TransformerRanker: A Tool for Efficiently Finding the Best-Suited Language Models for Downstream Classification Tasks

Lukas Garbas [*] Max Ploner [*†] Alan Akbik [*†]

[*] Humboldt-Universität zu Berlin
[†] Science Of Intelligence
`<firstname>.<lastname>@hu-berlin.de`

Abstract

Classification tasks in NLP are typically addressed by selecting a pre-trained language model (PLM) from a model hub, and fine-tuning it for the task at hand. However, given the very large number of PLMs that are currently available, a practical challenge is to determine which of them will perform best for a specific downstream task. With this paper, we introduce TransformerRanker, a lightweight library that efficiently ranks PLMs for classification tasks without the need for computationally costly fine-tuning. Our library implements current approaches for *transferability estimation* (LogME, H-Score, kNN), in combination with layer aggregation options, which we empirically showed to yield state-of-the-art rankings of PLMs (Garbas et al., 2024). We designed the interface to be lightweight and easy to use, allowing users to directly connect to the HuggingFace transformers and datasets libraries. Users need only select a downstream classification task and a list of PLMs to create a ranking of likely best-suited PLMs for their task. We make TransformerRanker available as a pip-installable open-source library.[1]

1 Introduction

There currently exists a multitude of pre-trained transformer language models (PLMs) that are readily available (e.g. through model hubs; Wolf et al., 2019). From a practical perspective, this raises the question of which PLM will perform best once fine-tuned for a specific downstream NLP task. However, since fine-tuning a PLM is both computationally costly and sensitive to hyperparameters (such as the learning rate used for fine-tuning), an exhaustive search of all models is infeasible. In practice, this restricts users to the exploration of only a small number of PLMs and may lead to the best-suited PLM for a particular task not being found.

[1] `https://github.com/flairNLP/transformer-ranker`. Released under the MIT License.

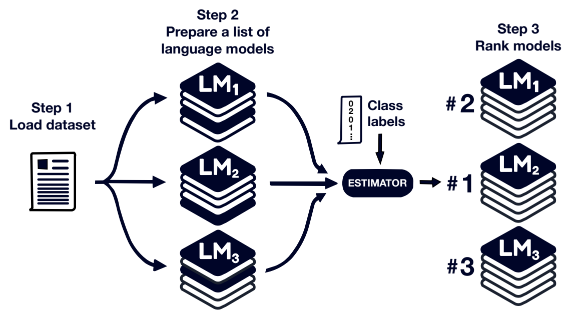

Figure 1: The three steps of TransformerRanker: (1) The user selects a downstream classification task by selecting a dataset from HuggingFace datasets. (2) The user also selects a list of language models from HuggingFace transformers. (3) Using the selected estimator, the library returns a ranking of which PLMs are likely to perform best on the selected task.

To address this issue, prior work proposed methods for *transferability estimation*. These methods avoid the high computational costs associated with fine-tuning a PLM by keeping the internal states frozen. Prominent examples of such methods include H-score (Bao et al., 2019; Ibrahim et al., 2023) and LogME (You et al., 2021). In our prior work, we found that these methods can be improved by incorporating features from deeper layers in the estimation process (Garbas et al., 2024). We empirically showed that this yields better per-model estimates since, depending on the downstream task, different layers in the transformer model are best suited to provide features. Further, we showed that averaging across layers makes the selection process more robust against the different pre-training objectives used in each model, allowing a better comparison over a diverse set of PLMs.

A library for transferability estimation. With this paper, we present TransformerRanker, a Python library that enables users to leverage transferability estimation to identify the best-suited PLM for a downstream classification task. With

TransformerRanker, we consolidate work in layer-wise analysis and transferability estimation methods into a single library and provide a three-step interface for ranking any transformer LMs available on the HuggingFace model hub (see Figure 1). Our goal is twofold:

- First, to give practitioners an easy-to-use method of using transferability estimation to select PLMs for their downstream tasks. To this end, we designed a simple interface that directly connects to the HuggingFace Transformers and Datasets libraries. From these, users need only select a downstream classification task and a list of PLMs. Using our default settings, the library will output a ranking of likely best-suited PLMs for their task.
- Second, to assist researchers in the field of transferability estimation by having a single library that implements multiple state-of-the-art estimators and aggregation methods to combine and compare against.

TransformerRanker supports NLP classification tasks of two main families: *(1)* Text classification tasks such as question classification (Voorhees and Harman, 2000), sentiment analysis (Socher et al., 2013) or textual entailment (Wang et al., 2018) in which a classification decision is made for an entire text (or text pair), *(2)* sequence labeling tasks such as named entity recognition (NER; Sang and Meulder, 2003) and part-of-speech tagging (Petrov et al., 2011) where classification decisions are made per-word.

Our library is built to rely solely on PyTorch (Paszke et al., 2019) and HuggingFace (Wolf et al., 2019) ecosystems and can be integrated as a fast model selection step in a larger NLP pipeline. We make TransformerRanker publicly available as a pip-installable open-source project.

2 TransformerRanker

We give an overview of TransformerRanker by first discussing the installation procedure (Section 2.1) and the standard three-step workflow for ranking transformer models (Section 2.2). We then discuss the implemented estimators (Section 2.3) and layer aggregation methods (Section 2.4).

2.1 Setup

TransformerRanker is implemented in Python and requires version 3.8 or higher. To install the library, run the following command: `pip install transformer-ranker`. This will download the latest version and all necessary dependencies, including `torch` and `transformers`. Alternatively, the library can be cloned directly from GitHub.

```
# Step 1. Load Classification Dataset
dataset = load_dataset('conll2003')

# Step 2. Create List of Candidate Models
language_models = [
    'bert-base-uncased',
    'xlm-roberta-large',
    'deberta-v3-base',
]

# Step 3. Run Transformer Ranker
ranker = TransformerRanker(dataset)

result = ranker.run(
    language_models,
    batch_size=64
)

# Inspect the result:
print(result)

# Rank 1. 'deberta-v3-base'
# Rank 2. 'xlm-roberta-large'
# Rank 3. 'bert-base-uncased'
# ...
```

Listing 1: Full code example for finding the best-suited PLM for the CoNLL-03 shared task on NER. We load the respective dataset and define a list of candidate PLMs. We run the ranker using default parameters to rank candidates according to their estimated suitability.

2.2 Three-Step Workflow

We illustrate the three-step workflow using a simple example in which we want to find the best-suited transformer model for the CoNLL-03 shared task of English-language NER (Sang and Meulder, 2003). The full code needed to execute this search is shown in Listing 1.

As the listing illustrates, our interface defines three main steps: First, we select the dataset that provides training and testing data for the task at hand, in this case CoNLL-03. Second, we create a list of language models to rank. Third, we initialize and run the ranker to obtain a ranking that indicates which models are best-suited for CoNLL-03.

Step 1: Load Your Dataset. To cover a large variety of existing NLP datasets, we provide support for those available in the HuggingFace Datasets library. The initial step involves loading an existing or custom dataset, by simply providing the corresponding dataset name (in this example 'conll03'). As Listing 1 shows, no wrapper is introduced; the operations are performed directly on the structure provided by the Dataset and DatasetDict classes as returned by DATASETS.

Step 2: Create List of Candidate Models. The next step is to select the language models of interest. The list can include strings referring to transformers in the HuggingFace model hub or models already loaded using the AutoModel class from the TRANSFORMERS library. In the example in Listing 1, we simply define a set of string identifiers as our candidate models.

Typically, users might chose candidate models by selecting the most popular (i.e., most downloaded models) on the HuggingFace model hub. We generally advise to search over models that were trained with different pre-training objectives and datasets in this list, as this can lead to significant performance differences. To help new users get started, we prepared two predefined lists of language models. The first contains smaller models (i.e. small and base models of popular PLMs), and the second contains large models. To load the list of smaller models, use the following code:

```
# Step 2: Define model candidate list
language_models =
    popular_candidate_models('base')
```

Depending on the computational requirements of the project, we recommend new users to select either from smaller- or larger-size PLMs.

Step 3: Rank Models. The final step is to initialize the TransformerRanker and run it. Initialization requires passing the dataset as a parameter. The dataset is preprocessed internally, retaining only the necessary columns (e.g., text and label). Next, the ranking process is executed by passing the language model list to the run method, together with optional hyperparameters (see Listing 1).

Two hyperparameters may be set to optimize the speed of the ranking approach: First, users must specify a batch_size which indicates how many data points are embedded with a single forward pass. Depending on the available GPU memory, this parameter can be increased for quicker processing. Second, users may provide an optional parameter dataset_downsample which downsamples the data to a percentage of its original size. This may yield significant speedups in model ranking, as we experimentally show in Section 3.3. We provide an example of how to downsample the dataset to 20% of its size in Listing 2.

```
# Initialize the ranker a downsampled dataset
ranker = TransformerRanker(
    dataset,
    dataset_downsample=0.2
)

# Set estimator and layer aggregator and run
result = ranker.run(
    language_models,
    estimator='knn',
    layer_aggregation='bestlayer',
    batch_size=64
)
```

Listing 2: Alternative configuration of Step 3 from Listing 1: The dataset is downsampled to 20% of its original size by passing dataset_downsample=0.2. An alternative estimator and layer_aggregation method are used for ranking.

Additionally, two hyperparameters may be set to configure the transferability estimation approach. By default, H-score is used as the estimator and *layer mean* as the layer aggregation method, a combination we found to yield the best rankings (**?**). We thus advise most users to use these default settings. However, it is also possible to exchange estimators and aggregation methods with other options discussed in Sections 2.3 and 2.4, using the estimator and layer_aggregation parameters. An example of instead using *kNN* with *best layer* is illustrated in Listing 2.

Inspecting the Result. The ranking is compiled into a Results object that may be printed to inspect the ranking of language models and their transferability scores. An example ranking produced for 20 language models on the CoNLL-03 shared task is shown in Figure 2.

The model with the best estimate will be at the top of the displayed list. Practitioners can use this information to select the most promising PLMs and proceed to fine-tune them using another framework such as FLAIR (Akbik et al., 2019) or TRANSFORMERS (Wolf et al., 2020).

2.3 Estimators

Each PLM is scored by an estimator to assess its suitability for a classification task. We extract hidden states and pool them into word- or sentence-level embeddings as described in Appendix B. The device for storing the embeddings can be specified.

```
Rank 1.   'google/electra-large-discriminator':        2.6960
Rank 2.   'microsoft/deberta-v3-base':                 2.6257
Rank 3.   'bert-large-uncased':                        2.6165
Rank 4.   'xlm-roberta-large':                         2.6120
Rank 5.   'microsoft/mpnet-base':                      2.5957
Rank 6.   'distilbert-base-uncased':                   2.5742
Rank 7.   'cardiffnlp/twitter-roberta-base':           2.5723
Rank 8.   'microsoft/mdeberta-v3-base':                2.5594
Rank 9.   'google/electra-base-discriminator':         2.5586
Rank 10.  'roberta-base':                              2.5542
Rank 11.  'typeform/distilroberta-base-v2':            2.5445
Rank 12.  'bert-base-uncased':                         2.5413
Rank 13.  'sentence-transformers/all-mpnet-base-v2':   2.5244
Rank 14.  'xlm-roberta-base':                          2.4445
Rank 15.  'sentence-transformers/all-MiniLM-L12-v2':   2.1480
Rank 16.  'google/electra-small-discriminator':        1.9119
Rank 17.  'SpanBERT/spanbert-base-cased':              1.6703
Rank 18.  'allenai/scibert_scivocab_cased':            1.5767
Rank 19.  'dmis-lab/biobert-base-cased-v1.2':          1.5231
Rank 20.  'emilyalsentzer/Bio_ClinicalBERT':           1.3159
```

Figure 2: A ranking of 20 language models produced by TRANSFORMERRANKER for the CoNLL-03 shared task data. The output is ordered by rank, with the estimated best-suited model at the top of the list. For each model, the H-score is printed in the third column. Using these results, a user may exclude the lower-ranked models to only focus on the top-ranked models for further exploration.

Once stored, embeddings are scored using one of the estimators:

Nearest Neighbors Embedding suitability can be evaluated through a nearest neighbor perspective. We adapt the kNN algorithm (Cover and Hart, 1967) as an estimator, applying it to the entire dataset. A pairwise distance matrix is calculated between all data points, with diagonal values excluded to avoid self-distances during the top-k search. To manage memory, we handle distance computations and top-k search in batches, eliminating the need to store a large distance matrix. We use PyTorch to perform batched distance calculations in parallel on a GPU.

LogME You et al. (2021) assumes a linear relationship between embeddings and labels and proposes an algorithm to estimate the Bayesian evidence. Instead of training a linear layer, which can be prone to overfitting, LogME uses a closed-form solution to calculate the model evidence by marginalizing over the weights. Through evidence maximization, two key parameters, α and β, are introduced, controlling regularization strength and noise. The two parameters are updated using fixed-point iteration to maximize the marginal likelihood. LogME is computed by performing Singular Value Decomposition (SVD) on the feature matrix, followed by iterative maximization of α and β for each class. We use PyTorch to implement this estimator with GPU acceleration for SVD and matrix multiplications in the fixed-point iteration.

H-Score (Bao et al., 2019) assesses the suitability of embeddings by evaluating how effectively they can distinguish class variability. Intuitively, well-suited representations should exhibit low within-class variation and high between-class differences, which is captured by using class means in the covariance calculation. By relying solely on covariance matrices, H-Score measures class separability without the need for training or iterative methods. Ibrahim et al. (2023) addressed practical issues with covariance estimation in high-dimensional settings by proposing a shrinkage-based version that regularizes the covariance matrix and uses pseudo-inversion for improved stability.

2.4 Layer Aggregation Strategies

Transferability estimation methods can be improved by including hidden states from deeper layers in the estimation process. The library supports three options: the *last layer*, which uses the hidden states of the final layer of the language model, a common practice when extracting features from a pre-trained model; the *layer mean*, which averages the hidden states of all layers into a single embedding; and the *best layer*, which scores all layers of the language model separately and selects the one with the highest transferability score. Intuitively, the *best layer* option ranks models by their task-specific layer scores.

The first two methods require a single estimation for a model, while *best layer* requires an estimation for each layer. Identifying the layer with the

Ranking Method	Sentence-level	Word-level	Average
Linear Probing $_{\text{layer mean}}$	0.89 $_{\tau=0.76}$	0.78 $_{\tau=0.66}$	0.84 $_{\tau=0.71}$
kNN $_{\text{best layer}}$	0.68 $_{\tau=0.63}$	0.70 $_{\tau=0.55}$	0.68 $_{\tau=0.55}$
H-score $_{\text{layer mean}}$	**0.91** $_{\tau=\mathbf{0.84}}$	**0.85** $_{\tau=0.64}$	**0.88** $_{\tau=0.74}$
LogME $_{\text{layer mean}}$	0.90 $_{\tau=0.80}$	0.82 $_{\tau=\mathbf{0.70}}$	0.86 $_{\tau=\mathbf{0.75}}$

Table 1: Pearson's ρ correlation and weighted Kendall's τ between the estimated scores and fine-tuning scores on three sentence and three per word classification tasks. The results are summarized from Garbas et al. (2024) to illustrate the expected ranking performance.

highest transferability score is useful not only in model ranking but also for finding the most suited layer within a model. Task-specialty of layers was studied by Xie et al. (2022) where a similar metric using the within- and between class variability was used. It was shown that the proposed metric correlates well to training a linear layer and helps finding well-suited layers for classification tasks.

3 Experiments

In this section, we present results of different ranking methods and evaluate how downsampling affects runtime and estimation quality. We further conduct experiments to measure runtime of our GPU-accelerated implementations. Finally, we show a practical example of TRANSFORMERRANKER.

3.1 Accuracy of Transferability Estimation

Table 1 summarizes the findings of Garbas et al. (2024), where the rankings of various estimators and layer aggregation methods were compared against a gold ranking of models obtained by full fine-tuning and hyperparameter selection. To evaluate the rankings, we used Pearson's ρ and weighted Kendall's τ correlations. We present averaged scores over 6 tasks from two task categories: word- and sentence-level classification.

H-score outperformed LogME in Pearson's ρ with scores of 0.88 and 0.86, respectively. In terms of weighted Kendall's τ, H-score and LogME shows similar performance, with average scores of 0.75 and 0.74, where τ for LogME was higher on word-level tasks. For both metrics, the highest scores were achieved using the *layer mean* aggregation. Both H-score and LogME surpass the performance of linear probing and kNN approaches.

Accordingly, we set H-score with *layer mean* as the default parameters in TRANSFORMERRANKER. Other transferability metrics may benefit from different layer aggregation strategies. For example, kNN achieves its best performance with the *best layer* strategy (see Table 1). For this reason, we give users the option selecting both the estimator and the layer aggregator.

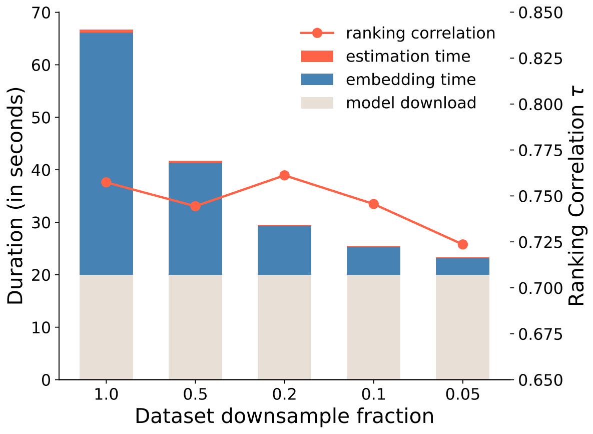

Figure 3: Time taken to estimate a single model including the model download time. We report runtimes for different downsample splits of the CoNLL-03 dataset and show how the ranking correlation changes with different dataset splits. We used the default parameters of *layer mean* with the H-Score estimator. Estimation was done using a batch size of 64 on a single Nvidia A100 (80GB) GPU.

3.2 Effect of Dataset Downsampling

Figure 3 shows the time required to estimate the transferability of a large PLM (DeBERTa-large, He et al., 2020) using a dataset with 22K sentences. Estimation on the full dataset takes about one minute. We devide the runtime into three parts: model download, dataset embedding, and estimation. The model download only occurs on the first run if it is not already stored locally; most of the time is spent on embedding the samples.

Since downsampling can substantially speed up the embedding, we conduct an experiment in which we downsample the CoNLL-03 dataset to 50%, 20%, 10%, and 5% respectively. As Figure 3 illustrates, the dataset can be sampled down substantially without a reduction in ranking correlation.

Device	Estimator kNN	H-score	LogME
CPU	17.76s	1.44s	3.82s
GPU	3.43s	0.20s	0.77s

Table 2: Runtime comparison of transferability estimators using CPU and GPU for 100 000 embeddings of hidden size 1024. We use Intel Xeon Gold 6134 @ 3.20GHz CPU and NVIDIA A100 GPU.

```
Rank  1. 'mdeberta-v3-base':    0.9740
Rank  2. 'twhin-bert-base':     0.9426
Rank  3. 'deberta-v3-base':     0.9112
Rank  4. 'xlm-roberta-base':    0.8884
Rank  5. 'bert-base-german':    0.8633
Rank  6. 'electra-base':        0.8064
Rank  7. 'roberta-base':        0.7970
Rank  8. 'bert-base-cased':     0.7943
Rank  9. 'PharmBERT-cased':     0.7679
Rank 10. 'biobert-base-cased':  0.7657
Rank 11. 'distilbert-base':     0.7504
Rank 12. 'distilroberta-base':  0.7482
Rank 13. 'all-mpnet-base-v2':   0.7427
Rank 14. 'spanbert-base':       0.7348
Rank 15. 'scideberta':          0.7173
Rank 16. 'all-MiniLM-L12-v2':   0.5025
Rank 17. 'electra-small':       0.3330
```

Figure 4: GERMEVAL18 ranking result with H-scores for models from the predefined list of smaller PLMs.

Model	Dev	Test
`mdeberta-v3-base`	**78.41** ± 0.97	**75.86** ± 0.54
`xlm-roberta-base`	77.56 ± 1.20	74.34 ± 0.13
`bert-base-german`	77.31 ± 0.43	73.55 ± 0.19

Table 3: Fine-tuned scores on the GermEval18 dataset for fine-grained classification of offensive language in German. We compare the two most popular models for text classification in German to models that were ranked highest by TRANSFORMERRANKER.

3.3 Runtime of Estimators

We evaluate our implementation of kNN, H-score, and LogME through runtime experiments in which we use 100,000 embeddings with a hidden size of 1024. When reporting runtimes, we exclude the embedding time and only measure the time needed to compute the metrics. Unlike the previous implementations, the estimators in TRANSFORMERRANKER can employ GPU acceleration. We thus measure the runtime on both devices.

Table 2 shows that H-score is faster to compute than LogME, which is consistent with the findings in the original work (Ibrahim et al., 2023). Further, we find that our implementation of kNN is by far the slowest of the three estimators. We also note significant speed-ups of GPU acceleration, amounting to about an order of magnitude faster computation on GPU.

3.4 Demonstration

To give a practical example of the TRANSFORMERRANKER, we use the GERMEVAL18 task of fine-grained classification of offensive language in German (Wiegand et al., 2018). Our baseline is the default approach a practitioner might take: Simply using the most popular PLMs for German on HuggingFace. In this case, our baselines are *(1)* `bert-base-german-cased`, the most downloaded German model, and *(2)* `xlm-roberta-base`, the most downloaded multilingual model.

We use TRANSFORMERRANKER to find one additional promising model from our predefined list of base models[2]. For faster ranking, we downsample GERMEVAL18 to 50%. The produced ranking is presented in Figure 4: Here, TRANSFORMERRANKER estimates `mdeberta-base` to be the best suited model for this task.

We then fine-tune the highest-ranking model and compare it to two baseline models. As Table 3 shows, we find that `mdeberta-v3-base` outperforms the baseline models significantly. This example illustrates the potential value of using automated transferability estimation to identify well-suited PLMs for downstream tasks.

4 Conclusion and Outlook

In this paper, we introduced TRANSFORMERRANKER, a library that aids researchers and practitioners in systematically selecting well-suited PLMs for their downstream NLP tasks. In a fraction of the time, the underlying approach yields a ranking of candidate models that strongly correlates with models ranked by their true downstream performance.

Future work will extend the scope of the addressed downstream tasks and incorporate new methods for transferability estimation. For instance, we are currently adding experimental support for regression tasks (limited to LogME and kNN estimators). We invite the community to experiment with TRANSFORMERRANKER by ranking different models on preferred datasets.

[2]We only select the highest-ranking model though in practice one might consider the top 3 highest-ranking models.

Limitations

TransformerRanker needs a fixed list of PLM names that is provided by the user or predefined in the list of popular models. This limits the exploration of the entire model hub, including lesser known models. To expand this list, the library could automatically suggest models based on their metadata, offering more model options without needing to access their weights.

To compute embeddings, each model must be downloaded if not already present in the cache. This can induce network traffic when exploring a large list of models. Processing embeddings directly within the model hub, or preloading mostly used models onto a dedicated system, could reduce the network traffic for each user.

Additionally, existing transferability metrics are limited to supervised tasks and are not suitable to rank LLMs for text generation.

Ethics Statement

Anticipating ethical concerns for this research is challenging because of the wide-ranging utility of language model selection. As fine-tuning the entire space of available language models is unsustainable and unethical in terms of climate sustainability, efficient encoder pre-selection using transferability estimation provide a positive step toward tackling this problem. TransformerRanker enhances the process of efficient model selection by providing a unified tool, thereby encouraging practitioners to avoid unnecessary fine-tuning.

Acknowledgements

Max Ploner and Alan Akbik are supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2002/1 "Science of Intelligence" – project number 390523135. Alan Akbik is further supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the Emmy Noether grant "Eidetic Representations of Natural Language" (project number 448414230).

References

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59.

Yajie Bao, Yang Li, Shao-Lun Huang, Lin Zhang, Lizhong Zheng, Amir Zamir, and Leonidas Guibas. 2019. An Information-Theoretic Approach to Transferability in Task Transfer Learning. In *2019 IEEE International Conference on Image Processing (ICIP)*.

Elisa Bassignana, Max Müller-Eberstein, Mike Zhang, and Barbara Plank. 2022. Evidence > intuition: Transferability estimation for encoder selection. *Preprint*, arXiv:2210.11255.

Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Lukas Garbas, Max Ploner, and Alan Akbik. 2024. Choose your transformer: Improved transferability estimation of transformer models on classification tasks. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12752–12768, Bangkok, Thailand. Association for Computational Linguistics.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced BERT with disentangled attention. *CoRR*, abs/2006.03654.

Shibal Ibrahim, Natalia Ponomareva, and Rahul Mazumder. 2023. Newer is Not Always Better: Rethinking Transferability Metrics, Their Peculiarities, Stability and Performance. In *Machine Learning and Knowledge Discovery in Databases*, pages 693–709. Springer International Publishing.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2011. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *CoRR*, cs.CL/0306050.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Ellen M Voorhees and Donna Harman. 2000. Overview of the sixth text retrieval conference (trec-6). *Information Processing & Management*, 36(1):3–35.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Michael Wiegand, Melanie Siegel, and Josef Ruppenhofer. 2018. Overview of the germeval 2018 shared task on the identification of offensive language. In *14th Conference on Natural Language Processing KONVENS 2018*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Shuo Xie, Jiahao Qiu, Ankita Pasad, Li Du, Qing Qu, and Hongyuan Mei. 2022. Hidden state variability of pretrained language models can guide computation reduction for transfer learning. *Preprint*, arXiv:2210.10041.

Kaichao You, Yong Liu, Mingsheng Long, and Jianmin Wang. 2021. Logme: Practical assessment of pre-trained models for transfer learning. *CoRR*, abs/2102.11005.

A Dataset Preprocessing

To ready datasets for TransformerRanker, our preprocessing class heuristically finds text and label columns when not user-specified. We leave an option to set text and label columns manually. For text pair tasks, a second text column can be set, and the columns are merged using a separator token. The task category—text classification, regression, or token classification—is determined by the data type of the label column. Additionally, this class downsamples datasets, removes empty texts, and creates label maps for classification.

B Embedding and Pooling

The *Embedder* class in our library computes embeddings from PLMs and does word and text pooling that are needed for different task.

For tasks that require word-level embeddings, such as NER or PoS tagging, the library supports pooling options to extract word-level embeddings since a single word may consist of multiple subword tokens. The *First* option uses the embedding from the first subword of each word. The default *mean* method averages all subwords.

For text classification tasks, TransformerRanker pools word embeddings to create a sentence embedding. It supports *first*, using the CLS-token for models pretrained with this token, and *mean*, the default, which averages all word embeddings. We set *mean* as the default option since we aim to rank models that vary in pre-training tasks. The *mean*, rather than the `CLS`-token, has been empirically shown to be more effective by Bassignana et al. (2022). Using the *last* word's embedding may be suitable for comparing causal models; however, they were not used in the experiments.

C Example using GermEval18

To demonstrate the library in action, we used the dataset of offensive language identification in German tweets, available on the Hugging Face hub as '*philschmid/germeval18*' using the `datasets` framework. To evaluate the TransformerRanker's result, all models were fine-tuned with FLAIR using default parameters: a 5e-6 learning rate, batch size of 4, 10 epochs, along with the AdamW optimizer and a linear scheduler with a 0.1 warmup fraction. Table 3 in the Experiments shows the average F1-micro score over five runs and the standard error.