Cost-effective Deployment of BERT Models in Serverless Environment

Marek Šuppa
Slido
msuppa@slido.com

Katarína Benešová
Slido
kbenesova@slido.com

Andrej Švec
Slido
asvec@slido.com

Abstract

In this study we demonstrate the viability of deploying BERT-style models to AWS Lambda in a production environment. Since the freely available pre-trained models are too large to be deployed in this way, we utilize knowledge distillation and fine-tune the models on proprietary datasets for two real-world tasks: sentiment analysis and semantic textual similarity. As a result, we obtain models that are tuned for a specific domain and deployable in the serverless environment. The subsequent performance analysis shows that this solution does not only report latency levels acceptable for production use but that it is also a cost-effective alternative to small-to-medium size deployments of BERT models, all without any infrastructure overhead.

1 Introduction

Machine learning models are notoriously hard to bring to production environments. One of the reasons behind is the large upfront infrastructure investment it usually requires. This is particularly the case with large pre-trained language models, such as BERT (Devlin et al., 2018) or GPT (Radford et al., 2019) whose size requirements make them difficult to deploy even when infrastructure investment is not of concern.

At the same time, the serverless architecture with minimal maintenance requirements, automatic scaling and attractive cost, is becoming more and more popular in the industry. It is very well suited for stateless applications such as model predictions, especially in cases when the prediction load is unevenly distributed. Since the serverless platforms have strict limits, especially on the size of the deployment package, it is not immediately obvious it may be a viable platform for deployment of models based on large pre-trained language models.

In this paper we describe our experience with deploying BERT-based models to AWS Lambda in a production environment. We consider two tasks: sentiment analysis and semantic textual similarity. While the standard approach would be to fine-tune the pre-trained models, this would not be possible in our case, as the resulting models would be too large to fit within the limits imposed by AWS Lambda. Instead, we adopt a knowledge distillation approach in combination with smaller BERT-based models. We show that for some of the tasks we are able to train models that are an order of magnitude smaller while reporting performance similar to that of the larger ones.

Finally, we also evaluate the performance of the deployed models. Our experiments show that their latency can reach levels accepted in production environments. Furthermore, the reported costs suggest it is a very cost-effective option, especially when the expected traffic is small-to-medium in size (a few requests per second) and potentially unevenly distributed.

2 Related work

Despite a number of significant advances in various NLP approaches over the recent years, one of the limiting factors hampering their adoption is the large number of parameters that these models have, which leads to large model size and increased inference time. This has negative implications to their environmental costs (Strubell et al., 2019) and may limit their use in resource-constrained mobile devices or any other environment in which model size and inference time is the limiting factor.

This has led to a significant body of work focusing on lowering both the model size and inference time, while incurring minimal performance penalty. One of the most prominent approaches include Knowledge Distillation (Buciluǎ et al., 2006; Hinton et al., 2015), in which a smaller model (the ”student”) is trained to reproduce the behavior of a larger model (the ”teacher”). It was used to produce smaller BERT alternatives, such as:

- **TinyBERT** (Jiao et al., 2019), which appropriates the knowledge transfer method to the Transformer architecture and applies it in both the pretraining and downstream fine-tuning stage. The resulting model is more than 7x smaller and 9x faster on inference

- **MobileBERT** (Sun et al., 2020), which only uses knowledge distilation in the pre-training stage and reduces the model's width (layer size) as opposed to decreasing the number of layers it consists of. The final task-agnostic model is more than 3x smaller and 5x faster than the original BERT$_{\text{BASE}}$.

When decreasing the model size leads to decreased latency, it can also have direct business impact. This has been demonstrated by Google, which found out that increasing web search latency from 100 ms to 400 ms reduced the number of searches per user by 0.2 % to 0.6 % (Brutlag, 2009). A similar experiment done by Booking.com has shown that an increase in latency of about 30 % results in about 0.5 percentage points decrease in conversion rates, which the authors report as a *"relevant cost for our business"* (Bernardi et al., 2019).

Each serverless platform has its specifics, which can have different impact on different use cases. Various works, such as (Back and Andrikopoulos, 2018; Wang et al., 2018; Lee et al., 2018), provide a comparison of performance differences between the available platforms. In order to evaluate specific use cases, various benchmark suites have been introduced such as FunctionBench (Kim and Lee, 2019), which also includes a language generation as well as sentiment analysis test case.

Possibly the closest published work comparable to ours is (Tu et al., 2018), in which the authors demonstrate the deployment of neural network models, trained for short text classification and similarity tasks in a serverless context. Since at the time of its publication the PyTorch deployment ecosystem has been in its nascent stages, the authors had to build it from source, which complicates practical deployment.

To the best of our knowledge, our work is the first to show the viability of deploying large pretrained language models (such as BERT and its derivatives) in the serverless environment.

Resource	**AWS**	**GCP**	**Azure**
Function size	250MB[1]	500MB	-
Execution time	15min	9min	-
Memory	10GB	4GB	14GB
Request size	6MB	10MB	100MB

Table 1: Limitations of the three main serverless providers: Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure (Azure).

3 Serverless environments

Serverless environments offer a convenient and affordable way of deploying a small piece of code. A survey by O'Reilly Media (O'Reilly Media, Inc, 2019) shows that the adoption of serverless was successful for the majority of the respondents' companies. They recognize reduced operational costs, automatic scaling with demand and elimination of concerns for server maintenance as the main benefits.

Since the functions deployed in a serverless environment share underlying hardware, OS and runtime (Lynn et al., 2017), there are naturally numerous limitations to what can be run in such environment. The most pronounced ones include:

- **Maximum function size**, mostly limited to a few hundreds of MBs (although some providers do not have this limitation). In the context of deployment of a machine learning model, this can significantly limit the model size as well as the selection of libraries to be used to execute the model.

- **Maximum memory** of a few GBs slows down or makes it impossible to run larger models.

- **No acceleration.** Serverless environments do not support GPU or TPU acceleration which can significantly increase the inference time for larger models.

A more detailed list of the main limitations of the three most common serverless providers can be found in Table 1. It suggests that any model deployed in this environment will need to be small in size and have minimal memory requirements. These requirements significantly limit the choice of models appropriate for this environment and warrants a specific training regimen, which we describe in the next section.

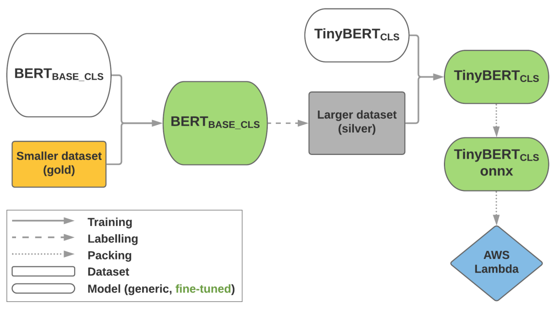


Figure 1: Schema of the distillation pipeline of $\text{BERT}_{\text{BASE}}$ for sentiment analysis. $\text{BERT}_{\text{BASE\_CLS}}$ is fine-tuned on the gold dataset and then used for labelling a large amount of data (silver dataset) which serve as a training set for distillation to TinyBERT. The distilled model is exported to the ONNX format and deployed to AWS Lambda (see Section 5). The same pipeline was executed for MobileBERT.

4 Model training

In the two case studies presented in this section, we consider BERT-provided classification token (`[CLS]` token) as an aggregate representation of a short text (up to 300 characters) for the sentiment analysis task and the embeddings produced by Sentence-BERT (SBERT) (Reimers and Gurevych, 2019) for estimating the semantic similarity of a pair of such short texts.

Since deploying even the smaller $\text{BERT}_{\text{BASE}}$ with over 400MB in size is not possible in our setup, in the following cases studies we explore several alternative approaches, such as knowledge distillation into smaller models or training a smaller model directly. To do so, we use TinyBERT (Jiao et al., 2019) and MobileBERT (Sun et al., 2020) having about 56 MB and 98 MB in size, respectively.

4.1 BERT for sentiment analysis

One of the direct applications of the special `[CLS]` token of BERT is the analysis of sentiment (Li et al., 2019). We formulate this problem as classification into three categories: *Positive*, *Negative* and *Neutral*.

The task is divided into two stages: first, we fine-tune $\text{BERT}_{\text{BASE}}$ for our domain using a labelled dataset of 68K training examples. Then we proceed with knowledge distillation into a smaller model with faster inference: the fine-tuned $\text{BERT}_{\text{BASE}}$ is used for labelling a large amount of unlabelled data which is used to train a smaller model with a BERT-like architecture. The distillation pipeline is illustrated in Figure 1.

4.1.1 Fine-tuning $\text{BERT}_{\text{BASE}}$

To utilize $\text{BERT}_{\text{BASE}}$ for a classification task, an additional head must be added on top of the Transformer blocks, i.e. a linear layer on top of the pooled output. The additional layer typically receives only the representation of the special `[CLS]` token as its input. To obtain the final prediction, the output of this layer is passed through a Softmax layer producing the probability distribution over the predicted classes.

We fine-tuned $\text{BERT}_{\text{BASE}}$ for sequence classification ($\text{BERT}_{\text{BASE\_CLS}}$) with this adjusted architecture for our task using a labelled dataset of size 68K consisting of domain-specific data. We trained the model for 8 epochs using AdamW optimizer with small learning rate 3×10^{-5}, L2 weight decay of 0.01 and batch size 128.

To cope with the significant class imbalance[2] and to speed up the training, we sampled class-balanced batches in an under-sampling fashion, while putting the examples of similar length together (for the sake of a more effective processing of similarly padded data). Using this method, we were able to at least partially avoid over-fitting on the largest class and reduce the training time about 2.5 times.

We also tried an alternative fine-tuning approach by freezing $\text{BERT}_{\text{BASE}}$ layers and attaching a small trainable network on top of it. For the trainable part, we experimented with 1-layer bidirectional GRU of size 128 with dropout of 0.25 plus a linear layer and Softmax output. $\text{BERT}_{\text{BASE\_CLS}}$ outperformed this approach significantly.

The accuracy evaluation of the both fine-tuned $\text{BERT}_{\text{BASE}}$ models on the validation dataset can be found in Table 2. In order to resolve the undesirable size of $\text{BERT}_{\text{BASE\_CLS}}$, we proceed to the knowledge distillation stage.

4.1.2 Knowledge distillation to smaller BERT models

Having access to virtually unlimited supply of unlabelled domain-specific data, we labelled almost 900K examples by the fine-tuned $\text{BERT}_{\text{BASE\_CLS}}$ "teacher" model and used them as ground truth labels for training a smaller "student" model. We

[1]Recently, a new way of deployment was added, allowing to deploy a container of size up to 10 GB.

[2]About 82% of the dataset were *Neutral* examples, 10% *Negative* and 8% *Positive*.

Model	**Size** (MB)	**F1**
BERT$_{\text{BASE}}$ + GRU	426	0.75
BERT$_{\text{BASE\_CLS}}$	420	**0.84**
TinyBERT (distilled)	56	0.82
MobileBERT (distilled)	98	**0.84**

Table 2: Comparison of fine-tuned BERT models and smaller distilled models on the validation dataset (macro-averaged F1 score). The slight decrease in TinyBERT's performance is an acceptable trade-off for the significant size reduction.

experimented with MobileBERT and even smaller TinyBERT as the student models since these are, in comparison to BERT$_{\text{BASE}}$, 3 and 7 times smaller in size, respectively.

During training we sampled the batches in the same way as in Section 4.1.1, except for a smaller batch size of 64. We trained the model for a small number of epochs using AdamW optimizer with learning rate 2×10^{-5}, weight decay 0.01 and early stopping after 3 epochs in case of TinyBERT and one epoch for MobileBERT (in the following epochs the models no longer improved on the validation set).

We managed to successfully distill the knowledge into significantly smaller TinyBERT with only 0.02 points decrease in F1 score (macro-averaged). In case of MobileBERT we were able to match the performance of BERT$_{\text{BASE\_CLS}}$. The performance comparison is summarized in Table 2. These results suggest that the large language models might not be necessary for classification tasks in a real-life scenario.

4.2 Sentence-BERT for semantic textual similarity

The goal of our second case study was to train a model that would generate dense vectors usable for semantic textual similarity (STS) task in our specific domain and be small enough to be deployed in a serverless environment. The generated vectors would then be indexed and queried as part of a duplicate text detection feature of a real-world web application. To facilitate this use-case, we use Sentence-BERT (SBERT) (Reimers and Gurevych, 2019).

While the SBERT architecture currently reports state-of-the-art performance on the sentence similarity task, all publicly available pre-trained SBERT models are too large for serverless deployment. The smallest one available is SDistilBERT$_{\text{BASE}}$

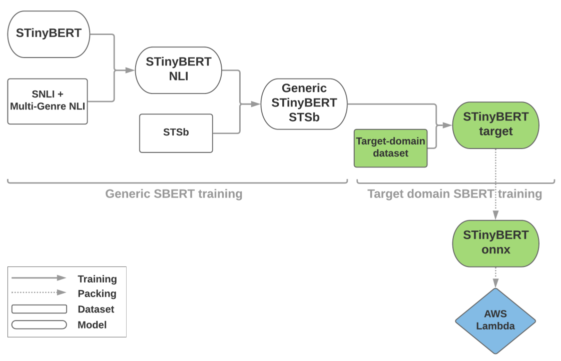


Figure 2: Schema of the fine-tuning pipeline of STinyBERT for STS task. In the first stage, STinyBERT is fine-tuned on NLI and STSb datasets to obtain Generic STinyBERT. In the second phase, the model is trained further on the target-domain dataset, exported to the ONNX format and deployed to AWS Lambda (see Section 5). The same pipeline was executed for SMobileBERT. SBERT$_{\text{BASE}}$ was only fine-tuned on target domain dataset.

with on-disk size of 255 MB. We therefore had to train our own SBERT model based on smaller BERT alternatives. We created the smaller SBERT models by employing the aforementioned BERT alternatives into the SBERT architecture, i.e. by adding an embedding averaging layer on top of the BERT model.

In order to make the smaller SBERT models perform on the STS task, we fine-tune them in two stages. Firstly, they are fine-tuned on standard datasets to obtain generic SBERT model and then further fine-tuned on target domain data. The fine-tuning pipeline is visualized in Figure 2.

4.2.1 Generic SBERT fine-tuning

To train a generic SBERT from smaller BERT-like models, we followed the SBERT training method as outlined in (Reimers and Gurevych, 2019) with slight alterations to hyperparameter values.

We first fine-tuned SBERT based on a smaller BERT alternative on a combination of SNLI (Bowman et al., 2015) and Multi-Genre NLI (Williams et al., 2018) datasets. We observed the best results when fine-tuning the model for 4 epochs with early stopping based on validation set performance, batch size 16, using Adam optimizer with learning rate 2×10^{-5} and a linear learning rate warm-up over 10 % of the total training batches.

We continued fine-tuning the model on the STS-benchark (STSb) dataset (Cer et al., 2017) using the same approach, except for early stopping based on STSb development set performance and a batch size of 128.

4.2.2 Target domain fine-tuning

Once we obtained a generic SBERT model trained on NLI and STSb data, we also fine-tuned it on examples from the target domain.

Dataset. We worked with a proprietary balanced dataset of 2856 pairs. Each pair was assigned to one of three classes: *duplicate* (target cosine similarity 1), *related* (0.5) or *unrelated* (0). The classes were assigned semi-automatically. *Duplicate* pairs were created by back-translation (Sennrich et al., 2016) using the translation models released as part of the OPUS-MT project (Tiedemann and Thottingal, 2020). *Related* pairs were pre-selected and expertly annotated and *unrelated* pairs were formed by pairing random texts together.

Validation and test sets are composed of 665 and 696 expertly annotated pairs, respectively. These sets are not balanced due to the fact that finding *duplicate* pairs manually is far more difficult than finding *related* or *unrelated* pairs, which stems from the nature of the problem. That is why *duplicate* class forms only approximately 13 % of the dataset, whereas *related* and *unrelated* classes each represent roughly 43 %.

As it is often the case with production systems, apart from small sets of annotated data, we also had access to a lot of data without annotations.

Fine-tuning on plain dataset. We first experimented with fine-tuning the generic SBERT model on the train set of the target domain dataset. We fine-tuned it for 8 epochs with early stopping based on validation set performance, batch size 64, Adam optimizer with learning rate 2×10^{-5} and a linear learning rate warm-up over 10 % of the total training batches.

Augmenting the dataset. Since we had a lot of data without annotations available, we also experimented with augmenting the dataset and fine-tuning Augmented SBERT (Thakur et al., 2020).

We pre-selected 379K duplicate candidates using BM25 (Amati, 2009) and annotated them using a pre-trained cross-encoder based on RoBERTa$_{\text{LARGE}}$. In the annotated data, low similarity values were majorly prevalent (median similarity was 0.18). For this reason, we needed to balance the dataset to get to a final balanced dataset of 32K pairs. We call the original expert annotations gold data and the cross-encoder annotations silver data.

After creating the silver dataset, we first fine-tuned the model on the silver data and then on the gold data. Correct hyperparameter selection was crucial for a successful fine-tuning. It was especially necessary to lower the learning rate for the final fine-tuning on the gold data and set the right batch sizes. For the silver dataset we used a learning rate of 2×10^{-5} and batch size of 64. For the final fine-tuning on the gold dataset we used a lower learning rate of 2×10^{-6} and a batch size of 16.

4.2.3 Results

As we can see in Table 3, smaller BERT alternatives can compete with SBERT$_{\text{BASE}}$. AugSMobileBERT manages to reach 93 % of the performance of SBERT$_{\text{BASE}}$ on the target dataset while being more than 3 times smaller in size.

We believe that the lower performance of smaller models is not only caused by the them having less parameters, but it also essentially depends on the size of the model's output dense vector. TinyBERT's output embedding size is 312 and MobileBert's is 512, whereas BERT$_{\text{BASE}}$ outputs embeddings of size 768. This would in line with the findings published in (Wieting and Kiela, 2019) which state that even random projection to a higher dimension leads to increased performance.

Model	STSb	Target
STinyBERT NLI	72.86	46.29
SMobileBERT NLI	78.29	52.08
SBERT$_{\text{BASE}}$ NLI	77.03	52.44
STinyBERT STSb	76.76	53.89
SMobileBERT STSb	81.52	59.05
SBERT$_{\text{BASE}}$ STSb	85.35	65.87
STinyBERT target	75.49	53.29
SMobileBERT target	79.56	59.27
SBERT$_{\text{BASE}}$ target	82.52	64.20
AugSMobileBERT target	**79.11**	**61.47**

Table 3: Spearman rank correlation between the cosine similarity of dense vectors and true labels measured for individual models on the test set of the STSbenchmark dataset (STSb column) and on the test set of the target domain dataset (Target column). The values are multiplied by 100 for convenience. We also present SBERT$_{\text{BASE}}$ performance as baseline. The model with the best performance on the target domain dataset, that is also deployable in serverless environment, is highlighted.

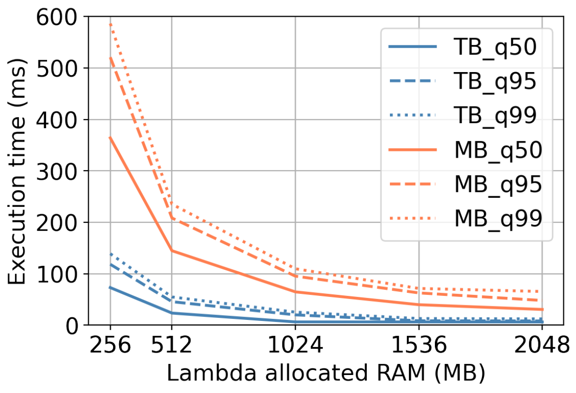
5 Deployment

As described in Section 3, numerous limitations must be satisfied when deploying a model to a serverless environment, among which the size of the deployment package is usually the major one. The deployment package consists of the function code, runtime libraries and in our case a model.

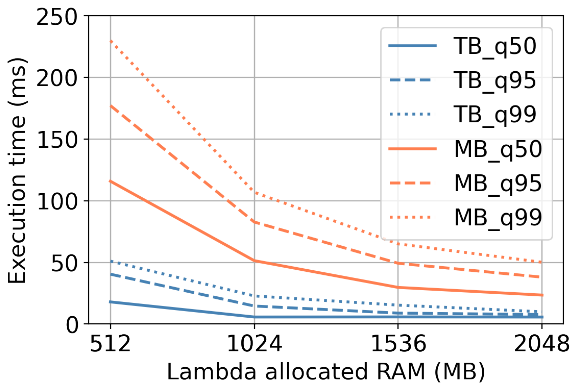
In order to fit all of the above in a few hundreds of MBs allowed in the serverless environments, standard deep learning libraries cannot be used: the standard PyTorch wheel has 400 MB (Paszke et al., 2019) and TensorFlow is 850 MB in size (Abadi et al., 2015).

We therefore used a smaller model interpreter library called ONNX Runtime (Bai et al., 2019), which is mere 14 MB in size, leaving a lot of space for the actual model. Prior to executing the model by the ONNX Runtime library, it needs to be converted to the ONNX format. This can be done using off-the-shelf tools, for instance the Hugging Face `transformers` library (Wolf et al., 2020) is for example shipped with a simple out-of-the-box script to convert BERT models to ONNX.

After training the models and converting them into the ONNX format, we deployed them to the AWS Lambda serverless environment.



(a) Sentiment analysis.



(b) SBERT encoding.

Figure 3: Results of performance tests of trained models deployed in AWS Lambda. TB stands for TinyBERT, MB for MobileBERT. q50, q95 and q99 denote the 0.5, 0.95 and 0.99 quantiles, respectively.

6 Deployment evaluation

We measured the performance of deployed models in scenarios with various amounts of allocated memory by making them predict on more than 5000 real-world examples. Before recording measurements we make the deployed model evaluate a small subsample of data in order to keep the infrastructure in a ”warm” state.

From the results in Figure 3 we can see that using the AWS Lambda, we can easily reach a 0.99 quantile of execution time below 100 ms for both tasks and models. We also observe that the execution time decreases with increasing RAM. This happens because AWS Lambda environment automatically allocates more vCPU with more RAM.

The AWS Lambda deployments are also cost-effective. The total costs of 1M predictions, taking 100 ms each and using 1 GB of RAM, are around \$2, whereas the cheapest AWS EC2 virtual machine with 1 GB of RAM costs \$8 per month.

7 Conclusion

We present a novel approach of deploying domain-specific BERT-style models in a serverless environment. To fit the models within its limits, we use knowledge distillation and fine-tune them on domain-specific datasets. Our experiments show that using this process we are able to produce much smaller models at the expense of a minor decrease in their performance. The evaluation of the deployment of these models in AWS Lambda shows that it can reach latency levels appropriate for production environments, while being cost-effective.

Although there certainly exist platforms and deployments that can handle much higher load (often times with smaller operational cost (Zhang et al., 2019)), the presented solution requires minimal infrastructure effort, making the team that trained these models completely self-sufficient. This makes it ideal for smaller-scale deployments, which can be used to validate the model’s value. The smaller, distilled models created in the process can then be used in more scalable solutions, should the cost or throughput prove inadequate during test deployments.

References

Martín Abadi et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Giambattista Amati. 2009. *BM25*, pages 257–260. Springer US, Boston, MA.

Timon Back and Vasilios Andrikopoulos. 2018. Using a microbenchmark to compare function as a service solutions. In *European Conference on Service-Oriented and Cloud Computing*, pages 146–160. Springer.

Junjie Bai et al. 2019. Onnx: Open neural network exchange. `https://github.com/onnx/onnx`.

Lucas Bernardi et al. 2019. 150 successful machine learning models: 6 lessons learned at booking. com. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1743–1751.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Jake Brutlag. 2009. Speed matters for google web search.

Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541.

Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation. *CoRR*, abs/1708.00055.

Jacob Devlin et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Xiaoqi Jiao et al. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Jeongchul Kim and Kyungyong Lee. 2019. Functionbench: A suite of workloads for serverless cloud function service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 502–504. IEEE.

Hyungro Lee et al. 2018. Evaluation of production serverless computing environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 442–450. IEEE.

Xin Li et al. 2019. Exploiting bert for end-to-end aspect-based sentiment analysis. *arXiv preprint arXiv:1910.00883*.

Theo Lynn et al. 2017. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *2017 IEEE CloudCom*, pages 162–169. IEEE.

O'Reilly Media, Inc. 2019. O'Reilly serverless survey 2019: Concerns, what works, and what to expect. `https://www.oreilly.com/radar/oreilly-serverless-survey-2019-concerns-what-works-and-what-to-expect/`. Accessed: 2021-01-12.

Adam Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Emma Strubell et al. 2019. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Zhiqing Sun et al. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. 2020. Augmented sbert: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. *arXiv preprint arXiv:2010.08240*.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.

Zhucheng Tu, Mengping Li, and Jimmy Lin. 2018. Pay-per-request deployment of neural network models using serverless architectures. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 6–10.

Liang Wang et al. 2018. Peeking behind the curtains of serverless platforms. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 133–146.

John Wieting and Douwe Kiela. 2019. No training required: Exploring random encoders for sentence classification. *arXiv preprint arXiv:1901.10444*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

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.

Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 1049–1062.