Towards Parameter-Efficient Automation of Data Wrangling Tasks with Prefix-Tuning

David Vos
University of Amsterdam
d.j.a.vos@uva.nl

Till Döhmen
University of Amsterdam
t.r.dohmen@uva.nl

Sebastian Schelter
University of Amsterdam
s.schelter@uva.nl

Abstract

Data wrangling tasks for data integration and cleaning arise in virtually every data-driven application scenario nowadays. Recent research indicated the astounding potential of Large Language Models (LLMs) for such tasks. However, the automation of data wrangling with LLMs poses additional challenges, as hand-tuning task- and data-specific prompts for LLMs requires high expertise and manual effort. On the other hand, finetuning a whole LLM is more amenable to automation, but incurs high storage costs, as a copy of the LLM has to be maintained. In this work, we explore the potential of a lightweight alternative to finetuning an LLM, which automatically learns a continuous prompt. This approach called prefix-tuning does not require updating the original LLM parameters, and can therefore re-use a single LLM instance across tasks. At the same time, it is amenable to automation, as continuous prompts can be automatically learned with standard techniques. We evaluate prefix-tuning on common data wrangling tasks for tabular data such as entity matching, error detection, and data imputation, with promising results. We find that in five out of ten cases, prefix-tuning is within 2.3% of the performance of finetuning, even though it leverages only 0.39% of the parameter updates required for finetuning the full model. These results highlight the potential of prefix-tuning as a parameter-efficient alternative to finetuning for data integration and data cleaning with LLMs.

1 Introduction

Data wrangling tasks such as finding duplicates during data integration, detecting errors in tables or imputing missing attribute values during data cleaning arise in virtually every data-driven application scenario [19, 18]. Traditionally, these tasks are framed as classification problems and tackled with machine learning techniques [8, 5, 2, 6, 17].

Data wrangling with large language models. Recent research [14] indicated the astounding potential of Large Language Models (LLMs) for data wrangling tasks. LLMs are neural networks pre-trained on large quantities of raw text. Narayan et al. showed that LLMs can achieve state-of-the-art performance on data wrangling tasks when manually tuned with a simple transfer learning technique called prompting [14]. In prompting, the model parameters are frozen and the model performs an inference task based on a textual input that describes the inputs, formulates the task, and potentially contains examples. The model prediction is taken from the generated textual output of the model in response to the prompt. A concrete example for data wrangling is to generate a prompt that asks an LLM to perform entity matching, e.g.: `Product A is Title: Macbook Pro Price: $1,999, Product B is Title: Macbook Air Price: $899. Are product A and product B the same?`. The output is evaluated by checking whether the LLM generates `Yes` or `No` as a response [14].

Data management challenges in data wrangling with LLMs. A major challenge for the outlined data integration and cleaning tasks is to automate them for complex real-world use cases [15]. Examples are enterprise data warehouses with large numbers of different tables or cloud database vendors, which host and maintain hundreds of customer databases. The approach of prompting LLMs is attractive for such use cases, as it can re-use a single pre-trained model for several tasks and tables. A major downside is however that *prompting requires high expertise and manual effort to engineer high-quality task- and data-specific prompts*. This is not actionable for enterprise databases with thousands of different tables, or for cloud vendor use cases, where employees are legally prohibited from viewing the customers' data. In summary, prompting incurs low storage costs (as the LLM can be re-used), but high manual costs for automation. A common alternative for transfer learning with LLMs is to finetune the LLM to a given task. While this can be automated and typically achieves high performance [14], it has the major disadvantage that it requires the maintenance of copies of the (adjusted) model parameters. Therefore, *finetuning results in low manual costs but high storage costs*.

Contributions and limitations. Based on these insights, we explore a lightweight alternative to transfer learning with LLMs which *automatically learns continuous prompts*. This approach called *prefix-tuning* [9] combines the advantages of both previously discussed approaches: (i) Analogous to prompting, prefix-tuning does not require updating the original parameters of the pre-trained LLM, and can re-use a single LLM instance across multiple tasks and tables; (ii) Prefixes are continuous model inputs (in contrast to discrete prompts) and can be automatically learned with standard techniques. Therefore, prefix-tuning has the potential to enable data wrangling with both *low manual costs* (as continuous prompts can be learned) and *low storage costs*, as continuous prompts require several orders of magnitude fewer parameters than the original LLM. We address the following research question:

RQ: *To what extent can prefix-tuning serve as a parameter-efficient alternative to finetuning for data wrangling tasks?*

We first describe how to automatically learn continuous prompts for LLMs applied to data wrangling tasks via prefix-tuning (Section 3). Next, we experimentally explore the potential of prefix-tuning on ten data wrangling tasks compared to finetuning in Section 4. A major obstacle for our work is that the current state-of-the-art approach uses prompting with the proprietary GPT-3 [3] model. GPT-3 is only accessible for inference through an API that does not support continuous prompts. As a consequence, we use the smaller publicly available T5 [16] model as an alternative. Our goal is not to beat the state-of-the-art achieved by GPT-3, but rather to introduce a parameter-efficient and automated way to learn continuous prompts. Despite the limitations, our results from prefix-tuning on T5 are promising:

We find that in five out of ten cases, prefix-tuning is within 2.3% of the performance of finetuning, even though it leverages only 0.39% of the parameter updates required for finetuning the full model.

We discuss the implications of these findings and outline directions for follow-up research on further automating data wrangling with LLMs in Section 5. We make the code for our approach and experiments available to the public at `https://github.com/davidvos/prefix-tuning-for-data-management/`.

2 Background

Large language models (LLMs) are neural networks pre-trained on large quantities of raw text data using a masked word prediction tasks. Examples are BERT [7], RoBERTa [11] and T5 [16], which leverage a transformer architecture with with hundreds of millions of parameters. Since data wrangling often involves string-based operations, various recent approaches leverage LLMs [10, 13, 12]

When utilizing an LLM for a downstream task, it can be finetuned by updating all the model parameters on a task-specific dataset. This is expensive as it requires maintaining a copy of all the model parameters for each separate task. Several parameter-efficient alternatives to fully finetuning an LLM have been proposed. An example is adapter-tuning, which inserts additional layers (adapters) between the layers of the LLM and optimizes only those. With around 3.6% of the original LLM parameters, this method still results in relatively high memory costs while achieving worse performance on common data wrangling tasks compared to finetuning [14].

Recently, OpenAI introduced GPT-3 [3], an LLM with hundreds of billions of parameters, which often provides state-of-the-art performance using manually engineered prompts without finetuning. Prefix-tuning takes inspiration from prompt engineering as it casts the manual selection of prompts to a continuous optimization problem. Automatically updating a prefix allows for LLM data wrangling approaches to be deployed at scale, compared to manually engineered prompting techniques.

3 Approach

In the following section, we introduce the three data wrangling tasks in the focus of this work, and detail how to automatically learn continuous prompts for them.

As already discussed, prefix-tuning has been proposed as a lightweight alternative to finetuning LLMs for natural language generation tasks. The parameters of the LLM are frozen, and only a small continuous task-specific prefix is updated. Instead of manually engineering a prompt that is prepended to the original input sequence, prefix-tuning allows for learning a continuous prompt consisting of 'virtual' tokens.

Data wrangling as text generation. Assume we have an LLM with an encoder-decoder architecture to perform a data wrangling task. We follow the setup proposed in [14] to cast data wrangling tasks to text-generation tasks. We serialize the attributes and values from each input tuple as: `serialize(tuple) = attribute-1: value-1 ... attribute-m: value-m`.

Entity matching. Entity matching (EM) is crucial for combining different data sources during data integration. The task is to predict whether two tuples refer to the same real-world entity, e.g., a product or song. To use LLMs, a textual input is created as follows: `Product A: serialize(tuple-1). Product B: serialize(tuple-2). Are product A and product B the same?`. Depending on the domain, we replace the term `Product` with a more appropriate term (e.g. `Song` for music data), following the approach of Narayan et al [14]. The classification targets are converted to text as well (`Yes` or `No`).

Error detection. Error detection (ED) predicts whether a certain attribute of a database tuple contains an error [1]. For example, if a tuple has the country value `Germany` but the attribute `capital` contains the value `Amsterdam` instead of `Berlin`, an ED algorithm should classify this value as an error. We convert the ED problem to the following input format: `serialize(tuple). Is there an error in attribute-j: value-j?`, and the textualised classification targets are either `Yes` or `No`.

Data imputation. Data imputation (DI) fills in missing values for textual or categorical attributes. The following input format is used to cast the DI problem to a text generation problem: `serialize(tuple) attribute-j?`, where `attribute-j` is the attribute for which the value is predicted. The target label is the value corresponding to `attribute-j`.

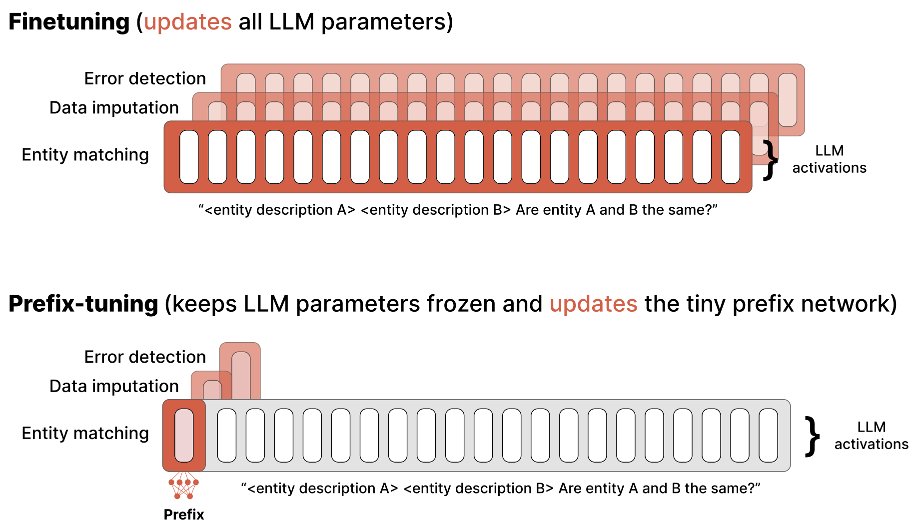

Figure 1: Prefix-tuning compared to finetuning. For finetuning, all activations are based on the updated LLM weights and a separate LLM copy is stored for each new task. When using prefix-tuning, only the prefix parameters are updated and copied for new tasks. The LLM parameters are frozen and activations are conditioned on the newly introduced prefix.

Learning continuous prompts via prefix-tuning. Let ϕ denote the parameters of the pre-trained LLM. Prefix-tuning freezes the LLM parameters ϕ and instead initializes a trainable matrix P_θ, parametrized by θ. The matrix P_θ contains N_{prefix} prefix vectors. N_{prefix} can be considered a hyperparameter whereas the size of each prefix vector should be the same size as the embedding used withing the LLM. Intuitively, this sequence of vectors is prepended to the sequence of tokens that represent the textual task-based input samples as previously introduced.

To be more precice, let P_{idx} represent the indices corresponding to the prefix. We can now compute the activation h_i at time step i as follows:

$$h_i = \begin{cases} P_\theta[i,:] & \text{if } i \in \text{P}_{\text{idx}} \\ LLM(z_i, h_{<i}) & \text{otherwise} \end{cases}$$

Here $LLM(z_i, h_{<i})$ computes the hidden state h_i of the LLM based on the hidden states from the left context $h_{<i}$ and the current token to be processed z_i. In cases where $i \in \text{P}_{\text{idx}}$, h_i is taken from the trainable P_θ. However, even when $i \notin \text{P}_{\text{idx}}$, the prefix activations P_θ are still in the left context $h_{<i}$ and influence all following activations.

In practice, the prefix P_θ is generated by a small neural network. The network takes a constant, predefined vector of integers as input and converts it into an embedding. This embedding is then forwarded through another small neural network to increase the training stability [9]. The training proceeds as usual, by passing the prefix together with the actual sample through the LLM, back-propagating the error and updating the prefix network parameters. After training, only the parameters of the prefix network θ need to be stored to generate the task-prefix. The size of θ is typically less than a percent of the size of the LLM weights ϕ.

4 Experimental Evaluation

4.1 Experimental Setup

Datasets and metrics. We experiment with the datasets and their corresponding data splits as recommended by Narayan et al. [14], listed in Table 1. We evaluate the performance for the binary classification problems EM and ED via the F1-score on the test set, and measure the performance for DI via the accuracy of the generated values to impute. We use accuracy for DI, as the labels are arbitrary pieces of text and not classification values.

Table 1: Datasets with their corresponding task, domain and label distribution.

Task	Dataset	Domain	#Samples	Frac. Positive
Entity matching	Beer	food	450	15.1%
	iTunes-Amazon	music	539	24.5%
	Fodors-Zagats	food	946	11.6%
	Walmart-Amazon	electronics	10,242	90.6%
	Amazon-Google	software	11,460	10.2%
	DBLP-ACM	citation	12,363	25%
	DBLP-Google	citation	28,707	18.6%
Error detection	Hospital	healthcare	19,000	2.7%
Data Imputation	Buy	electronics	651	-
	Restaurant	address	864	-

Architectures and hyperparameters. We use the T5-base implementation from Hugging Face [20]. For all training procedures, we apply the AdamW optimizer and a linear learning rate scheduler, as recommended by the default Hugging Face setup. We train each setup for 50 epochs with a batch size of 16, a learning rate of $5 \cdot 10^{-4}$, and a prefix-length of 100. Upon generation time, we use beam search with a beam size of 5. We use these hyperparameters for both prefix-tuning and finetuning. We try different learning rates ($5 \cdot 10^{-3}$, $5 \cdot 10^{-4}$and $5 \cdot 10^{-5}$) and choose $5 \cdot 10^{-4}$ based on validation metrics. For the testing phase, we leverage the model with the highest validation F1-score after 50 epochs of training. The prefix-tuning setups for the Restaurant, Amazon-Google and Walmart-Amazon datasets showed no clear convergence after 50 epochs of training. For this reason we trained these three setups for a total of 100 epochs.

Baseline methods. To assess the performance of prefix-tuning T5 on data wrangling tasks, we compare it to two benchmarks. Firstly, we compare prefix-tuning to finetuning T5. Finetuning is automatable using a similar training procedure as prefix-tuning, but requires 256 times more parameters. For this reason, prefix-tuning T5 should at least come close to the performance of a full finetuning procedure in order to be relevant in practical data wrangling settings.

We compare prefix-tuning T5 to the zero-shot prompting results achieved using GPT-3 [14]. The T5 model makes it hard to compare prefix-tuning to zero- or few-shot prompting as it does not support this for any other task than the 18 tasks it was pre-trained on [16]. We found empirically that indeed zero-shot prompting T5 on new data wrangling tasks lead to suboptimal results. Comparing prefix-tuning T5 (220M parameters) to zero-shot prompting GPT-3 (175B parameters) gives an idea of what can be achieved when prefix-tuning can be applied to GPT-3. Zero-shot prompting does not require any training (only the design of an adequate prompt), and is therefore very attractive from an automation perspective. A method like prefix-tuning, which requires training and introduces additional parameters to learn must therefore outperform such zero-shot prompting by a large margin to justify its additional cost. We conduct such a comparison for validating our approach.

4.2 Results

Prefix-tuning against finetuning. Table 2 lists our results, ranked by the relative performance of prefix-tuning compared to finetuning. In five of the ten datasets, the performance of prefix-tuning is within 2.3% of finetuning. For eight out of ten datasets, the performance is within 5.2%. Especially for entity matching and error detection, prefix-tuning is able to achieve a performance close to fully finetuning T5. Data imputation seems to be particularly hard for prefix-tuning. However, prefix-tuning performs within 5% of finetuning even for DI. Two exceptions are the results for entity matching on the Amazon-Google and Walmart-Amazon datasets, where prefix-tuning only achieves a relative performance of 90% on these datasets.

Parameter-efficiency. Note that prefix tuning only leverages 0.39% of the parameter updates required for finetuning. Storing all the 222,882,048 parameters of a finetuned copy of T5 takes 892 MB. Our prefix-tuning approach however requires only 864,512 parameters, two orders of magnitude less than the full model, which take up 3.5 MB. As LLMs continually increase in size, the importance

Table 2: Relative performance of prefix-tuning compared to finetuning on ten data wrangling tasks. In five out of ten cases, prefix-tuning is within 2.3% of the performance of finetuning, even though it leverages only 0.39% of the parameter updates required for finetuning the full model.

Task	Dataset	Metric	Prefix-tuning	Finetuning	Rel. Perf.
Entity matching	DBLP-Google	F1-score	0.9517	0.9552	99.6%
Entity matching	DBLP-ACM	F1-score	0.981	0.9876	99.3%
Error detection	Hospital	F1-score	0.9766	0.9912	98.5%
Entity matching	iTunes-Amazon	F1-score	0.9286	0.9455	98.2%
Entity matching	Fodors-Zagats	F1-score	0.9767	1.000	97.7%
Entity matching	Beer	F1-score	0.8571	0.8966	95.6%
Imputation	Buy	Accuracy	0.9231	0.9692	95.2%
Imputation	Restaurant	Accuracy	0.8488	0.8953	94.8%
Entity matching	Walmart-Amazon	F1-score	0.7961	0.8806	90.4%
Entity matching	Amazon-Google	F1-score	0.6642	0.7436	89.3%

of parameter-efficient alternatives to finetuning becomes even more drastic. For example, GPT-3 already contains 175 billion parameters leading to a memory size of 700 GB. These findings imply that prefix-tuning can be deployed as a parameter-efficient alternative to an expensive finetuning setup for data wrangling, with a minimal loss in performance in many cases.

Discussion. The imputation datasets are challenging, because the range of possible imputation values is not known a priori and they are potentially not contained in the training data [14]. Prefix-tuning is able to achieve a relative performance of 95.2% on one dataset but only 88.3% on the other. As this task requires complex language generation (in contrast to deciding on `Yes` and `No` in the other tasks), we expect that an LLM such as GPT-3 in combination with prefix-tuning can come closer to the full finetuning result. This is because GPT-3 has been shown to accurately generate complex language for new domains, and can generate samples not seen in task-specific training data [3].

The relatively low performance of prefix-tuning on the Amazon-Google and Walmart-Amazon datasets is in line with the findings by Narayan et al.[14]. Fully finetuning an LLM outperforms parameter-efficient techniques using both GPT-3 and T5 for the Amazon-Google dataset. Our findings confirm that currently this dataset is hard for any approaches other than full finetuning. The Walmart-Amazon case shows a similar pattern, albeit less significant.

Prefix-tuning against zero-shot prompting. The results in Table 3 show that prefix-tuning drastically outperforms zero-shot prompting across all tasks, while using the smaller T5 model. These findings validate our expectation that the additional training effort in prefix-tuning translates into significantly higher prediction quality, compared to zero-shot prompting without hand-engineered prompts.

Table 3: Prefix-tuning drastically outperforms (trainingless) zero-shot prompting across all tasks.

Task	Dataset	Metric	Prefix-tuning T5 (220M params)	Zero-shot prompting GPT-3 (175B params)
Entity matching	DBLP-Google	F1-score	0.9517	0.646
Entity matching	DBLP-ACM	F1-score	0.981	0.935
Entity matching	iTunes-Amazon	F1-score	0.9286	0.659
Entity matching	Fodors-Zagats	F1-score	0.9767	0.872
Entity matching	Beer	F1-score	0.8571	0.786
Entity matching	Walmart-Amazon	F1-score	0.7961	0.606
Entity matching	Amazon-Google	F1-score	0.6642	0.543
Imputation	Buy	Accuracy	0.9231	0.846
Imputation	Restaurant	Accuracy	0.8488	0.709
Error detection	Hospital	F1-score	0.9766	0.069

5 Discussion & Next Directions

We obtain a positive response to our research question. The experimental results indicate that prefix-tuning can serve as a parameter-efficient alternative to finetuning for entity matching, error detection and data imputation. Prefix tuning only fell behind finetuning by more than 5% for two specific datasets.

The fact that prefix-tuning only requires 0.39% of the amount of the parameters required for finetuning an LLM means that this approach is easier to scale. In scenarios, where a large enterprise requires a data wrangling solution for thousands of tables, finetuning a model for each table is too expensive in terms of storage. Introducing prefix-tuning on the other hand can reduce the storage requirements by a factor of more than 250 without a big drop in performance. Finetuning can still be used to ensure optimal performance for high values tables with critical data. A down-side that prefix-tuning shares with finetuning is a high training cost, as errors need to be back-propagated through the whole network during training.

Optimizing continuous prompts for LLMs shows promising results for data wrangling tasks. Current state-of-the-art data wrangling approaches use models much larger than T5 (GPT-3) with manually engineered textual prompts. As prefix-tuning is inspired by prompt engineering, we expect that the performance of this method extends to models like GPT-3, and could possibly beat the state of the art. Li et al. [9] show that prefix-tuning extends well from a small version to a large version of GPT-2, implicating that scaling to GPT-3 should be possible as well. However, the proprietary nature of GPT-3 and its limited API currently make it impossible for us to validate this statement.

The parametrization of prefix-tuning as used in this paper is a basic one, and approaches to give prefix networks more fine-grained control have been proposed. A prominent example are control prefixes [4], which support conditioning on attribute specific information to increase prefix-tuning performance. In future work, we aim to explore the potential of control prefixes to further advance the parameter-efficient automation of data wrangling with LLMs.

Acknowledgements. We want to acknowledge Qualcomm, the ELLIS Unit in Amsterdam and the AIRLab Amsterdam for making it possible to write and present this work.

References

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *VLDB*, 9(12), 2016.

[2] Felix Biessmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. Datawig: Missing value imputation for tables. *JMLR*, 20(175), 2019.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020.

[4] Jordan Clive, Kris Cao, and Marek Rei. Control prefixes for text generation. *arXiv preprint arXiv:2110.08329*, 2021.

[5] Xin Luna Dong and Theodoros Rekatsinas. Data integration and machine learning: A natural synergy. *SIGMOD*, 2018.

[6] Bojan Karlaš, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *VLDB*, 2021.

[7] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT*, 2019.

[8] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *VLDB*, 2016.

[9] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *ACL*, 2021.

[10] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *VLDB*, 14(1), 2020.

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[12] Yinan Mei, Shaoxu Song, Chenguang Fang, Haifeng Yang, Jingyun Fang, and Jiang Long. Capturing semantics for imputation with pre-trained language models. In *ICDE*, 2021.

[13] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond. In *SIGMOD*, 2021.

[14] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911*, 2022.

[15] Laurel Orr, Karan Goel, and Christopher Ré. Data management opportunities for foundation models. *CIDR*, 2021.

[16] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140), 2020.

[17] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *VLDB*, 10(11), 2017.

[18] Cedric Renggli, Luka Rimanic, Nezihe Merve Gürel, Bojan Karlaš, Wentao Wu, and Ce Zhang. A data quality-driven view of mlops. *IEEE Data Engineering Bulletin*, 2021.

[19] Michael Stonebraker, Ihab F Ilyas, et al. Data integration: The current status and the way forward. *IEEE Data Engineering Bulletin*, 41(2), 2018.

[20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. *EMNLP*, 2020.

Appendix

In addition to the metrics already reported, we provide the precision and recall results for the entity matching and error detection tasks.

Table 4: Entity matching results by precision, recall and F1-score.

Dataset	Metric	Prefix-tuning	Finetuning
Amazon-Google	Precision	0.5946	0.7436
	Recall	0.7521	0.7436
	F1-score	0.6642	0.7436
Beer	Precision	0.8571	0.8667
	Recall	0.8571	0.9286
	F1-score	0.8571	0.8966
DBLP-ACM	Precision	0.9734	0.9865
	Recall	0.9887	0.9887
	F1-score	0.981	0.9876
DBLP-Google	Precision	0.9456	0.9435
	Recall	0.9579	0.9673
	F1-score	0.9517	0.9552
Fodors-Zagats	Precision	1.000	1.000
	Recall	0.9545	1.000
	F1-score	0.9767	1.000
iTunes-Amazon	Precision	0.8966	0.9286
	Recall	0.963	0.963
	F1-score	0.9286	0.9455
Walmart-Amazon	Precision	0.7489	0.9022
	Recall	0.8497	0.8601
	F1-score	0.7961	0.8806

Table 5: Error detection results measured by precision, recall and F1-score.

Dataset	Metric	Prefix-tuning	Finetuning
Hospital	Precision	1.000	1.0000
	Recall	0.9542	0.9826
	F1-score	0.9766	0.9912