such a system in the wild, we would like to explore how we can extend our approach to be user-centric and define new measures for success.

Unfortunately, without better datasets it is hard to validate that new approaches actually improve upon prior art. Improving current datasets is a priority for our work and we will investigate annotating a large corpora.

A ACKNOWLEDGEMENT

We would like to thank Christian Canton, Chris Oslund, Nick Wilson, Lena Yeoh and Yordan Zaykov for the insightful conversations, design jams and overall support throughout our project.

B PROPERTY LINKING

Given a set of candidate entities for the main column we try to find the most likely mapping between relational columns and Wikidata properties. We take a similar approach to [40] with the exception of two problems that we address differently: property sparseness and numerical matching.

In this section we first describe our generic property linking function (Algorithm 3), and then we describe our type-specific scoring functions for numeric and string values.

Algorithm 3 Property linking for column index j

```
$\mathsf{link}(T, \mathcal{L}, j) =$
  $\mathcal{P} \leftarrow \{p \in \mathcal{P} \mid \exists e \in \mathcal{L}.\, p(e) \neq \bot\}$
  $\text{scores} \leftarrow \{p \mapsto \sum_{i \leq m} \text{score}_{ij}(p) \mid p \in \mathcal{P}\}$
  $\text{scores}_{\max} \leftarrow \{p \mapsto n \in \text{scores} \mid n = \max(\text{image}(\text{scores}))\}$
  $\text{scores}^{\approx} \leftarrow \{p \mapsto n + n' \mid \forall p \mapsto n \in \text{scores}_{\max}.$
    $n' = \sum_{i \leq m} \text{if } \text{score}_{ij}(p) = 0 \text{ then } \text{score}^*{}_{ij}(p) \text{ else } 0\}$
  $\text{scores}^{\approx}_{\max} \leftarrow \{p \mapsto n \in \text{scores}^{\approx} \mid n = \max(\text{image}(\text{scores}^{\approx}))\}$
   if $\text{scores}_{\max} = \{p \mapsto n\} \wedge n \geq$ threshold then return $p$
   if $\text{scores}^{\approx}_{\max} = \{p \mapsto n\} \wedge n \geq$ threshold then return $p$
   else fail "Cannot Resolve Confidently"
```

Generic Property Linking. Algorithm 3 presents our generic property linking function $\mathsf{link}(T, \mathcal{L}, j)$. The function $\mathsf{link}(T, \mathcal{L}, j)$ accepts as input an (m, n)-table T, linked main column $\mathcal{L}$, and target column index j; the function returns the linked property for column j or fails. We write $x \leftarrow e$ to denote the assignment of the value of expression e to identifier x.

The property linking function is implicitly parametrised by two scoring functions that assign a score to a property p for a particular cell (i, j): we write $\text{score}_{ij}(p)$ for the *exact* property score, and we write $\text{score}^*{}_{ij}(p)$ for the *approximate* property score. The implementation of score and score* depends on the type of the column we are linking: numeric or string. The concrete implementations of these functions is described later in the section.

Function $\mathsf{link}(T, \mathcal{L}, j)$ proceeds as follows. First, define $\mathcal{P}$ to be the set of properties such that there exists an entity in main column $\mathcal{L}$ with that property. Define scores to be the set of property-score ($p \mapsto n$) mappings where p is in $\mathcal{P}$, and n is the sum of exact property scores for all values in the target column. Define $\text{scores}_{\max}$ to be the set of property-score ($p \mapsto n$) mappings where n is the largest score in scores. Multiple properties may share the highest score, hence we record a set. Define $\text{image}(M) = \{n \mid (p \mapsto n) \in M\}$. Values as written in the table will frequently differ from the exact value as defined in the KB, hence we also compute an approximate set of scores. Define $\text{scores}^{\approx}$ to be the set of property-score ($p \mapsto n + n'$) mappings where p is in $\mathcal{P}$, n is the exact score, and n' is the approximate adjustment. The approximate adjustment is defined as the sum of approximate property scores for all values in the target column that did not match exactly. Define $\text{scores}^{\approx}_{\max}$ analogously to $\text{scores}_{\max}$. Finally, we determine the property link using $\text{scores}_{\max}$ and $\text{scores}^{\approx}_{\max}$. When $\text{scores}_{\max}$ has a unique high score (is a singleton set $\{p \mapsto n\}$), and provides sufficient coverage (determined by *threshold*), we link using p. When that fails, we apply the same process to $\text{scores}^{\approx}_{\max}$, and if approximate matching fails, we return no match.

Numeric Properties. Matching numerical-valued columns to properties is an interesting problem as standard approaches for string matching (like fuzzy matching) are not as effective. For example, consider a column about the heights of athletes. As these values can vary between measurements and various sources could report different values, linking directly to a KB would not be possible.

To address such challenges if direct matching methods are not fruitful (i.e. we cannot directly match within unit conversions) we consider the values in the columns holistically and we compare their statistics (for this paper we only consider ranges) with the statistics of the numerical properties for a given class of entities. For instance if we identify that the main column has type "athlete" we check if the values in the column match any of the numerical property values in the KB that are representative for athletes. If that is not the case we compare the statistics of the values in the column against reasonable statistics for numerical properties of athletes (e.g. if the range of our data is within 102-210 it is consistent with heights for athletes).

Formally, for numeric properties we introduce the concept of *characteristic ranges.* Write $C_{p;\tau}$ for the characteristic range of property p for type τ, defined as $C_{p;\tau} = \text{range}(\text{characteristic}(\mathcal{E}_{p;\tau}))$, where $\mathcal{E}_{p;\tau} = \{p(e) | \forall e \in \mathcal{E}.\, type(e) = \tau,\, p(e) \neq \bot\}$ and p is a numeric property.

The *characteristic* function removes the outliers from the set by using the Isolation Forest Algorithm [18]. The *range* function returns upper and lower bounds for a set of numbers.

We now define a property scoring function for a numeric cell value $t_{i,j}$ in table T, given linked main column $\mathcal{L}$ which is a column vector of entities. First, write $\mathbb{1}$ for the *indicator* function that maps true to 1 and false to 0. Define $\text{score}_{ij}(p) = \mathbb{1}(conv(p(\mathcal{L}_i) = t_{i,j}))$. We say that a cell value (t_{ij}) is consistent with the property value of the linked entity in the same row $(\mathcal{L}_i)$ if their values are equal within unit conversion (*conv*). Define $\text{score}^*{}_{ij}(p) = \mathbb{1}(\exists \tau.\, conv(t_{i,j}) \in C_{p;\tau} \wedge type(\mathcal{L}_i) = \tau)$. We say that a cell value (t_{ij}) is approximately consistent with the property value of the linked entity in the same row $(\mathcal{L}_i)$ if their values are within the characteristic range, modulo unit conversion (*conv*).

String Properties. Wikidata has significant property sparseness since users are not obliged to include all properties when adding entities. This makes assigning properties to columns more challenging as direct or fuzzy string-matching can be insufficient. For example, if a column is about the eye colour of athletes we are likely to struggle to identify the property as a significant amount of athlete entities do not have eye colour as a property.

We want to address the case where we find that at least one value in the column matches to a property value, but other entities do not have that property and thus we cannot confidently assign the column. Our method estimates how likely it is that the property is missing (but should be present) for the other entities by checking if similar entities have it. In our previous example if most athletes in our table do not have the property eye colour, but similar athletes to them do, we can increase our confidence that eye colour is the correct match for our column.

We define a property scoring function for a string cell value $t_{i,j}$ in table T, given linked main column $\mathcal{L}$. Define $\text{score}_{ij}(p) = \mathbb{1}(p(\mathcal{L}_i) \approx t_{i,j})$. We say that a cell value (t_{ij}) is consistent with the property value of the linked entity in the same row $(\mathcal{L}_i)$ if their values are equal within fuzzy matching, written $\approx$ (similarly to [40]). Define $\text{score}^*{}_{ij}(p) = \arg\max_{p \in pred} \max_{e \in \mathcal{E}_{1i}} \text{score}(p, e)$. We estimate how likely it is that the property is missing. We do this by looking at the properties of the n nearest neighbors in PBG that share the type of e.

Similar to *Word2Vec*, PBG generates multi-relation embeddings by maximizing the (transformed) similarity score for existing RDF triples while minimizing it for non-existent ones. The similarity is measured by cosine distance (TransE) or dot product distance (RESCAL, DistMult, ComplEx) over transformed vectors for the source and object. The transformations considered are linear transformations (RESCAL), translations (TransE) and complex multiplication (DistMult, ComplEx).

Thus, for TransE embeddings we compute a score to estimate what is the likelihood that entity e is missing property p as $\text{score}(p, e) = \overline{\sum_n} sim(e_i, e)(\mathbb{1}(p(e_i) \neq \perp) - \mathbb{1}(p(e_i) = \perp))$, where $sim(e, e_i) = 1 - \frac{L_2(e,e_i)}{\max(\{L_2(e,e_j) | j<n)\})}$ and where the operator $\overline{f}$ acting on function f is defined as $\overline{f}(x) = \min(\max(0, f(x)), 1)$.

Entity-Property linking results. In our work we extend a previous system *LinkingPark* that was recently externally evaluated in the SemTab Competition. The system ranked 2nd overall performing particularly well for the main entity type annotation task (CTA)[10]. By enhancing the prior pipeline with the new property linking module we improve performance slightly over the SemTab dataset (unfortunately for this task simple methods like direct linking achieve around 95% precision). For Rounds 1-3 we have an up to 1% improvement over the base model and for Round 4 a 1.2% improvement. The main performance difference can be seen for the WikiTables dataset. By including the new module we improve coverage from 38% to 44%. This dataset does not have ground truth available for KB linking but we can infer from the gap filling performance via RDFs that precision is 98.7%.

[10]https://www.cs.ox.ac.uk/isg/challenges/sem-tab/2020/results.html

REFERENCES

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[2] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *The Semantic Web - ISWC 2015*.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* (2017).

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea. Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *ArXiv* abs/2107.03374 (2021).

[6] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. arXiv:2006.14806

[7] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. 2015. Top-k Entity Augmentation Using Consistent Set Covering. Association for Computing Machinery.

[8] Zoubin Ghahramani and Katherine A Heller. 2006. Bayesian sets. In *Advances in neural information processing systems.*

[9] Rahul Gupta and Sunita Sarawagi. 2009. Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment* (2009).

[10] Yeye He and Dong Xin. 2011. Seisa: set expansion by iterative similarity aggregation. In *Proceedings of the 20th international conference on World wide web.*

[11] Benjamin Heinzerling and Kentaro Inui. 2021. Language Models as Knowledge Bases: On Entity Representations, Storage Capacity, and Paraphrased Queries. In *EACL.*

[12] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Association for Computational Linguistics.

[13] Diederik P. Kingma and M. Welling. 2014. Auto-Encoding Variational Bayes. *CoRR* abs/1312.6114 (2014).

[14] Markus Krötzsch. 2018. Ontological modelling in Wikidata. Talk at *Workshop on Ontology Design and Patterns 2018,* held at *ISWC 2018.* Available at https://iccl.inf.tu-dresden.de/w/images/e/ed/Ontology_modelling_Wikidata_Markus_Kroetzsch_WOP2018.pdf.

[15] Guy Kushilevitz, Shaul Markovitch, and Yoav Goldberg. 2020. A Two-Stage Masked LM Method for Term Set Expansion. *arXiv preprint arXiv:2005.01063* (2020).

[16] Nayeon Lee, Belinda Z. Li, Sinong Wang, Wen tau Yih, Hao Ma, and Madian Khabsa. 2020. Language Models as Fact Checkers? *ArXiv* abs/2006.04102 (2020).

[17] A. Lerer, Ledell Yu Wu, Jiajun Shen, Timothée Lacroix, Luca Wehrstedt, A. Bose, and Alexander Peysakhovich. 2019. PyTorch-BigGraph: A Large-scale Graph Embedding System. *ArXiv* (2019).

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. (2012).

[19] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ArXiv* (2021).

[20] Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, M. Wang, and X. He. 2020. Generative Adversarial Active Learning for Unsupervised Outlier Detection. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[21] F. Qi, X. Wu, and N. Wang. 2017. Building Top-k Consistent Results for Web Table Augmentation. In *2017 14th Web Information Systems and Applications Conference (WISA).*

[22] Zhenyu Qi, Kang Liu, and Jun Zhao. 2012. Choosing Better Seeds for Entity Set Expansion by Leveraging Wikipedia Semantic Knowledge. In *Pattern Recognition - Chinese Conference, CCPR 2012, Beijing, China, September 24-26, 2012. Proceedings.*

[23] Dominique Ritze, Oliver Lehmberg, and C. Bizer. 2015. Matching HTML Tables to DBpedia. In *WIMS '15.*

[24] Xin Rong, Zhe Chen, Qiaozhu Mei, and Eytan Adar. 2016. Egoset: Exploiting word ego-networks and user-generated ontology for multifaceted set expansion. In *Proceedings of the Ninth ACM international conference on Web search and data mining.*

[25] Luis Sarmento, Valentin Jijkuon, Maarten De Rijke, and Eugenio Oliveira. 2007. " More like these" growing entity classes from seeds. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management.*

[26] Jiaming Shen, Zeqiu Wu, Dongming Lei, Jingbo Shang, Xiang Ren, and Jiawei Han. 2017. Setexpan: Corpus-based set expansion via context feature selection and rank ensemble. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.*

[27] Isabel Valera and Zoubin Ghahramani. 2014. General Table Completion using a Bayesian Nonparametric Model. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada.*

[28] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[29] Chi Wang, Kaushik Chakrabarti, Yeye He, Kris Ganjam, Zhimin Chen, and Philip A Bernstein. 2015. Concept expansion using web tables. In *Proceedings of the 24th International Conference on World Wide Web.*

[30] Zhijun Xiao, Cuiping Li, and Hong Chen. 2020. PatternRank+NN: A Ranking Framework Bringing User Behaviors into Entity Set Expansion from Web Search

Queries. *ACM Trans. Web* (2020).

[31] Lingyong Yan, Xianpei Han, Ben He, and Le Sun. 2020. End-to-End Bootstrapping Neural Network for Entity Set Expansion.. In *AAAI*.

[32] Lingyong Yan, Xianpei Han, Le Sun, and Ben He. 2019. Learning to Bootstrap for Entity Set Expansion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

[33] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

[34] Puxuan Yu, Zhiqi Huang, Razieh Rahimi, and James Allan. 2019. Corpus-based Set Expansion with Lexical Features and Distributed Representations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[35] Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[36] Shuo Zhang and Krisztian Balog. 2017. Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[37] Shuo Zhang and Krisztian Balog. 2019. Auto-Completion for Data Cells in Relational Tables.

[38] Shuo Zhang and Krisztian Balog. 2020. Web Table Extraction, Retrieval, and Augmentation: A Survey. *ACM Trans. Intell. Syst. Technol.* (2020).

[39] Shuo Zhang, Krisztian Balog, and Jamie Callan. 2020. Generating Categories for Sets of Entities.

[40] Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. 2020. Novel Entity Discovery from Web Tables. In *Proceedings of The Web Conference 2020*.

[41] Xiangling Zhang, Yueguo Chen, Jun Chen, Xiaoyong Du, Ke Wang, and Ji-Rong Wen. 2017. Entity Set Expansion via Knowledge Graphs. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*.

[42] Yunyi Zhang, Jiaming Shen, Jingbo Shang, and Jiawei Han. 2020. Empower Entity Set Expansion via Language Model Probing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*.

[43] Zhenzhong Zhang, Le Sun, and Xianpei Han. 2016. A Joint Model for Entity Set Expansion and Attribute Extraction from Web Search Queries. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*.

[44] Yue Zhao and M. K. Hryniewicki. 2018. XGBOD: Improving Supervised Outlier Detection with Unsupervised Representation Learning. *2018 International Joint Conference on Neural Networks (IJCNN)* (2018).

[45] Yue Zhao, M. K. Hryniewicki, Zain Nasrullah, and Z. Li. 2019. LSCP: Locally Selective Combination in Parallel Outlier Ensembles. In *SDM*.

[46] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. PyOD: A Python Toolbox for Scalable Outlier Detection. *Journal of Machine Learning Research* (2019).

[47] Yuyan Zheng, Chuan Shi, Xiaohuan Cao, Xiaoli Li, and Bin Wu. 2017. Entity Set Expansion with Meta Path in Knowledge Graph. In *Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part I*.

[48] Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual Probing Is [MASK]: Learning vs. Learning to Recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.