Extending research - Experimenting on paper: Deep Entity Matching with pre-trained Language Models

Kostas Panoutsakos*
panou10_@hotmail.com
University of Athens - Department of Informatics and Telecommunications
Greece

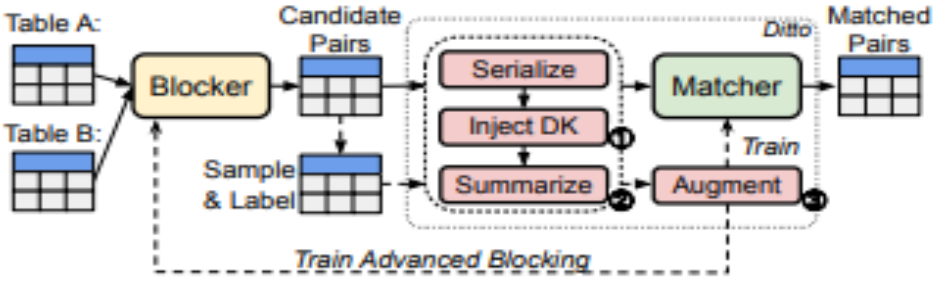

Figure 1. An EM system architecture with Ditto as the matcher, [1] (https://arxiv.org/pdf/2004.00584.pdf).

Abstract

This report addresses the research topic of validating the results as well further improving the architecture of Ditto [1]. Ditto is a novel entity matching system scaled on pre- trained Transformer-based language models that was proposed in 2020. It enabled fine-tuning and casting EM as a sequence-pair classification problem to leverage such models with a simple architecture, and managed to achieve significant results on benchmark datasets, improving remarkably the matching quality of previously proposed SOTA EM solutions. Further to that, it developed three optimization techniques for more robust matching - capable performance:

- `Data Augmentation (DA)`
- `Domain Knowledge (DK)`
- `Summarization (SU)`

In my research I deal with the matcher's architecture. I improve the fine - tuning method of the pre - trained Transformer-based language models. I develop 4 alternative NNs for that, and compare them to the Original Ditto's proposal on various benchmark datasets, while keeping the rest conditions stable (DA - DK - SU). I manage to ameliorate original Ditto's performance by up to 3.58 % on certain benchmark datasets, while requiring approximately the same training time for my models.

Keywords: datasets, neural networks, Language Models, LSTM, GRU

*Post- Graduate Student

1 Introduction

The main goal of my research was to familiarize myself with DITTO's architecture, namely how it tackles the entity matching (EM) recognition problem (Examples are depicted in Figure 2). DITTO, like any sophisticated EM solution, includes a blocker (for a quick, low-cost pruning of the highly unlike pairs) and a matcher. After conducting several execution routines to understand the whole system's philosophy, I decided to deal specifically with the matcher's improvement, which is oriented to the SOTA solutions that the authors proposed, and in particular, the neural network that makes the final choice on whether there is a match or not. That is, I have kept intact the authors' robust proposal on how to provide input to the matcher (Serialization Technique), as well as all the optimization techniques that may be implemented before the matcher (DA – DK – SU), as noted in the abstract. At this point, it is necessary to remind the serialization technique proposed by DITTO authors, which is:

To serialize a candidate pair (e, e), we let
serialize(e, e) ::= [CLS] serialize(e) [SEP] serialize(e) [SEP],
where [SEP] is the special token separating the two sequences and [CLS] is the special token necessary for BERT to encode the sequence pair into a 768-dimensional vector which will be fed into the fully connected layers for classification.

Contributions In summary, the following are my contributions:

- I present four sub - DITTOs, (EM solutions based on pre-trained language models (LMs) such as RoBERTa[2] or DistilBERT[3]) that fine-tune and cast EM as a sequence-pair classification problem to provide deeper language understanding. More specifically, I take advantage of more tokens than just the [CLS] token produced after implementing the pre-trained LM to our serialized tokens.
- In the first case, 3 bi-directional **LSTM layers** are implemented in the whole sequence of tokens, with a 20 % Dropout layer intervening between each couple of them.
- In the second case, a bi-directional **GRU layer** is implemented in the whole sequence of tokens.
- In the third case, I develop a **mapping technique** that is capable of interpolating acquired knowledge from both **[CLS] and connecting [SEP]** tokens into a new tensor, which is then fed to a traditional linear layer.
- In the fourth case, I mix cases 2 and 3, meaning that I use a **GRU Layer** as the final one, but this time it receives as an input only **two tensors, the initial [CLS] and the intermediate [SEP] token that separates the two serialized entities.**
- Finally, I open-source my Ditto variation at https://github.com/nhvd3500111/ditto.

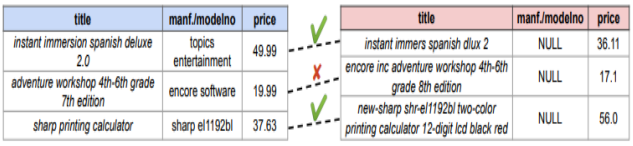

Figure 2. Entity Matching: determine the matching entries from two datasets,
[1] (https://arxiv.org/pdf/2004.00584.pdf).

2 Presenting the Models

Before analyzing our models' structure, let's remember the original DITTO's architecture as presented in Figure 3.

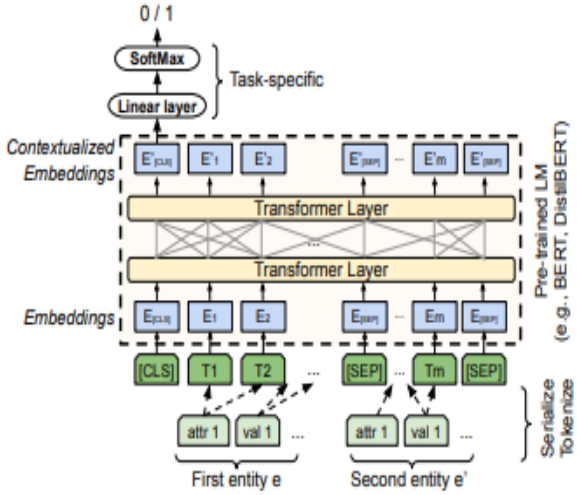

Figure 3. Ditto's Original architecture,
[1] (https://arxiv.org/pdf/2004.00584.pdf).

The above figure illustrates a philosophy quite simple at its core. Capturing only the [CLS] token among all the tokens of the sequence, and passing it through a linear layer with a softmax activation function at the end that performs the classification task. Now, let's compare it to my proposals.

As noted in the introduction, there are *four different submodels* that I developed to further experiment with DITTO's architecture. Let's analyze and portray each.

2.1 LSTM Model

My first idea was to generally introduce a Recurrent Neural Network on top of the pre-trained LM, rather than just a simple linear layer. That is because, an RNN would facilitate me to give the whole sentence (sequence of entities) as an input for prediction rather than just one word. Due to the facts that:

- I might had long-distance information dependencies as input sequences and
- I wanted to tackle a possible vanishing gradient problem,

I preferred an **LSTM [4] layer** over a traditional recurrent neural network. My final proposal included 3 LSTM layers on top of the pre-trained LM, and 20% Dropout [5] layers between each couple of them, serving as a very computationally cheap and remarkably effective regularization method to reduce overfitting and improve generalization error.
The architecture of my first model is depicted in Figure 4.

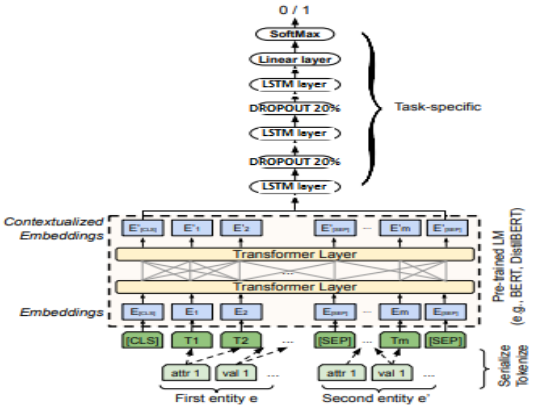

Figure 4. Model 1 - Pre-Trained LM with an LSTM Architecture with Dropout Layers on Top

Thus, the central plank of this strategy was to utilize the whole sequence of tokens rather than the first initial [CLS] token. In that way, I can capture the serialized information from as many as 512 tokens (max length limit of tokens for BERT models), instead of just one.

2.2 GRU Model

The next idea introduces a simpler architecture than the one presented in the previous section. Once again I want to take advantage of the benefits that RNN's mechanism offers, together with the ability to learn long term sequences, but without the gradient overflow problem.
Enters **GRU [6] mechanism** (Gated Reccurent Unit). A sophisticated variation of RNN that includes two gates (update gate and reset gate) instead of RNN's three. Basically, these are two vectors which decide what information should be passed to the output. GRU does not possess any internal memory and does not have an output gate that is present in LSTM. This idea is presented in figure 5 below:

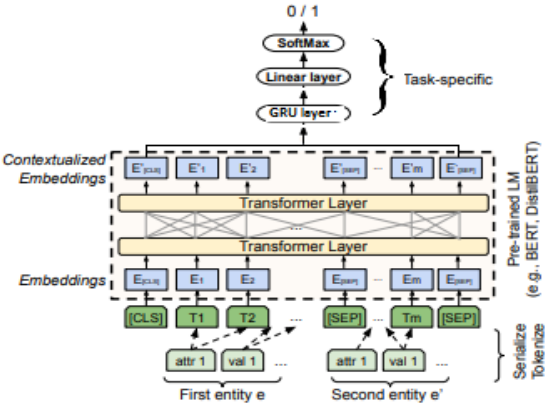

Figure 5. Model 2 - Pre-Trained LM with a GRU architecture on top

It is a simple GRU architecture right after the pre-trained LM. It captures knowledge from the entire sequence of tokens, as long as it does not reach the maximum length of 512 (BERT constraint). It is bi-directional as well, meaning that there are actually two sub-GRUs in one, with the first taking the input in a forward direction and the other in a backward direction. In the end, a linear layer captures the representation produced by the GRU layer (twice the hidden size of GRU - $2 \cdot 128 = 256$ dimensions). Finally, this feeds a softmax function that will classify whether the sequence of entities is a match or not.

2.3 CLS_SEP Architecture

In addition to the more sophisticated Recurrent Layers, another choice was to keep the simple architecture of the linear layer, but feed it with tensors of different dimensions. These tensors will result from the two vectors representing [CLS] and intermediate [SEP] token, interpolated with a map function of equal weight (Figure 6 below).

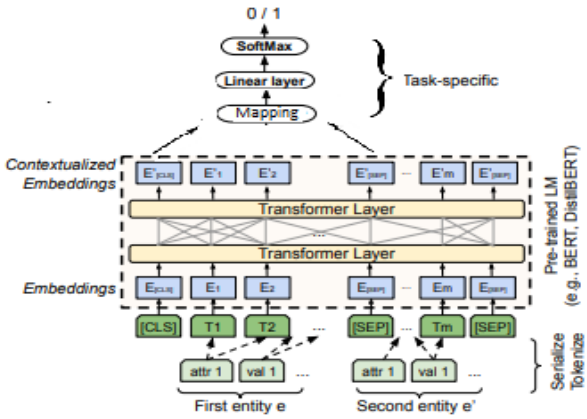

Figure 6. Model 3 - Capturing information from the [CLS] and the intermediate [SEP] token produced by a pre-trained LM and feeding it to a linear layer on top

To begin with, [CLS] token itself captures a lot of information regarding our task. CLS stands for classification, it is there to represent sentence-level classification and was introduced to make pooling scheme of BERT work. However, the idea was that [SEP] token, that is actually a separator between the two sub - sequences, hides a lot of information regarding the whole sequence too. Therefore it would be good research start to feed the linear layer with both tokens' representations.

2.4 CLS_SEP Architecture combined with a GRU layer

Last but not least, I proceed with the bold experimentation to combine architectures presented in sections 2.2 and 2.3. This is a little unorthodox as a thought at its core because GRU has proven its efficiency when fed with a long sequence

of token representations. In our case, I provide only two tokens as input to the GRU layer, after their representations are produced by the pre-trained LM. However, this solution provided robust results in some datasets as well. The architecture is depicted in figure 7.

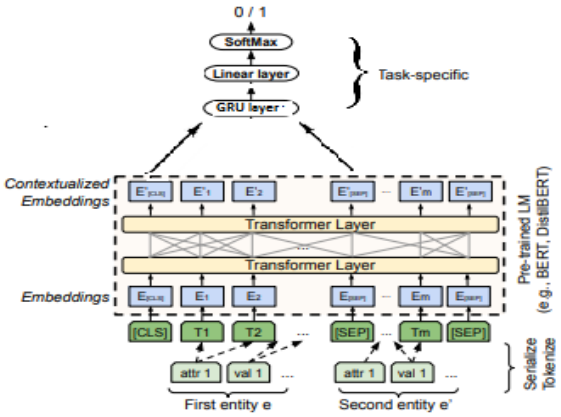

Figure 7. Model 4 - Capturing information from the [CLS] and the intermediate [SEP] token produced by a pre-trained LM and feeding it to a GRU layer on top

Of course, similarly to the plain GRU architecture introduced in section 2.2, the GRU layer is followed by a linear layer and a softmax function that decides the class.

3 Experiments

3.1 Setup

The original DITTO repo was forked, and then many modifications were introduced in order to customize the experiments' setup and try all four variations. Some essential alterations are:

- Each time train_ditto is executed, the user has to provide additional arguments. The most important of them is **neural**, which selects the type of DITTO architecture that will be exploited when training.
- Another argument provided is the name of an .xlsx file where the results of each training session will be stored. This file is in the same directory where matcher.py is stored.
- Four more .py files, similar to ditto.py were created and stored into the ditto_light directory (ditto.py was renamed to ditto_original.py to avoid having a namesake with the others). Inside every file, each DITTO's structure is further developed. Of course, depending on the user's input when calling train_ditto.py, the respective file will be called and executed.
- Our experiments are validated when the user executes matcher.py. Another feature was introduced in that file, that receives the optimized threshold for the softmax function (it is calculated when the model has been created, on validation data) and returns the f1_score on the respective testset, given this new threshold.
- Since gru and lstm receive as input a batch of sequences of tokens, padding was added in dataset.py file. Each batch contains sequences of different lengths, so a formalization to the maximum sequence length of the batch is required. This handy feature can adapt depending on each batch's characteristics. More specifically, it pads to the longest sequence in the batch, or, if only a single sequence is provided, it does not pad at all. Therefore, it can satisfy all 5 DITTO's variations (original and the four newly proposed).

Of course, many other minor modifications were introduced that can not be mentioned here for simplicity's sake.

3.2 Datasets

The models were tested on the following datasets:

- wdc_watches_small
- wdc_computers_small
- wdc_shoes_small
- wdc_cameras_small
- wdc_all_medium
- Structured/iTunes-Amazon
- Structured/Beer
- Dirty/Walmart Amazon

On every run, different variations were executed, proposing distinct data augmentation or domain knowledge techniques.

3.3 Assumptions

Additional prerequisite information regarding the training sessions:

- The newly introduced code **can be executed only on a CUDA environment.**
- There is a known bug in General DK Injector class included in knowledge.py, therefore this technique **should not be used.**
- Training with mixed precision (aka enabling the FP16 optimization) **should be avoided when sessions are executing using kaggle GPUs** due to inconsistent results (occasional excessive memory use).

3.4 Results

Although I will elaborate on the results of the experiments in the Appendix Section, it should be mentioned that in all tests, **there was at least one new model that achieved a better F1 Score than the Original Ditto.**
But what is more impressive is that there is a proposed architecture that **managed to outperform the original in all experiments but one, and that is the GRU** discussed in section 2.2.
Regarding the time required for each training session, the

differences observed depending on each model were bearable and insignificant. Nevertheless, these measurements may lack credibility, since many sessions with different CUDAs were enabled, and I did not have a stable Server at my disposal to conduct the experiments.

4 Conclusion

I presented four different model structures based on fine-tuning pre-trained Transformer-based language models. They leverage pre-trained LMs and are further optimized by injecting domain knowledge, text summarization, and data augmentation. My results show that in most tested cases, they outperform the original DITTO proposal. Of course, due to the lack of available GPU time, the conducted experiments were limited to a small number considering all the datasets and all the different (DK-DA-SU) techniques available. Additionally, I plan to extend these experiments and cover every possible execution plan for each architecture in order to come to a more robust conclusion on whether I have developed an architecture that should definitely substitute the authors' proposal.

References

[1] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan and Wang-Chiew Tan. 2020. *Deep Entity Matching with Pre-Trained Language Models.* arXiv preprint arXiv:2004.00584 (2020).

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

[3] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* In Proc. *EMC*^{2} '19.

[4] Sepp; Hochreiter and J¨urgen Schmidhuber. 1997. *Long Short-Term Memory* Neural computation, 9(8):1735–1780, 1997.

[5] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* JMLR, 2014.

[6] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. 2014. *On the properties of neural machine translation: Encoder-decoder approaches.* arXiv preprint arXiv:1409.1259, 2014.

A Research Results

A.1 Stable Parameters

Every training session with its specific parameters was executed twice in order to obtain more robust and credible results (modifying only CUDA's random seed). When conducting the experiments, the following arguments remained intact in all training sessions:

- number of Epochs: 20
- batch Size: 32
- learning Rate: 3e-5
- batch Size: 32
- max_len: 128

Regarding max_len, this number is doubled if any data augmentation technique has been enabled, leading to a maximum allowed number of 256 tokens per sequence. After the serialization of the couple of entities examined, the pre-trained LM model produces its representation, and if that leads to a tokens' length>128, then the sequence is truncated to tokens' length=128.In case a GRU or an LSTM layer is exploited, each batch of sequences is padded to the batch's longest sequence, which in any case can not include more than 256 tokens.

A.2 F1 Score

Let's examine the results of the experiments regarding F1 score on testset, presented per dataset.

A.2.1 wdc_cameras_small . In this dataset, I used DistilBERT for all experiments as a pre-trained LM. In the following table we can see the average F1 in all conducted experiments on wdc_cameras_small dataset, regardless DA - DK techniques. **FP16 Optimization was not enabled**.

Table 1. Overall Results wdc_cameras_small

Model_Architecture	F1_Testset
cls_sep_gru	0.80490
gru	0.79842
linear	0.78934
cls_sep	0.78534
lstm	0.75776

The DA technique in all experiments was entry_swap. In some experiments Product DK was infused. The following diagram presents the average results subject to the optimizations of the experiment.

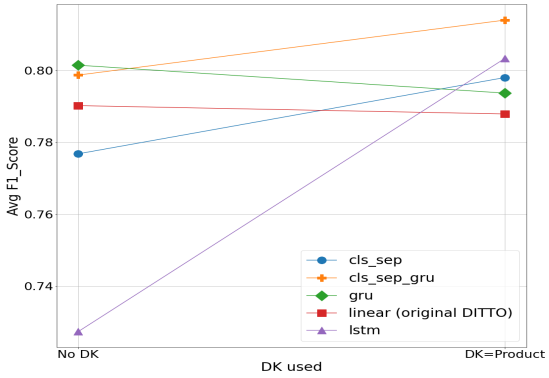

Figure 8. Average F1 Score on wdc_cameras_small test dataset depending on DK infusion (DA=entry_swap for all testings)

It is clear in Figure 8 that both GRU and CLS_SEP_GRU models outperformed original DITTO in all experiments.

A.2.2 wdc_computers_small . In this dataset, I used DistilBERT for all experiments as a pre-trained LM. In the following table we can see the average F1 in all conducted experiments on wdc_computers_small dataset, regardless DA - DK techniques. **FP16 Optimization was not enabled**.

Table 2. Overall Results wdc_computers_small

Model_Architecture	F1_Testset
lstm	0.8277
gru	0.81425
linear	0.8134
cls_sep_gru	0.80375
cls_sep	0.799775

The DA technique in all experiments was entry_swap. In some experiments Product DK was infused. The following diagram presents the average results subject to the optimizations of the experiment.

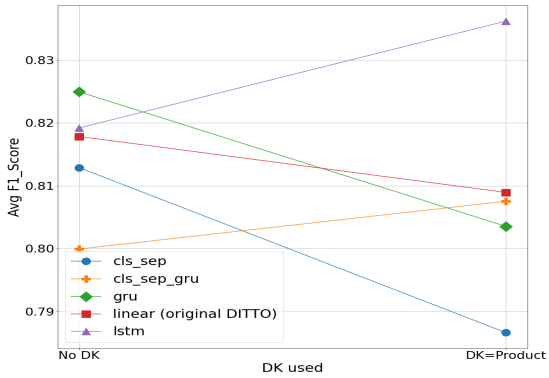

Figure 9. Average F1 Score on wdc_computers_small test dataset depending on DK infusion (DA=entry_swap for all testings)

In the above figure, we can see the only subset of experiments that GRU architecture underperformed the original. This was for DA=entry_swap and DK_Infusion=Product. In other respects, we can see that LSTM structure delivered more robust results in all cases of this dataset.

A.2.3 wdc_watches_small . In this dataset, I used DistilBERT for all experiments as a pre-trained LM. In the following table we can see the average F1 in all conducted experiments on wdc_watches_small dataset, regardless DA - DK techniques. **FP16 Optimization was not enabled**.

Table 3. Overall Results wdc_watches_small

Model_Architecture	F1_Testset
gru	0.856125
lstm	0.8495
cls_sep	0.8389
cls_sep_gru	0.8369
linear	0.832975

Once again GRU architecture manages to outperform the original linear in the overall results, this time by more than 2.3 %. The DA technique in all experiments was entry_swap. In some experiments Product DK was infused. The following diagram presents the average results subject to the optimizations of the experiment.

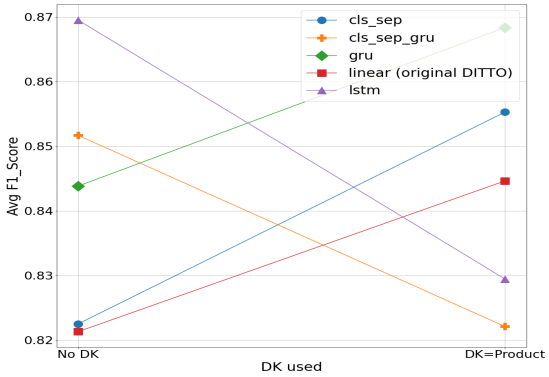

Figure 10. Average F1 Score on wdc_watches_small test dataset depending on DK infusion (DA=entry_swap for all testings)

In the above figure, we can see that GRU overtoped the original method in all sub-experiments of wdc_watches_small dataset.

A.2.4 wdc_shoes_small . In this dataset, I used DistilBERT for all experiments as a pre-trained LM. **FP16 Optimization was not enabled**. Table 4 shows the average F1 in all conducted experiments on wdc_shoes_small dataset, regardless DA - DK techniques.

Once again GRU architecture manages to outperform the original linear in the overall results, this time by more than 1.1 %. The DA technique in all experiments was entry_swap. In some experiments Product DK was infused. Figure 11 presents the average results subject to the optimizations of the experiment.

Table 4. Overall Results wdc_shoes_small

Model_Architecture	F1_Testset
cls_sep_gru	0.763
gru	0.75125
cls_sep	0.7475
linear	0.7407
lstm	0.7407

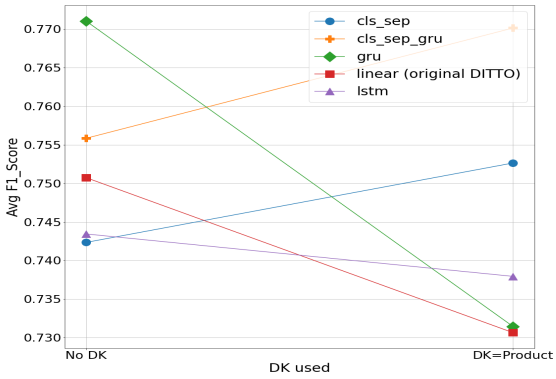

Figure 11. Average F1 Score on wdc_shoes_small test dataset depending on DK infusion (DA=entry_swap for all testings)

In the above figure, we can see that GRU performed better than the original method in all sub-experiments of the dataset.

A.2.5 wdc_all_medium . For the final set of experiments on WDC product data corpus, I decided to dig deeper and analyze wdc_all_medium dataset. It contains 25,567 entities, which is almost ten times larger than every one of the four small datasets that I previously examined. **It is the combination of all wdc_medium datasets (wdc_shoes_medium, wdc_watches_medium, wdc_computers_medium and wdc_cameras_medium)**
Due to the significantly big training time, I ran 2 experiments for each setup and compared my most prominent architecture (GRU) to the original.
I used DistilBERT for all experiments as a pre-trained LM. **FP16 Optimization was not enabled**. Table 5 presents the average F1 in all conducted experiments on wdc_all_medium dataset, regardless DA - DK techniques.

Table 5. Overall Results wdc_all_medium

Model_Architecture	F1_Testset
gru	0.89365
linear	0.8927

We can see that GRU performed slighlty better than the original architecture. I was expecting a decrease in GRU's performance as long as the size of the Dataset increases, since GRU is a very robust and clean solution for small datasets. However, we can see that it did pretty well in a relatively large dataset as well. In the future, I will test my solution on wdc_all_xLarge and see how it behaves. A good trade-off is, after surpassing a pre-defined dataset size, to switch from GRU to LSTM architecture since LSTM layers use more training parameters and therefore are more accurate on a larger dataset, by all accounts.

A.2.6 Structured/iTunes-Amazon . In this dataset, I used both DistilBERT and RoBERTa as pre-trained LMs. In the following table we can see the average F1 in all conducted experiments on Structured/iTunes-Amazon dataset, regardless DA - DK techniques. **FP16 Optimization was enabled**.

Table 6. Overall Results Structured/iTunes-Amazon

Model_Architecture	F1_Testset
cls_sep	0.9480
gru	0.94282
linear	0.93758
cls_sep_gru	0.92366
lstm	0.79244

For every expeiment, no DK was infused in the Dataset. Some experiments enabled append_col as the DA method and others del_col method. The following diagram presents the average results subject to the optimizations of the experiment.

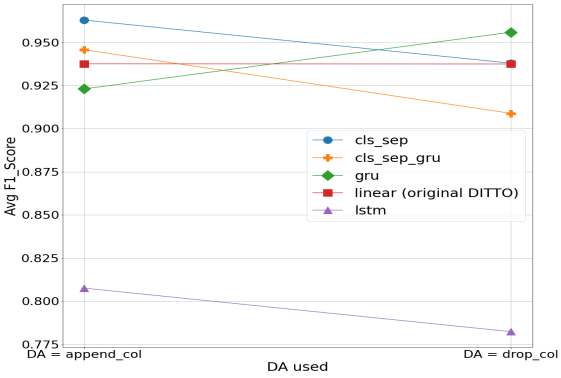

Figure 12. Average F1 Score on Structured/iTunes-Amazon test dataset depending on DA method (No DK infusion for all testings)

It is clear from Figure 12 that many custom models earned better results than the original DITTO in all sub- experiments.

A.2.7 Structured/Beer . In this dataset, I used only DistilBERT as the pre-trained LM. In the following table we can see the average F1 in all conducted experiments on Structured/Beer dataset, regardless DA - DK techniques. **FP16 Optimization was enabled**.

Table 7. Overall Results Structured/Beer

Model_Architecture	F1_Testset
cls_sep	0.68265
gru	0.64880
linear	0.62385
cls_sep_gru	0.59495
lstm	0.2667

No DK was infused in the Dataset on every experiment. Some experiments enabled append_col as the DA method and others del_col method. It is clear in Table 7 that CLS_SEP managed to perform almost 6% better than the original DITTO. Moreover, my stable proposal which is robust in most cases (GRU architecture) produced 2.5 % better results than the original. The following diagram presents the average results subject to the optimizations of the experiment.

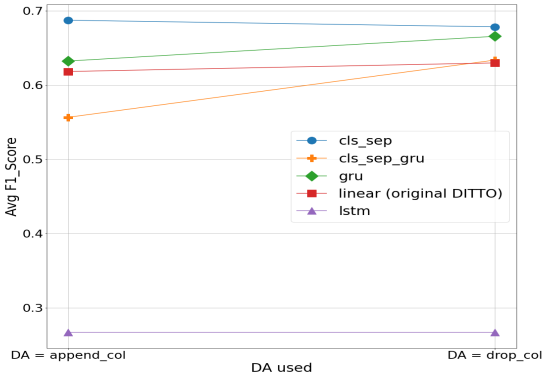

Figure 13. Average F1 Score on Structured/Beer test dataset depending on DA method (No DK infusion for all testings)

One more time, Figure 13 illustrates the high quality of the GRU structure.

A.2.8 Dirty/Walmart-Amazon . In this dataset, I used only DistilBERT as the pre-trained LM. I wanted to check my solution's F1 Score on Dirty data with missing (null) values, values that are placed under the wrong attributes or values with some tokens missing. In the following table we can see the average F1 in all conducted experiments on Dirty/Walmart-Amazon, regardless DA - DK techniques. **FP16 Optimization was enabled**. Due to the significantly big training time, I ran 2 experiments for each setup and compared my most prominent architecture (GRU) to the authors' original proposal.

Table 8. Overall Results Dirty/Walmart-Amazon

Model_Architecture	F1_Testset
gru	0.7869
linear	0.781315

For every expeiment, no DK was infused in the Dataset. Some experiments enabled del as the DA method and others the swap method. Table 8 presents a lead for the GRU method of approximately 0.5 %.

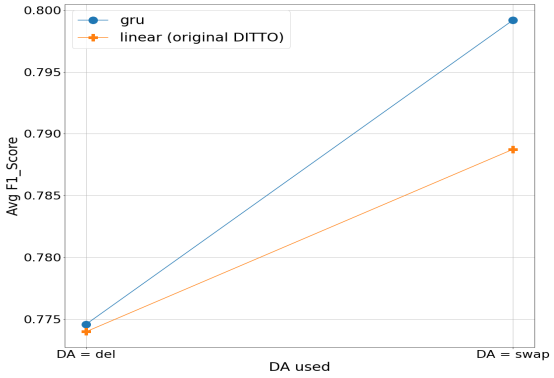

Figure 14. Average F1 Score on Dirty/Walmart-Amazon test dataset depending on DA method (No DK infusion for all testings)

As depicted above, GRU is the winner in both sub - categories of experiments.

A.3 Training Time

As mentioned in Section 3.4, although I calculated the time of execution of each training session, this is not very accurate due to the lack of an organized setup with its own GPU. However, in Table 9, I present the total time required for all training processes, grouped by the model's architecture.

Obviously, there was much more training time consumed for the GRU and the Original architecture, since I conducted experiments on wdc_all_medium and Dirty/Walmart-Amazon dataset only for these models. However, despite the apparent robustness of the GRU structure, the total time of execution is far from prohibitive to try and test it on other datasets as well, since it increases time consumption by barely 2.50 %, compared to the original architecture .

Table 9. Total Time Consumed for each Model's training sessions

Model_Architecture	Total Time (seconds)
cls_sep	9518.75
cls_sep_gru	9535.92
lstm	9260.79
gru	23458.15
linear	22870.40