Performance-Efficiency Trade-Offs
in Adapting Language Models to Text Classification Tasks

Laura Aina, Nikos Voskarides, Roi Blanco
Amazon, Barcelona, Spain
{eailaura,nvvoskar,roiblan}@amazon.com

Abstract

Pre-trained language models (LMs) obtain state-of-the-art performance when adapted to text classification tasks. However, when using such models in real-world applications, efficiency considerations are paramount. In this paper, we study how different training procedures that adapt LMs to text classification perform, as we vary model and train set size. More specifically, we compare standard fine-tuning, prompting, and knowledge distillation (KD) when the teacher was trained with either fine-tuning or prompting. Our findings suggest that even though fine-tuning and prompting work well to train large LMs on large train sets, there are more efficient alternatives that can reduce compute or data cost. Interestingly, we find that prompting combined with KD can reduce compute and data cost at the same time.

1 Introduction

State-of-the-art techniques in NLP, such as adapting pre-trained language models (LMs) to downstream tasks, typically rely on large model and/or train set sizes (Radford et al., 2018; Devlin et al., 2019). In real-world applications, serving large models or having large train sets may be prohibitive due to budget constraints, too slow inference, or scarcity of expensive-to-obtain labeled data. Thus, solutions to build models that save time, money, and energy are preferable (Strubell et al., 2019).

A powerful technique to reduce model size is knowledge distillation (KD) (Hinton et al., 2015). KD requires the availability of a large unlabeled dataset –the *transfer set*– which is usually easy to gather for real-world applications since examples of task inputs are abundant. In KD, the transfer set is weakly labeled with the predictions of a *teacher* model; then a smaller *student* model is trained to match these soft labels. The goal is to retain the teacher's abilities in a more compact architecture.

In this work, we study how performance varies across different scenarios (i.e., model or train set size) when adapting pretrained LMs to downstream tasks. We focus on text classification as a prominent downstream task. We consider the following training procedures to adapt LMs for classification:

- *finetuning (F)*: A classification layer is added to the LM architecture, and the parameters of that layer are jointly learnt with the rest of the LM parameters (Devlin et al., 2019);

- *prompting (P)*: The LM is trained rephrasing the downstream task as word prediction following the objective the LM was trained on (Liu et al., 2021). This technique – also known as *prompt-based fine-tuning*– was found to outperform standard fine-tuning when using small train sets (Schick and Schütze, 2021d);

- *finetuning + KD (F+KD)*: a LM is trained using fine-tuning (F) and then KD is performed to obtain a smaller model (Turc et al., 2019);

- *prompting + KD (P+KD)*: a LM is trained using prompting (P) and then KD is performed to obtain a smaller model. We show that this training procedure can yield classifiers that are both compact and sample-efficient.[1]

We study how the aforementioned training procedures perform on multiple text classification datasets, as we vary model and data cost. Model size is used as an estimate of model cost (Dehghani et al., 2021). While recent studies have focused on training cost (Strubell et al., 2019; Kaplan et al., 2020; Izsak et al., 2021; Yao et al., 2022; Hoffmann et al., 2022), we instead focus on inference

---

[1]In analogous spirit to *P+KD*, previous works used LM-generated data to train sentence embeddings (Schick and Schütze, 2021b), or to obtain compact commonsense models (West et al., 2022). Concurrent to our work, Meng et al. (2022) propose to use data generated by large unidirectional models to transfer their zero-shot NLU abilities to more compact bidirectional models.

Model	# Parameters	Relative speedup
BERT-large	336.2M	-
BERT-base	110.1M	2.6
BERT-medium	41.7M	5.1
BERT-small	29.1M	6.3
BERT-mini	11.3M	7.7

Table 1: Details of the BERT models used in our experiments. Relative speedup is measured with respect to BERT-large, based on average inference time of our final models (across training procedure, tasks and configurations) on a single 16GB GPU with batch size 32, except for BERT-large where we use a batch size of 8.

cost as the most concerning aspect of compute cost in real-world applications: when serving millions of users, inference happens very frequently and on large-scale. On the other hand, train set size is used as a proxy for data cost, assuming the cost of annotating one example is a constant. This allows us to discuss sample efficiency, that is, the amount of data required to achieve acceptable performance.

Our contributions are two-fold. First, we show that *P+KD* allows to reduce both model and train set size while retaining high performance. Second, we extrapolate recommendations on how to efficiently adapt LMs for downstream tasks. The trends we identify indicate that the cost of increasing model parameters or train set size is not always worth it: small models or models trained with little data often achieve comparable performance than larger models trained with more data.

2 Experimental setup

2.1 Models

We experiment with 5 BERT LMs of increasing size released by Turc et al. (2019): BERT-mini, -small, -medium, -base, -large. These models have been trained on the same English text corpus, share the BERT architecture but differ in hidden size, number of hidden layers and attention heads. Therefore, they differ in *inference speed*: for instance, BERT-mini is 7x faster than BERT-large. Please refer to Table 1 for more details on the compared models and their relative inference speed.

2.2 KD

We use *pre-trained distillation* (Turc et al., 2019), where both the teacher and the student are pre-trained LMs that are adapted to the downstream task. This was shown to work better than training

both models from scratch directly on the downstream task (Turc et al., 2019). When doing *KD*, we use as teacher model the best on dev data among those for that train set size. For instance, for *F+KD* on BERT-small with train set size 20, we use as teacher the BERT-large model trained with fine-tuning on 20 training examples. BERT-large is always used as the teacher model in KD; therefore it is never used as student model in *F+KD* or *P+KD*.

2.3 Prompting

We adapt a LM to the downstream task by tuning its weights to output the correct predictions on the train set, as in Pattern-Exploiting Training (PET) (Schick and Schütze, 2021a,c). In its original formulation, PET trains multiple LM instances with different templates[2] and then uses their predictions to obtain a single classifier from the original LM. For simplicity in experimentation and without substantial loss in accuracy, we instead follow Le Scao and Rush (2021): we tune the LM with a single template and use this as our final classifier. Note that the original PET algorithm also involves KD, but differently from our setting, it is used to obtain a model of the *same size* of the starting LM from the ensemble of LMs trained on different templates. In contrast, we distill *a large LM to a smaller LM*.

2.4 Datasets

We use 4 English text classification datasets (Zhang et al., 2015): Yelp-full (sentiment; 5 classes); Yelp-polarity (sentiment; 2 classes), Yahoo-questions (question; 10 classes); AG news (news article; 4 classes). The test size of each dataset is 50K, 38K, 60K, 7.6K, respectively. These datasets are large enough to allow us to both flexibly explore the effect of train set size and also build a large transfer set of unlabeled examples to be used by KD. We sample 10K examples from the original train set of each dataset as the transfer set (discarding the gold labels of those examples). In preliminary experiments we varied the size of the transfer set from 5K to 10K without observing substantial differences in the trends; larger transfer sets can be explored in future work.

The aforementioned datasets were employed in

---

[2]A template is the way the task is set up. For instance, one template is to append "All in all, it was..." to the end of a review and map output adjectives to sentiment labels (e.g., "terrible" → 1; "great" → 5).

the experiments of Schick and Schütze (2021a); we build on their experimental setup and prompting templates.[3] For each task, we consider 11 exponentially growing train set sizes from 20 to 20480, sampled from the original train set. We exclude from each train set a 10% portion to be used for sampling dev sets, with a minimum of 20 examples. Studies exploring train set size in analogous spirit to ours used either no dev set (Schick and Schütze, 2021a,d) or one that is kept constant across train set size (Le Scao and Rush, 2021). We strike a balance between these approaches by using a dev set of proportional size to the train set, which is a realistic assumption. Even if very small, having a dev set is useful as it gives an indication of quality during model development. In all train and dev sets we balance the number of examples for each class.

2.5 Hyperparameter Search

We focus on batch size and learning rate, and on the task-specific prompting template (Schick and Schütze, 2021a). Keeping these constant across experiments could be unfair, as different combinations of train sets sizes, models, training procedures and tasks may favor different hyperparameters. On the other hand, to run a search for each combination would be extremely costly and time-consuming. We go for an intermediate strategy; for instance, for BERT-mini trained with 80 examples, we use the hyperparameters selected for BERT-small with 20 examples. We describe our choices in more detail in Appendix A.

2.6 Other Details

For every task and configuration (combination of LM and train set size; 5×11), we run training 4 times and report on the model achieving the highest accuracy on the dev set (Le Scao and Rush, 2021). As there tends to be variation across runs (Dodge et al., 2020; Schick and Schütze, 2021d), we focus on the highest achieved accuracy, as opposed to the mean, in order to compare each configuration in its best case scenario. Note that we observe similar trends when considering mean and standard deviation across runs (see Fig. 3 in the Appendix).

(a) Yahoo-Q   (b) Yahoo-Q

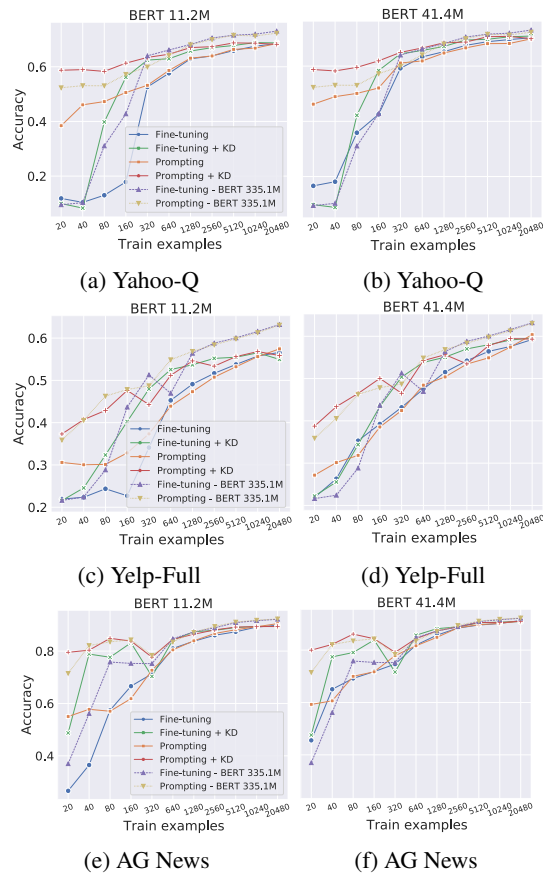(c) Yelp-Full   (d) Yelp-Full

(e) AG News   (f) AG News

Figure 1: Classification accuracy (y-axis) across train set sizes (x-axis) and training procedures (lines) on the BERT 11.2M (mini) and the BERT-41M (medium). To facilitate the comparison we also show the performance of BERT 335M (large), used as teacher in KD.

3 Results and Discussion

3.1 Effect of Training Procedure

Fig. 1 shows classification accuracy on Yahoo Questions (Yahoo-Q), Yelp-Full and AG News across training procedures, models and train set sizes.[4] We identify consistent trends across datasets that can be summarized as follows:

- Prompting (both *P* and *P+KD*) outperforms fine-tuning on small train sets (< 320).

- Unless the train set is large (> 2560), KD (both *F+KD*, *P+KD*) allows to increase the quality of compact models.

- *P+KD* leads to effective small models using small train sets (see red lines in Fig. 1).

---

[3]The code to run our experiments was developed upon Schick and Schütze's code: `https://github.com/timoschick/pet`

[4]Please refer to Fig. 2 in the Appendix for the complete set of plots for of all model and dataset combinations we considered; the trends are consistent.

- There is gradually less difference across training procedures as the train set gets larger.

Fine-tuning. In line with Le Scao and Rush (2021); Schick and Schütze (2021a), we find that fine-tuning performs worse than prompting in few-shot learning (< 320 examples). There are large accuracy gaps between the smallest and largest train sets (20 and 20K), sometimes starting from random-level performance. On very small train sets, using more model parameters is not always beneficial; for instance, BERT-large (335.1M) is worse than BERT 41.4M on Yelp-full for <160 examples.[5] As a result, KD (which uses BERT-large as teacher) does not always improve over fine-tuning. However, with more training data, bigger models perform better and KD allows smaller models to fill the gap with the teacher.

Prompting. Prompting near-always improves the performance over fine-tuning with small train sets (< 320), with or without KD. Interestingly, smaller LMs trained with prompting are almost always improved with KD (*P+KD*). This can lead to having compact models with high accuracy in the low data regime: for instance, BERT with 11M parameters performs comparably with BERT 335.1M for small train sets (<320, Fig. 1), even though it has 30x fewer parameters and 7x faster inference (Turc et al., 2019). As the train set becomes larger, accuracy increases, and there is less difference between fine-tuning and prompting.

KD. Combined with either fine-tuning or prompting, KD allows to build effective small models, including with limited train set size. Larger student models tend to fill more the gap with the teacher model, presumably due to their wider capacity (see BERT 11M vs BERT 40M). For both fine-tuning and prompting, KD does not bring improvements when the teacher is exposed to more than 10240 training examples. One explanation is that KD is beneficial only if the transfer set is much larger than the train set the teacher was exposed to, whereas we used 10K unlabeled examples for all train set sizes (§2). Moreover, the gap between the largest

---

[5]An explanation is that smaller models, because of the fewer number of parameters to update, may be less sensitive to model initialization. Another factor that can explain some of the fluctuations in accuracy is that occasionally when the dev set is very small, the best model on it is not necessarily the best on test data. This should not be taken as a limitation of our experiments but rather as evidence of issues which could realistically occur in scenarios with limited data.

model size		
train	max. acc. diff. = 0.01	max. acc. diff = 0.05
20480	[335M] *P*, *F*	[11M] *P*, *F*, *F+KD*
10240	[109M] *P+KD*, *F+KD*	[11M] *P+KD*, *F+KD*
5120	[109M] *P+KD*, *F+KD*	[11M] *P+KD*, *F+KD*
2560	[109M] *P+KD*, *F+KD*	[11M] *P+KD*, *F+KD*
1280	[109M] *P+KD*, *F+KD*	[11M] *P+KD*, *F+KD*
640	[109M] *P+KD*, *F+KD*	[11M] *P+KD*, *F+KD*
320	[109M] *F+KD*	[11M] *F+KD*
160	[109M] *P+KD*	[11M] *P+KD*
80	[109M] *P+KD*	[11M] *P+KD*
40	[109M] *P+KD*	[11M] *P+KD*
20	[28M] *P+KD*	[11M] *P+KD*
train set size		
params	max. acc. diff. = 0.01	max. acc. diff = 0.05
335M	[10240] *P*	[2560] *P*, *F*
109M	[10240] *P+KD*, *F+KD*	[1280] *P+KD*, *F+KD*
41M	[5120] *P+KD*	[1280] *P+KD*, *F+KD*
28M	[10240] *P+KD*, *F+KD*	[1280] *P+KD*, *F+KD*
11M	[10240] *P+KD*, *F+KD*	[1280] *P+KD*, *F+KD*

Table 2: Combination of training procedure and smallest train set/model size leading to optimal accuracy (has at most 0.01 or 0.05 accuracy difference from the highest achieved for that train set or model size), for each model/train set size, in at least 3 of the 4 tasks considered. *F*: fine-tuning, *P*: prompting, *F+KD*: fine-tuning followed by KD, *P+KD*: prompting followed by KD.

model and the smaller ones always decreases with more training examples: with a large train set, there is, to begin with, less to gain by learning from a larger model.

3.2 Recommendations for Adapting LMs

Here, we explore which combinations of training procedure, train set and model size lead to high performance with low data and compute cost. Note that in real-world applications, trading some performance (e.g., accuracy) for efficiency may be acceptable. This is because small differences in offline metrics such as accuracy may not strongly affect the actual efficacy of models in online metrics (Yi et al., 2013). Also, savings that efficient models enable may counterbalance the impact of a slightly worse performance.

We structure the search for the most favorable combinations as follows. For each model size, we seek for the combination of training procedure and smallest train set size that leads to optimal accuracy. For each train set size, we seek for the combination of training procedure and *smallest model that leads to optimal accuracy*. A combination with optimal accuracy is one that has at most 0.01 or 0.05 accuracy difference from the highest achieved for that

train set or model size.

Table 2 shows the results. We observe that, independently of train set size, we can reduce model size: with a potential accuracy loss of max 0.05, KD allows us to use the smallest LM considered (11M parameters). If the train set is small, KD should be combined with prompting. Train set size can be reduced less safely than model size, though still considerably: across model sizes, medium-to-large train sets seem to work best, but we do not necessarily need the maximum size considered (20480); we can reduce this further (1280) with some accuracy loss.

4 Conclusion and Future Work

We investigated performance-efficiency trade-offs of different training procedures for adapting LMs to text classification tasks. We considered efficiency both as the compute cost associated with inference and the data cost of labeling training examples. We found that both prompting and fine-tuning work well to train large LMs on large train sets, but there exist more efficient alternatives to build effective models. To reduce compute cost, one can prompt or fine-tune compact LMs; if the train set is small, provided the availability of a transfer set, KD from a large model should be applied to obtain a small model. To reduce data cost, prompting is recommended, better if combined with KD for smaller models. To simultaneously reduce both compute and data cost, *P+KD* is the most efficient training procedure.

Our results can help NLP practitioners to identify the best strategy to follow on adapting pretrained LMs to text classification tasks based on compute budget and the cost of data collection. Note that in this work we focused on the compute of cost of inference rather than that of training due to its larger impact on real-world applications where models are served frequently and on a large scale. However, there are differences in training cost among the procedures we investigated (e.g., KD requires training both a teacher and a student model), which will have to be taken into account if facing a limit in train cost budget and resources.

For future work, we want to expand our study to other NLP tasks such as sequence tagging, as well as languages other than English. Also, in our experiments we used a relatively small teacher model to facilitate experimentation; we expect the performance of *P+KD* to increase if using larger

LMs with in-context learning (Brown et al., 2020; Alex et al., 2021), as opposed to prompt-based fine-tuning.

Acknowledgements

References

Neel Alex, Eli Lifland, Lewis Tunstall, Abhishek Thakur, Pegah Maham, C Jess Riedel, Emmie Hine, Carolyn Ashurst, Paul Sedille, Alexis Carlier, et al. 2021. RAFT: A real-world few-shot text classification benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. 2021. The efficiency misnomer. *arXiv preprint arXiv:2110.12894*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Peter Izsak, Moshe Berchansky, and Omer Levy. 2021. How to train BERT with an academic budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10644–10652.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Teven Le Scao and Alexander M Rush. 2021. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2627–2636.

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 preprint arXiv:2107.13586*.

Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating training data with language models: Towards zero-shot language understanding. In *Advances in Neural Information Processing Systems*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269.

Timo Schick and Hinrich Schütze. 2021b. Generating datasets with pretrained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6943–6951, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021c. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352.

Timo Schick and Hinrich Schütze. 2021d. True few-shot learning with prompts–a real-world perspective. *arXiv preprint arXiv:2111.13440*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*.

Peter West, Chandra Bhagavatula, Jack Hessel, Jena Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. 2022. Symbolic knowledge distillation: from general language models to commonsense models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4602–4625, Seattle, United States. Association for Computational Linguistics.

Xingcheng Yao, Yanan Zheng, Xiaocong Yang, and Zhilin Yang. 2022. NLP from scratch without large-scale pretraining: A simple and efficient framework. In *International Conference on Machine Learning*, pages 25438–25451. PMLR.

Jeonghee Yi, Ye Chen, Jie Li, Swaraj Sett, and Tak W Yan. 2013. Predictive model performance: Offline and online evaluations. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1294–1302.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

Appendix

A Training & Model development

Hyperparameter Search. We make a set of simplifications to the hyperparameter search process to considerably speed up the process (i.e., not having to run a search for each combination of task, training procedure, model and train set size) while still aiming to a later fair comparison in our experiments.

1. For prompting/fine-tuning batch size and learning rate, we run the search only for one task and dataset (Yelp-full), generalizing the optimal configurations to others. For the prompting template, we run the search for each task (except for Yelp-polarity, where we can use the identical templates as Yelp-full). In the template search on tasks other than Yelp-full, we set batch size and learning rate based on the Yelp-full search.

2. We consider only a few combinations of train set and model sizes, generalizing the optimal configurations to analogous setups, based on some ranges. In particular: 20, 320, 2560 examples; BERT-large, BERT-base, BERT-small. Configurations picked for 20 examples are generalized to any value between 20 and

320, etc. BERT-small configurations are generalized to the other two models with close size (BERT-mini, BERT-medium). These choices are based on the assumption that models of close size or trained on train sets of close size should work well with the same hyperparameters configurations.

3. We only run the search for: 1) fine-tuning, and 2) prompting - both without KD. We use the optimal configurations from fine-tuning any time we need to train a classifier from a certain LM, including when training it as a student model with KD.

To run the search, we run the training 4 times, and consider the maximum average dev accuracy (with mimimum standard deviation in case of ties) to establish the optimal configuration. We consider the following values:

1. Learning rate: 1e-05, 2e-05, 5e-05;

2. Batch size: 8, 16, 32; for BERT-large, for memory reason we set the batch size to 8 but effectively obtain batch sizes > 8 by modulating the number of steps to accumulate gradients.

3. Prompting template: We use the task-specific templates from Schick and Schütze (2021a); 4 for Yelp-full and Yelp-polarity; 6 for other tasks (see Section B)

We run grid search for fine-tuning and bayesian search (maximum 18 models) for prompting, due to the bigger set combinations to try for the latter.

In Table 3 we report the selected hyperparameters based on the search, jointly with the standard deviation in average dev performance across configurations. The selected hyperparameters for each combination of train set and model size tend to vary, and their choice can be impactful on the achieved accuracy.

Early Stopping. For all models we use as criterion for early stopping performances on dev data: we stop training when dev accuracy does not grow after 3 epochs (1 epoch when using more than 20K examples).

For all tasks, we use a maximum sequence length of 256 tokens.

B Prompting templates

B.1 Yelp-Full & Yelp-Polarity

Input:
0) [review] + *It was [MASK] .*
1) [review] + *. All in all, it was [MASK] .*
2) [review] + *Just [MASK]!*
3) [review] + *In summary, the restaurant is [MASK] .*
Output:
Yelp-Full: *terrible, bad, okay, good, great* → 1-5
Yelp-Polarity: *bad, good* → negative, positive.

B.2 Yahoo Questions

Input: (question, answer)
0) *[MASK] :* + [question] + [answer]
1) *[MASK] Question:* + [question] + [answer]
2) [question] + *([MASK])* + [answer]
3) [question] + [answer] + *([MASK])*
4) *[Question: [MASK]]* + [question] + [answer]
5) *[MASK] -* + [question] + [answer]
Output:
Society, Science, Health, Education, Computer, Sports, Business, Entertainment, Relationships, Politics
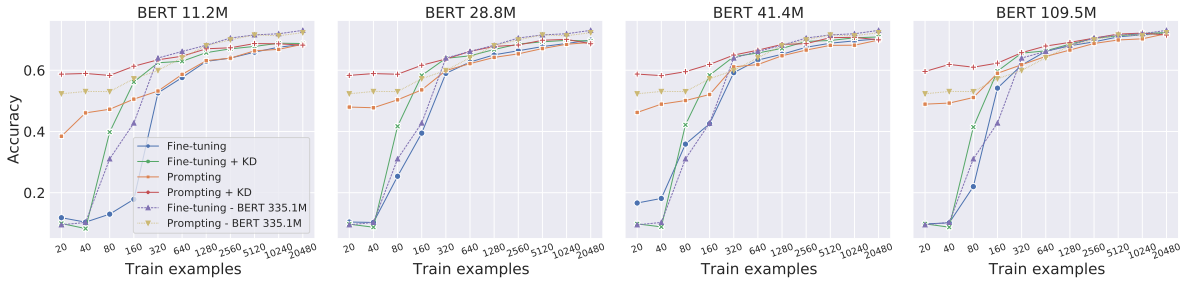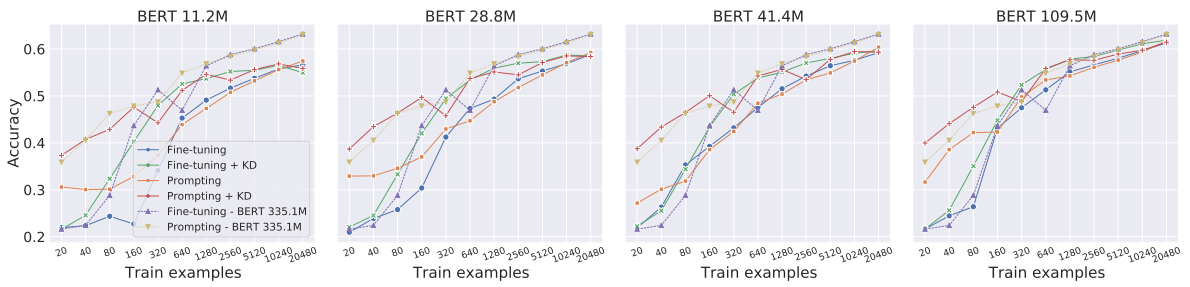
B.3 AG News

Input: (headline, text)
0) *[MASK] :* + [headline] + [text]
1) *[MASK] News:* + [headline] + [text]
2) [headline] + *([MASK])* + [text]
3) [headline] + [text] + *([MASK])*
4) *[News: [MASK]]* + [headline] + [text]
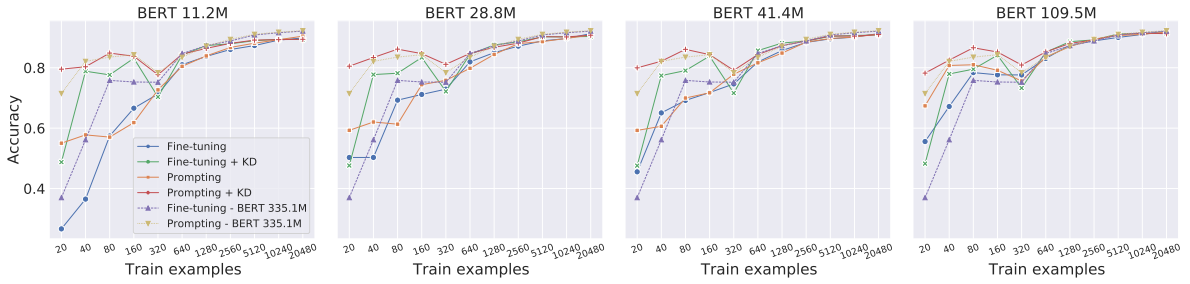5) *[MASK] -* + [headline] + [text]
Output:
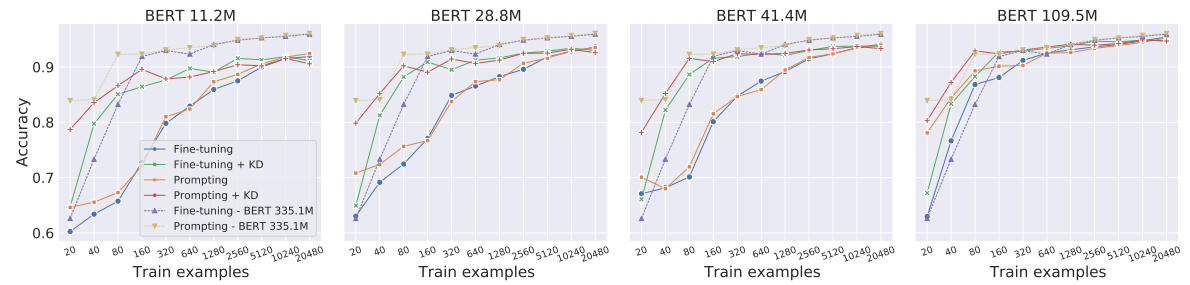*World, Sports, Business, Tech*

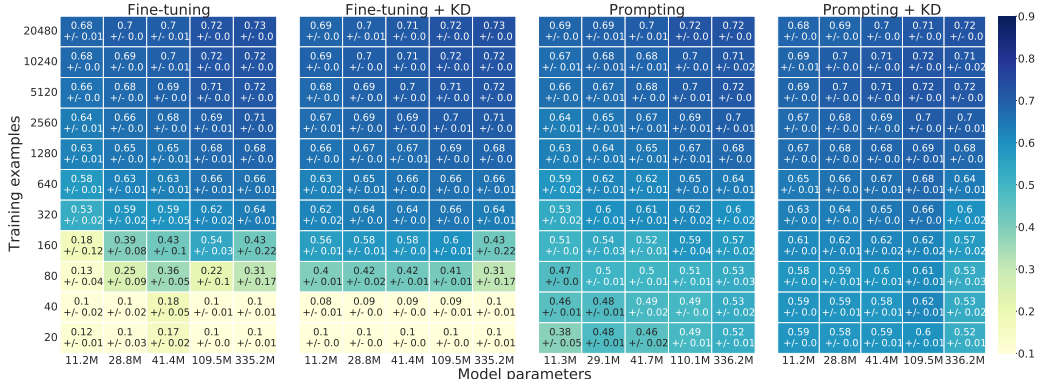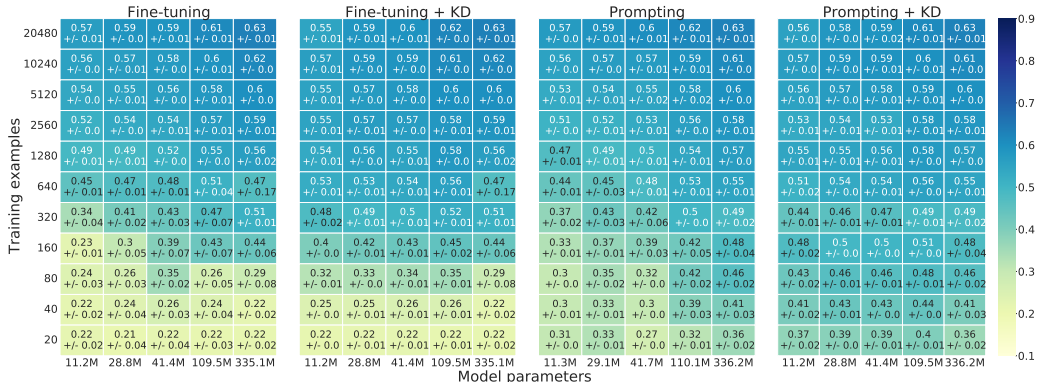(a) Yahoo Questions

(b) Yelp-Full

(c) AG News

(d) Yelp-Polarity

Figure 2: Classification accuracy (y-axis) across train set sizes (x-axis), model sizes (plots), and training procedure (lines), for AG News ad Yelp-Polarity. To facilitate the comparison, for each model, we show the performance of BERT-large (335M parameters), used as teachers for KD, in the same plot.
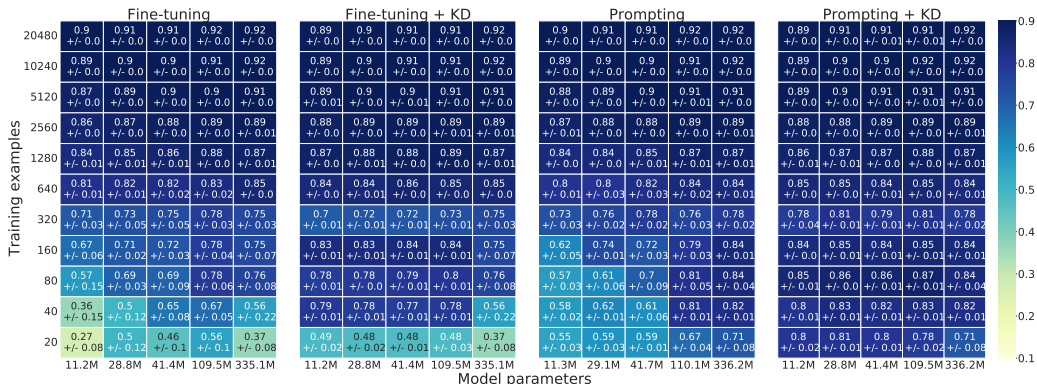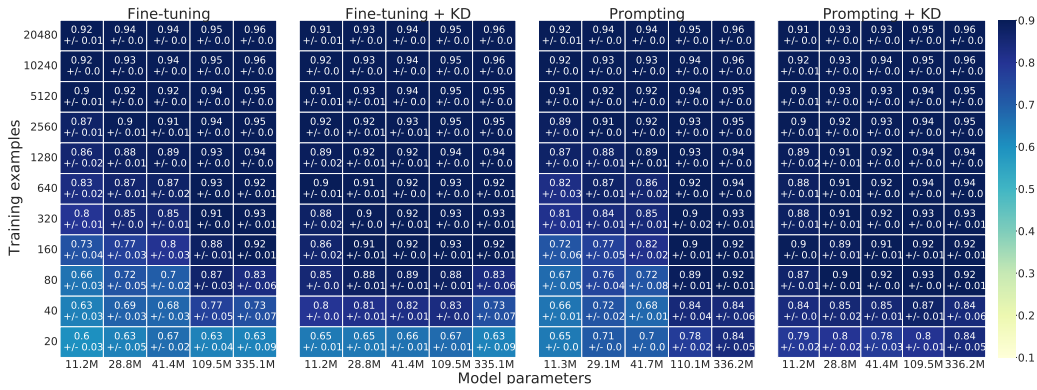
(a) Yahoo Questions

(b) Yelp-Full

(c) AG News

(d) Yelp-Polarity

Figure 3: For each task, heatmap of mean test classification accuracy (with standard deviation) across models with different number of parameters (x-axis) and train set sizes (y-axis) over 4 training runs.

	Fine-tuning		
		train set size	
parameters	20	320	2560
28M	Yelp: 2e-05, 8 (std: 0.04)	Yelp: 5e-05, 8(std: 0.05)	Yelp: 1e-05,32 (std: 0.01)
109.5M	Yelp: 2e-05, 8 (std: 0.03)	Yelp: 2e-05, 16 (std: 0.04)	Yelp: 1e-05,32 (std: 0.02)
335.1M	Yelpl: 2e-05, 8 (std: 0.03)	Yelp: 2e-05, 8 (std: 0.12)	Yelp: 1e-05,32 (std: 0.12)
	Prompting		
		train set size	
parameters	20	320	2560
28M	Yelp: 1e-05, 32, 0 (std: 0.08) Yahoo: 0 (std: 0.06) AG news: 3 (std: 0.07)	Yelp: 5e-05, 32, 0 (std: 0.02) Yahoo: 2 (std: 0.03) AG news: 2 (std: 0.01)	Yelp: 2e-05, 8, 0 (std: 0.01) Yahoo: 4 (std: 0.01) AG news: 1 (std: 0.01)
109.5M	Yelp: 2e-05, 16, 1 (std: 0.07) Yahoo: 2 (std: 0.06) AG news: 0 (std: 0.03)	Yelp: 1e-05, 16, 0 (std: 0.03) Yahoo: 5 (std: 0.04) AG news: 3 (std: 0.02)	Yelp: 2e-05, 16, 3 (std: 0.01) Yahoo: 1 (std: 0.01) AG news: 1 (std: 0.01)
335.1M	Yelp: 1e-05, 16, 3 (std: 0.07) Yahoo: 5 (std: 0.07) AG news: 1 (std: 0.07)	Yelp: 1e-05, 32, 3 (std: 0.03) Yahoo: 5(std: 0.02) AG news: 3 (std: 0.01)	Yelp: 1e-05, 16, 3 (std: 0.01) Yahoo: 0 (std: 0.01) AG news: 0 (std: 0.01)

Table 3: Best hyperparameters based on search per task (Yelp-Full: learning rate, batch size, pattern id; other tasks: pattern id only). std = standard deviation of average dev performance across hyperparameters configurations.