Exploiting Student Parallelism for Low-latency GPU Inference of BERT-like Models in Online Services

Weiyan Wang
Tencent
Hong Kong University of Science and Technology
Beijing, China
neowywang@tencent.com

Yilun Jin
Hong Kong University of Science and Technology
Hong Kong SAR, China
yilun.jin@connect.ust.hk

Yiming Zhang
Shanghai Jiao Tong University
Shanghai, China
zhangyiming@cs.sjtu.edu.cn

Victor Junqiu Wei
Macau University of Science and Technology
Macau SAR, China
wjqjsnj@gmail.com

Han Tian
Hong Kong University of Science and Technology
Hong Kong SAR, China
htianab@connect.ust.hk

Li Chen
Zhongguancun Laboratory
Beijing, China
lichen@zgclab.edu.cn

Jinbao Xue
Tencent
Beijing, China
jinbaoxue@tencent.com

Yangyu Tao
Tencent
Beijing, China
brucetao@tencent.com

Di Wang
Tencent
Beijing, China
diwang@tencent.com

Kai Chen*
Hong Kong University of Science and Technology
Hong Kong SAR, China
kaichen@cse.ust.hk

Abstract

BERT-like models have been widely adopted in text mining and web search due to their high accuracy. However, large BERT-like models suffer from inefficient *online* inference on GPUs for two main reasons. First, their high accuracy relies on large model depth, which linearly increases sequential computation on GPUs. Second, stochastic and dynamic online workloads lead to extra costs due to batching and padding. To address the problem, we present Student Parallelism for efficient GPU inference of BERT-like models under real-world online workloads. At its core, Student Parallelism adopts stacking distillation and boosting ensemble, distilling the original deep model into a group of shallow but virtually stacked student models running in parallel. This enables Student Parallelism to achieve a low model depth (e.g., two layers), and thus low inference latency while maintaining accuracy. In addition, we design adaptive student pruning to adjust the number of students according to the dynamic online workloads. For example, during workload bursts, it can temporarily decrease the number of students with minimal accuracy loss to improve system throughput. Extensive experiments on real-world datasets and workloads show that Student Parallelism achieves up to 4.1× lower latency while maintaining accuracy and up to 22.27× higher throughput during workload bursts.

*Corresponding author

CCS Concepts

• **Computing methodologies** → **Natural language processing**.

Keywords

Text Mining, Information Retrieval, Efficient Inference, Distillation

ACM Reference Format:
Weiyan Wang, Yilun Jin, Yiming Zhang, Victor Junqiu Wei, Han Tian, Li Chen, Jinbao Xue, Yangyu Tao, Di Wang, and Kai Chen. 2025. Exploiting Student Parallelism for Low-latency GPU Inference of BERT-like Models in Online Services. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3736949

1 Introduction

BERT-like models [11, 17, 30] have been the cornerstone for discriminative natural language understanding (NLU) tasks like text mining and web search in online services. However, large BERT-like models obtain strong model capability at the cost of slow inference, limiting their applications in time-sensitive services that require efficient

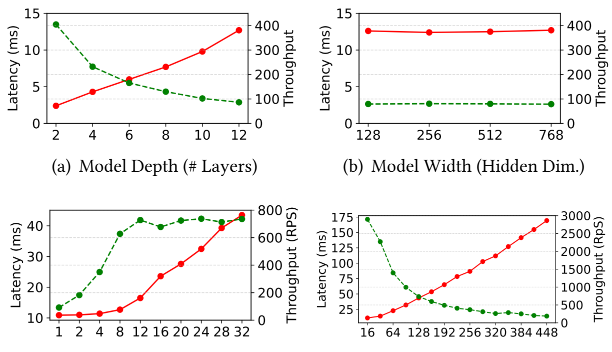


Figure 1: Impacts of various model and input factors on inference latency (red solid line, left axis) and throughput (green dashed line, right axis). Detailed settings in Appendix B.1.

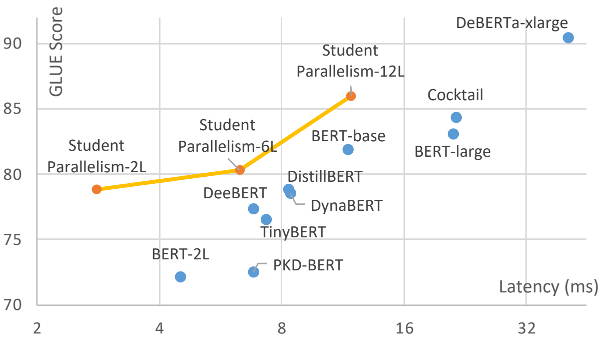


Figure 2: Accuracy-latency tradeoff comparison of all compared methods. Methods closer to the upper-left corner achieve better tradeoff.

online inference. For example, news sentiment classification for automatic stock trading is time-sensitive as even a sub-millisecond delay can impact the profits [3]; Retrieval Augmented Generation (RAG) applications [28] generally have a limited time budget of $\sim$ 10ms [53] for each query to search for relevant web pages for informed answer generation; online writing assistants need to immediately respond to text streams while users are typing [22].

Much effort has been devoted to efficient BERT-like model inference, such as compact model designs [44, 52], knowledge distillation [21, 39, 43], model pruning [13, 18, 34], and early exit [29, 51, 58]. Although they are effective in reducing the overall amount of computation during inference, they suffer from two drawbacks and are thus unsuitable for efficient GPU inference for online services. First, these methods often fail to reduce the number of model layers, leading to significant inference latency. Since deep learning essentially relies on model depth for high model capability [27], a high number of layers is necessary to maintain model accuracy. For example, methods based on compact model designs, knowledge distillation, and model pruning primarily reduce the width of the models (i.e. hidden dimension) instead of their depth (about half of the layers remained). Moreover, for methods based on early exit, there are always hard data samples that go through all model layers. Consequently, the high number of layers results in a significant amount of sequential computation, which still suffers from high inference latency on parallel-computing devices such as GPUs even with reduced model sizes. Figure 1(a) shows that model depth linearly increases the latency due to the layer-by-layer forward propagation. In contrast, both latency and throughput remain the same with varying model widths (as long as they lie within the computation capacity of the GPU). Therefore, reducing model depth is crucial for efficient GPU inference of BERT-like models.

Second, existing works suffer from extra overheads of batching and padding upon dynamic online inference workloads. Large data batches are often favored by GPUs due to their strong parallel computing capacity [26]. For example, as shown in Figure 1(c), inference throughput keeps increasing while latency remains stable until a batch size of 12. However, different from pre-processed regular data batches during training, samples during online inference have different lengths and arrive in dynamic patterns. Therefore, a waiting queue is often used to batch sufficient data samples during online inference [2, 9, 57], leading to increased latency for samples that arrive earlier. Moreover, as GPUs require uniform data shapes, padding is applied to pad shorter data samples to the maximum length of the batch [11], leading to extra computation overhead. As shown in Figure 1(d), inference latency increases with longer sequences. Consequently, the overhead of batching and padding should be mitigated for efficient inference for online services.

To address the challenges, we present Student Parallelism for efficient GPU inference of BERT-like models for real-world online workloads. At its core, Student Parallelism distills the original deep model into an adaptive group of parallel, shallow, but virtually stacked student models. Thus, the sequential layer-by-layer computation is replaced with parallel computation of students while maintaining model accuracy. To achieve this, we propose a novel knowledge distillation method based on explicit boosting ensemble and virtual stacking distillation (§ 3.1). Despite their parallel inference, students are sequentially trained with boosting ensemble to gradually reduce the error between the teacher and the group of students. Furthermore, each student distills the output features of all previous students into its intermediate layer, so that it virtually stacks on previous students. Consequently, the group of students can maintain accuracy with significantly fewer layers (1/4 of the original number of layers — sometimes only two), thus reducing inference latency on GPUs. Beyond distillation, we conduct adaptive student pruning to improve generalization and enable dynamic reduction of the number of students. By dynamically dropping some last-trained students, Student Parallelism improves generalization of the remaining students. More importantly, it enables dynamic adjustments of the number of students according to the changing workload. Upon workload bursts, it can temporarily reduce the number of students (e.g. one or two) with minimal accuracy loss. With Student Parallelism, we further make specialized designs to reduce the overhead of batching and padding. It allocates the students of the same group on different GPUs to better parallelize inference of a single sample. It further replicates the student group into multiple copies to run concurrently on the same set of GPUs for better GPU utilization. By employing the length-aware buffer, it enables immediate and concurrent inference of different samples to be free of batching large and regular data.

Evaluation results (§ 4) on real-world online workload traces show that Student Parallelism achieves up to 4.1× lower average and tail latency while achieving the best prediction accuracy in all baselines with compressed models. Upon workload bursts, Student Parallelism achieves up to 22.27× higher throughput with competitive accuracy by adaptively reducing the number of students. In addition, Student Parallelism can effectively compress 24-layer and 48-layer BERT-like models into students with only 1/4 of the original number of layers (6 and 12 layers, respectively), while other baselines fail. Figure 2 demonstrates that Student Parallelism achieves a better tradeoff between accuracy and latency, where Student Parallelism either achieves better accuracy with the same latency or lower latency with the same accuracy.

Our contributions can be summarized as follows:

- We leverage virtual stacking and boosting ensemble to distill the deep model into a group of parallel and shallow students for low inference latency with comparable accuracy.
- To handle online workloads, we can reduce the student number with minimal accuracy loss for workload bursts, and we also make specialized designs for student parallelism to reduce waiting and padding.
- Comprehensive experiments with real-world workload traces verify that Student Parallelism achieves better tradeoff between accuracy and efficiency compared to baselines.

2 Related Work

Previous works have explored various methods for efficient inference of BERT-like models. However, they still suffer from high inference latency due to large model depth and extra overhead upon online workloads. Existing works can be categorized as follows:

- **Model Compression**. Methods based on model compression generate a single model with fewer parameters but still suffer from considerable model depth and high latency on GPUs. MobileBERT [44] and NAS-BERT [52] make *compact model designs* that can achieve similar accuracies. *Knowledge distillation* transfers the knowledge from a large teacher model [21, 39, 43] or teacher ensemble [4] to a small student model, so that the student model mimics the output of the teacher model. *Pruning* removes redundant weights, attention heads, or tokens in well-trained models for lower inference costs and better generalization [18, 25, 31]. Methods based on *early exit* [29, 51, 58] add multiple classifiers on different layers as paths for early exit, so that 'easy' samples do not go through all layers. *Quantization* [24, 41, 45, 54] replaces high-precision floating point operators with faster integer operators that can be executed in a higher level of parallelization on GPUs. However, deep learning essentially relies more on model depth than model width for high model capability [27], so they still require a considerable number of model layers to maintain model accuracy. For example, MobileBERT and NAS-BERT are similarly deep (12 or 24 layers) but with lower width to avoid accuracy loss. DistilBERT with 6 layers can maintain 96.3% of the score of 12-layer teacher BERT model, but the 3-layer PKD-BERT only maintains 88.5% (§ 4.2.1). Model pruning methods [18] can only reduce up to a half of layers to maintain model accuracy. In models with early exit, "difficult" samples still go through all layers. Quantization methods maintain the same model depth as the original model.
- **Model Ensemble**. Methods based on model ensembles generate multiple smaller models from the original large model for efficient inference. *Mixture-of-Experts (MoE)* models [40, 60] have routers to activate a subset of experts in each MoE layer and thus can be viewed as an implicit ensemble of experts. It can scale up model width while maintaining constant inference costs, but fails to decrease depth of the model. *Bagging ensembles* like Cocktail [8, 15] select different smaller models from a model pool to run in parallel for lower latency and high accuracy. However, these methods rely on different model architectures to improve the diversity of the ensemble, resulting in different inference latencies and thus the straggler problem. *SensAI* [49] divides the original multi-class classification model into an ensemble of pruned binary classification models, which still have large model depths and irregular pruned architectures. *Progressive ensemble distillation* [10] aims to progressively adds more student models with different architectures to gradually approximate the accuracy of the teacher model. However, the maximum number of layers among the heterogeneous students remains large, and its accuracy decreases significantly upon any dropped student. *Collaborative and Self-distillation* [42, 55, 59] explore distilling multiple teachers and existing students into a new better student for higher accuracy, but they do not leverage multiple students together to reduce the number of layers.
- **Handling Online Workload.** Methods for efficient model inference discussed above generally ignore the stochastic online workloads and thus suffer from overheads caused by batching and padding. They reserve some over-provisioned GPUs in advance for potential workload bursts, which are underutilized for most of the time [15, 38, 57]. According to workload prediction and profiling, some existing works can dynamically set the batch size and maximum waiting time to reduce the idle waiting [2, 57], but their effectiveness are bounded by the empirical profiling. Some recent works [1, 5, 36, 56] bring in supports of ragged batches to avoid extra padding, but their support of operators and application scenarios remains limited [12].

In contrast, Student Parallelism trains a group of parallel students with low model depth instead of a single student with smaller size but similar depth. All students have the same model architecture, such that Student Parallelism does not suffer from stragglers. Through adaptive student pruning, Student Parallelism can reduce the number of student with minimal accuracy loss to achieve higher throughput, thus alleviating the need for over-provisioned GPUs. With student parallelism, we also make specialized designs to save the costs of batching and padding for online workloads. Besides, Student Parallelism is orthogonal to quantization and can thus quantize all students for further efficiency improvement.

3 Methodology: Student Parallelism

In this section, we first show how to combine virtual stacking and boosting ensemble to train the student group generating similar

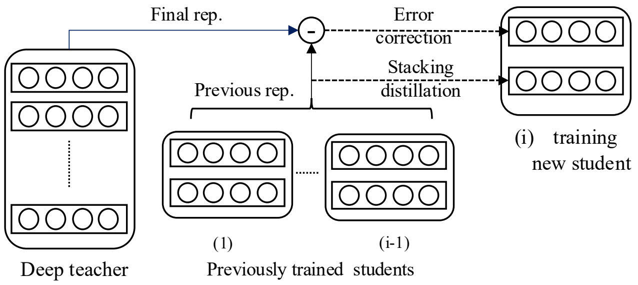


Figure 3: The training process of Student Parallelism. The intermediate layer of a student imitates the output of all previous students via virtual stacking distillation, while the top layers reduce the residual error via boosting ensemble.

Algorithm 1 Sequential Student Training for Student Parallelism

```
Require: Teacher model $T(x)$, Small pre-trained language model $S_{pre}(x)$
1:  studentList = [$S_{pre}(x)$]
2:  $i = 1$
3:  while $B^{(i-1)}(x)$ not overfitting do
4:      $S_i(x)$ = studentList[-1].copy()
5:      Sample training subset for the $i$-th student.
6:      while Eqn. 4 not converged do
7:          Optimize Eqn. 4 to train the $i$-th student $S_i(x)$.
8:      end while
9:      studentList.append($S_i(x)$)
10: end while
11: studentList = studentList[:-1]
```

final representations of the deep teacher. Then, we introduce our adaptive student pruning to train the final classifier to achieve better generalization and enable dynamic student number. Finally, we describe how to orchestrate the multiple students to run in parallel for online inference workloads.

3.1 Virtual Stacking and Boosting Ensemble

We recall that BERT-like models stack all the attention layers with residual connections, which add the input of a layer x back to the output of the layer $F(x)$, making its output $H(x) = F(x) + x$. Because there are multiple attention layers stacked with residual connections, the output of the model can be considered as the sum of multiple forward passes within the BERT-like model, with each path going through different layers, which behaves like the ensemble of networks with different depths [19, 47].

Inspired by the implicit ensemble within BERT, we propose to combine virtual stacking and boosting ensemble to build a group of parallel student models with fewer layers, while generating similar final representations as deep teachers. To alleviate stragglers, all students share the same shallow model architecture, resulting in weak model capacity and diversity. To improve model capacity, students are not only added to gradually reduce the error (i.e. boosting), but also trained with stacking distillation to imitate virtual stacking of a student upon previous ones, as illustrated in Figure 3. In addition, the training strategy (Algorithm 1) improves the diversity of the ensemble from other aspects, including different training losses, initializations, and subsets of training data.

3.1.1 Boosting Ensemble. To train the group of students for Student Parallelism, we first train the first student to directly mimic the output representations of the teacher. Then, further students are sequentially trained in the style of boosting ensemble to gradually correct the residual error between the final representations of the previous students and of the teacher model. We keep adding new students until overfitting happens on validation data (Line 3, Algorithm 1). Formally, we define the boosting ensemble with M students, each having N layers to generate the final representations

$$B^{(M)}(x) := \sum_{m=1}^{M} \alpha^{(m)} S_N^{(m)}(x) \tag{1}$$

where $S_N^{(m)}$ is the representations of the m-th student model of its N-th layer, and $\alpha^{(m)}$ is its multiplier in the ensemble. We optimize the following residual training loss $L_{\text{boost}}^{(i)}$ to train the i-th student,

$$L_{\text{boost}}^{(i)}(x) = \frac{1}{2}\left\|T(x) - B^{(i-1)}(x) - S_N^{(i)}(x)\right\|^2, \tag{2}$$

where $T(x)$ is the final representations of the teacher model, $B^{(i-1)}(x)$ is the boosting ensemble of the previous $i-1$ students. We keep other models like $T(x)$ and $B^{(i-1)}(x)$ fixed and only train the i-th student model to optimize $L_{boost}^{(i)}(x)$. After training the i-th student, we can set its multiplier $\alpha^{(i)}$ by line search to minimize the loss $\frac{1}{2}(T(x) - B^{(i)}(x))^2$, except that $\alpha^{(1)}$ is always 1.

3.1.2 Virtual Stacking. Although all students run independently in parallel, we propose stacking distillation to virtually stack every student upon all previous students. Specifically, each student learns to distill the ensemble of its previous students into the representations of its $\lceil N/2 \rceil$-th layer. By doing so, the intermediate representations of the i-th student imitates the ensemble of all the previous students, and thus the following layers can better correct the error by refining the representations similar to a deep BERT-like model. The loss $L_{\text{stack}}^{(i)}$ for stacking distillation for the i-th student is defined as follows

$$L_{\text{stack}}^{(i)}(x) = \frac{1}{2}\left\|B^{(i-1)}(x) - S_{\lceil N/2 \rceil}^{(i)}(x)\right\|^2 \tag{3}$$

where $S_{\lceil N/2 \rceil}^{(i)}(x)$ denotes the representations of the $N/2$-th layer in the i-th student model. We keep $B^{(i-1)}$ fixed and only train $S^{(i)}$.

The overall loss $L^{(i)}$ to train the i-th student for Student Parallelism is as follows,

$$L^{(i)} = L_{\text{boost}}^{(i)} + \lambda L_{\text{stack}}^{(i)} \tag{4}$$

where λ is a hyper-parameter to balance boosting distillation and virtual stacking distillation. Each student is trained with a different $L_{\text{stack}}^{(i)}$, thus improving the diversity of the ensemble.

Besides different training losses for different students, Student Parallelism also improves the diversity of students from different perspectives. All students have different initialized parameters. The first student is initialized with a small pre-trained language model

Algorithm 2 Adaptive Student Pruning

```
Require: Teacher model $T(x)$, group of student models $S^{(i)}(x), i = 1 \dots M$ learned from Algorithm 1, a classifier layer Classifier.
1: while training do
2:     Sample a batch of data $x$.
3:     softLabel = $T(x)$
4:     for $k \leftarrow 1$ to $M$ do
5:         $B^{(k)}(x) = \sum_{j=1}^{k} \alpha^{(j)} S^{(j)}(x)$
6:         output$_k$ = Classifier($B^{(k)}(x)$)
7:         $L_{prune}^{(k)}$ = softCrossEntropy(output$_k$, softLabel)
8:         Calculate and accumulate gradients for $S^{(j)}(x), j = 1 \dots k$.
9:     end for
10:    Update all students with the accumulated gradients
11: end while
12: for $k \leftarrow 1$ to $M$ do
13:     Evaluate Classifier($B^{(k)}(x)$) on the validation set.
14: end for
15: Choose the best-pruned group of students.
```

with fewer layers, while later students are initialized with the parameters of the previous student (Line 4, Algorithm 1). Additionally, each student model is trained on a different subset of training data (Line 5, Algorithm 1), with only the first student trained on all training data. For each student, we pick the top $a\%$ samples with the largest residual errors between the teacher and the existing students and randomly sample another $b\%$ of all training data. All subsets of training data are augmented following previous works [18, 21].

Following the theory of boosting ensemble [32, 33], we make mathematical proof on the convergence of the final representations of all students based on the gradient boosting ensemble as follows:

Theorem 3.1. *The MSE loss in Student Parallelism has the lower bound 0, and it is Lipschitz differentiable loss function (for any $L > 1$, we always have $| \nabla L_{boost}(x_1) - \nabla L_{boost}(x_1)| < L|x_1 - x_2|$). Let $F^{(0)}, F^{(1)}, \dots$ be the sequence of combined hypotheses generated by the Student Parallelism training algorithm, using small enough step-sizes $\alpha_i := -\frac{\langle \nabla L_{boost}(T, B^{(i-1)}), S^{(i)} \rangle}{L|S^{(i)}|^2}$. Then our sequential training of students either halts on round T with $-\langle \nabla L_{boost}(T, F^{(i-1)}), S^{(i)} \rangle \leq 0$, or $L_{boost}(F^{(i)})$ converges to some finite value L^*, in which case*

$$\lim_{i} \langle \nabla L_{boost}(T, B^{(t)}), S^{(t+1)} \rangle = 0. \tag{5}$$

More proof details can be found in Appendix A.1.

3.2 Adaptive Student Pruning

In this section, we introduce our adaptive student pruning, which prunes the students $S^{(j)}(x), j = 1 \dots M$ obtained via Algorithm 1 for better generalization and dynamic adjustment of the number of students. We follow sub-network training in Neural Architecture Search (NAS) [7] to prune redundant students for more robust final representations. Then, a classifier layer takes the final representations from different numbers of students to mimic the predictions of the teacher. Since later students in the boosting ensemble gradually

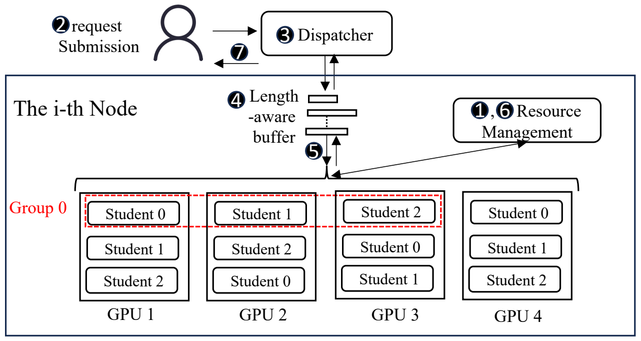


Figure 4: Overview of strategies for online inference: student allocation and inference sample dispatching.

reduce the residual error, we simply drop and prune the last students without computing importance scores as in previous pruning methods [18]. Following the theory of Occam's razor, student pruning can improve generalization by removing redundant students and only keeping necessary ones. Moreover, adaptive student pruning enables dynamic adjustment of the number of students with minimal accuracy loss. For example, for high throughput, we can simply drop some of the last students, while the remaining ones are fine-tuned with adaptive student pruning to maintain high accuracy. § 4.3.1 shows the better generalization of the best-pruned set of students and better accuracy with different numbers of students.

We state the details of adaptive student pruning in Algorithm 2. For each batch of training data, it samples different numbers of students (Line 4 in Algorithm 2) to generate the final representations (Line 5), which incurs limited communication costs due to the small size of the final representations. We then put a classifier layer on the output representations of the selected students (Line 6). We take the soft labels generated by the teacher model (Line 3) as the target and apply the soft cross-entropy loss (Line 7) to train the students and the classifier. Since there are multiple forward and backward passes with different numbers of students in each batch of data, we accumulate all gradients to update the classifier and all students (Lines 8 and 10). Finally, we evaluate different numbers of first-k students on the validation set to choose the best-pruned group of students instead of keeping all students (Lines 12-15).

3.3 Efficient Inference upon Online Workloads

We design specialized student allocation and inference sample dispatching strategies for efficient GPU inference upon online workloads to reduce the extra costs of batching and padding to collect large and regular data batches. We assume that the GPU cluster for online inference has multiple nodes, each of which has multiple GPUs, and a global dispatcher to balance the workloads among all nodes. Figure 4 shows the overview and procedure of online inference as follows:

(1) In the initialization, the resource manager starts multiple student groups running in parallel, and every group distributes its students over multiple GPUs.
(2) User submits a request triggering one or more inference samples.
(3) The dispatcher assigns the samples to different GPU nodes in the round-robin manner.

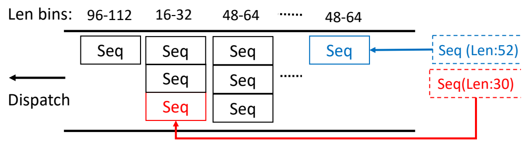


Figure 5: Inference sample dispatching with length-aware buffers. A new sample (the red one with length 30) can be merged with previous ones into a small batch when they lie within the same length range and the buffer is not full. Otherwise, when the buffer is full, the new sample (e.g. the blue one with length 52) is added to a new buffer at the tail.

(4) After reaching the GPU node, the sample is pre-processed and then enters the length-aware buffer. And if the buffer is not empty, it tries to merge the samples of the similar length into small batches.
(5) Whenever any student group is idle, it immediately sends out the first buffered data sample or batch of samples in similar lengths for immediate inference without waiting.
(6) According to the dynamic intensity of online workload, resource management decreases the student number for larger throughput or increases it for better accuracy.
(7) It returns results to users.

3.3.1 Student Allocation. As shown in Figure 4, with Student Parallelism, we can distribute a group of students over multiple GPUs in the same node, thereby better parallelizing inference for different samples. NVIDIA Multi-Process Service (MPS) [35] can guarantee different students on the same GPU have separate computation units and memory bandwidth to run concurrently without interference. Although communication is needed for students in the same group to gather the final representations, it only takes about 0.2 ms due to the small size of the final representations and fast NV-Link or PCI-E. We further replicate the student group into multiple copies that run on the same set of GPUs in the node. Specifically, assuming M students running on a total of G GPUs on a node, the i-th student of the j-th student group is allocated on

$$\mathrm{GPU}_{id} = (i + j \times M)\%G. \tag{6}$$

Therefore, if $i + j \times M > G$, students from multiple groups will locate on the same GPU to process different samples concurrently.

During inference, the student allocation strategy keeps tracking the intensity of online workloads (e.g., requests per second) to adjust the number of student accordingly. It temporarily decreases the number of students (§ 3.2) to run more student groups for workload bursts, instead of reserving idle GPUs in advance. Correspondingly, it can also increase the number of students for high accuracy after workload bursts. More details about hyper-parameters for student allocation can be found in Appendix B.3.2).

3.3.2 Inference Sample Dispatching. To reduce the extra costs of idle waiting and padding, Student Parallelism conducts immediate and concurrent inference for different samples. If there are idle student groups and no buffered samples, the dispatcher can immediately send out the current sample for inference without waiting for batching. Since samples do not arrive at the same time, different student groups can concurrently process them by overlapping data transfer to GPU and computation for inference.

However, upon workload bursts, there may be a massive number of concurrent samples that occupy all student groups. We thus design length-aware buffers to merge samples with similar lengths into data batches without much padding. To efficiently merge samples on-the-fly, the length-aware buffers are organized with a dictionary mapping length ranges to positions to achieve $O(1)$ time complexity, as shown in Figure 5. More details about the length-aware buffer can be found in Appendix B.3.3.

3.3.3 Efficiency Analysis. We build an analytical model to quantitatively analyze the efficiency of Student Parallelism. By considering factors like model depth D, model width W, batch size B, input length N, waiting time $Q(B)$, computation capacity of each GPU C, PCI-E bandwidth T, the number of GPUs G, and the number of students M, inference latency can be modeled as:

$$\text{Latency} \propto D\left\lceil \frac{WBN^2M}{CG} \right\rceil + Q(B) + \frac{BN}{T} \tag{7}$$

Compared to existing works, Student Parallelism has the advantages of lower model depth (lower $D\left\lceil \frac{WBN^2M}{CG} \right\rceil$) and lower overhead for waiting and padding (lower $Q(B)$). Please refer to Appendix A.2 for details about the quantitative analysis and comparison.

4 Evaluation

In this section, we evaluate the effectiveness of Student Parallelism in terms of accuracy and efficiency. We also conduct ablation studies to analyze the impacts of individual designs in Student Parallelism and hyper-parameters. We draw the following main conclusions:

- **Accuracy**. Student Parallelism achieves the best accuracy among all compact models with reduced depth or width. With adaptive student pruning, Student Parallelism achieves comparable accuracy even with only one student remaining.
- **Efficiency**. Student Parallelism achieves up to 4.1× lower latency and up to 22.27× higher throughput compared to baselines due to the lower model depth, minimal waiting and padding, and adaptive student pruning.

4.1 Experimental Settings

4.1.1 Implementation Details. We search from all 24 compact BERT variants [46] and all DeBERTa variants to find the one which needs the fewest layers to reach the target accuracy of various teacher models. In practice, we use BERT-2L256D [46] for BERT-base, BERT-6L768D for BERT-large, and DeBERTa-12L768D for DeBERTa-xlarge as the student models[1]. We conduct the same data augmentation following [21, 51]. We conduct a grid search on various hyper-parameters, including epoch number, batch size, learning rate, λ, and data sampling ratio. We optimize $\alpha^{(i)}$ in Eqn. 1 together with the student model as they are both differentiable. We run experiments over 16 NVIDIA RTX3090 on 4 nodes. More implementation details can be found in Appendix B.

4.1.2 Datasets and Metrics. We use the GLUE benchmark [48] following [18, 39, 51]. It consists of 8 datasets with various text mining (SST-2, CoLA, MNLI, and RTE) and information retrieval (MRPC, QQP, STS-B, and QNLI) tasks. For each dataset, we report the metric

[1]The suffix -pLqD denotes a model with p layers, each of which has a hidden dimension of q. Also introduced in Tables 1 and 2.

Compared Methods	**# Params (Millions)**	SST-2	RTE	MNLI	CoLA (MCC)	QQP	MRPC (F1)	STS-B (PCC)	QNLI	**Overall Score**
*BERT-base-12L768D	109	92.7	67	83.6	52.8	89.6	88.6	89	91.8	*81.89 (100%)
*I-BERT-12L768D	109	91.6	65.7	81.3	49.1	87.1	85.2	86.9	87.4	*79.29 (96.82%)
DeeBERT-12L768D	109	91.5	66.1	80	43.4	87.3	85.2	-	87.9	77.34 (94.45%)
DistillBERT-6L768D	67	91.3	58.4	81.1	49	88.1	86.9	86.9	88.9	78.83 (96.26%)
DynaBERT-6L192D	9	92	63.2	82.15	43.7	90.4	81.4	87	88.5	78.54 (95.92%)
TinyBERT-4L312D	15	87.8	60.8	76.9	44.1	87.7	85.8	83.3	86	76.55 (93.48%)
PKD-BERT-3L768D	46	87.5	58.2	76.3	24.8	87.8	80.7	79.8	84.7	72.48 (88.51%)
BERT-2L256D	10	87.1	54.6	74.7	23.2	87	80.3	86	84.4	72.16 (88.12%)
Student Parallelism-2L (pruned to 1)	10	91.3	64.9	77.3	42.3	88.1	86.7	85.2	88.2	78.00 (95.25%)
Student Parallelism-2L (best-pruned)	10M	91.8	65.9	78.2	43.2	88.7	86.9	86.4	89.8	**78.86 (96.31%)**
*BERT-large-24L1024D	335	93.2	70.4	86.6	60.6	91.3	89.3	90	92.3	83.06 (100%)
Student Parallelism-6L768D (best-pruned)	67M	92.1	61.4	82.5	51.2	89.2	88.9	87.2	89.7	**80.28 (96.65%)**
*DeBERTa-xlarge-48L1024D	658	97	93.1	91.3	70	92.3	94.3	92.8	95.1	90.40 (100%)
Cocktail	-	94.2	72.1	87.1	62.1	91.5	89.7	-	93.4	84.30 (93.25%)
Student Parallelism-12L768D (best-pruned)	109M M	94.1	78.7	88.8	59.3	91.9	90.1	91.5	93.1	**85.94 (95.06%)**

Table 1: Experimental results on prediction accuracy. * denotes the teacher model with full depth and width; the suffix -pLqD denotes a model with p layers and a hidden dimension of q; results in bold are the best overall scores of models without full depth or width; results with underlines meet the target prediction quality (95%). M is the number of students in the best-pruned student group. DeeBERT and Cocktail require class probabilities and thus cannot handle STS-B with continuous outputs.

recommended by GLUE, including accuracy, F1 score, Matthews correlation coefficient (MCC), and Pearson correlation coefficient (PCC). We build our workload generator upon the real-world online trace of twitter [23]. We report **average** and **95% tail latency** for text mining tasks and **throughput** for information retrieval tasks, as retrieval often involves a larger number of samples.

4.1.3 Teachers Models and Baselines. We adopt the fine-tuned 12-layer **BERT-base-12L768D** [11], 24-layer **BERT-large-24L1024D** [11], and 48-layer **DeBERTa-xlarge-48L1024D** [17] as the teacher models. We set the target prediction quality as 95% of the overall scores achieved by the teacher models, as recommended by MLPerf [37].

We train 2-, 6-, and 12-layer student models for Student Parallelism for each teacher model correspondingly, reducing over 75% of layers. We compare Student Parallelism with various baselines related to BERT-like models.

- For the teacher model of BERT-base-12L768D, we conduct quantization to get **I-BERT-12L768D** with the same model architecture. We adopt **DeeBERT-12L768D** [51] as the baseline for methods based on early exit. We adopt the 3-layer **PKD-BERT-3L768D** [43], 4-layer **TinyBERT-4L312D** [21], and 6-layer **DistilBERT-6L768D** [39] as baselines for knowledge distillation. For pruning baselines, we choose the fastest version of **DynaBERT-6L192D** [18] satisfying the target prediction quality. We also compare with shallow fine-tuned models like **BERT-2L256D** [46].
- The above baselines only work for BERT-base instead of larger teacher models like BERT-large and DeBERTa-xlarge. Therefore, we only compare our work with **Cocktail** based on bagging ensemble to approach the performance of the teacher model DeBERTa-xlarge.

Following [16, 57], all baselines employ dynamic batch sizes for dynamic online workloads. We set the time budget as 10ms to set the batch size and waiting window according to online workloads [57].

4.2 Overall Performance

In this section, we show that Student Parallelism can maintain prediction accuracy while improving inference latency and throughput.

4.2.1 Prediction Accuracy. As shown in Table 1, for BERT-base, Student Parallelism achieves the highest prediction accuracy among all compared methods with only two-layer students. In contrast, other baselines meeting the target prediction quality can only reduce half of the layers of BERT-base (DistilBERT-6L, DynaBERT-6L). Compared with baselines with no more than 6 layers, Student Parallelism has a significant advantage in prediction accuracy. For example, 3-layer PKD-BERT and 2-layer BERT-2L512D only maintain <90% of the overall score of BERT-base. Student Parallelism also outperforms all baselines with 6 or more layers (i.e. over 3× deeper than Student Parallelism) and is comparable with I-BERT having full model depth and width. With improved generalization from adaptive student pruning, Student Parallelism can use only one student to handle workload bursts with a competitive overall score (95.25% of the BERT-base). In addition, with deeper student models, Student Parallelism can also reduce the model depth of larger teacher models (BERT-large and DeBERTa-xlarge) by 75%, while other baselines fail. For example, Cocktail fails to meet the target prediction quality even with 24 layers.

4.2.2 Latency. In discriminative text mining tasks like news sentiment classification for automatic trading, users mainly care about the inference latency of individual samples. Therefore, Table 2 has two columns showing the average and tail latency of all methods on the SST-2 dataset. Student Parallelism achieves both the lowest average and tail latency among all methods with a reduction of 4.1 ∼ 1.6×. Student Parallelism-2L3S is the best-pruned version with a slightly higher latency than Student Parallelism-2L1S due to the final communication among students. Compared with other baselines except for BERT-2L, Student Parallelism has a lower model depth to achieve low latency. Furthermore, it does not require long waiting time to collect data batches, leading to a lower latency than BERT-2L. The different lengths are the main reason for small latency variances in Student Parallelism. Because average latency

Compared Methods	Model Config (# Params)	Avg. Latency		95% Tail Latency		Throughput per GPU	
		Value (ms)	Reduction	Value (ms)	Reduction	Value (# samples/s)	Improvement (×)
*BERT-base	12L768D (109 M)	11.6	0.00%	11.7	0.00%	691.5	1.00
*I-BERT	12L768D (109 M)	4.9	-57.8%	9.5	-18.80%	2115.99	3.06
DeeBERT	*12L768D (109 M)*	*6.8*	*-41.38%*	*11.5*	*-1.71%*	*941.4*	*1.36*
DynaBERT	6L192D (9 M)	8.3	-28.45%	9.6	-17.95%	7538.7	10.9
DistilBERT	6L768D (67 M)	8.4	-27.59%	9.7	-17.09%	1508.1	2.18
TinyBERT	*4L312D (15 M)*	*7.3*	*-37.07%*	*9.7*	*-17.09%*	*5873.6*	*8.49*
PKD-BERT	*3L768D (46 M)*	*6.8*	*-41.38%*	*9.5*	*-18.80%*	*2781.3*	*4.02*
BERT-2L	*2L256D (10 M)*	*4.5*	*-61.21%*	*9.4*	*-19.66%*	*14420.3*	*20.85*
Student Parallelism-2L1S	2L256D1S (10 M)	**2.8**	**-75.86%**	**3.6**	**-69.23%**	**15398.6**	**22.27**
Student Parallelism-2L3S	2L256D3S (30 M)	**3.0**	**-74.13%**	**3.7**	**-68.37%**	5201.91	7.52
*BERT-Large	24L1024D (335 M)	21.1	0.00%	21.3	0.00%	239.6	1.00
Student Parallelism-6L1S	6L768D1S (67 M)	**6.3**	**-70.14%**	**7.1**	**-66.67%**	**1508.2**	**6.29**
Student Parallelism-6L3S	6L768D3S (201 M)	**6.6**	**-68.87%**	**7.3**	**-65.72%**	512.4	2.14
*DeBERTa-xlarge	48L1024D (658 M)	40.5	0.00%	41.4	0.00%	160.9	1.00
Cocktail	-	*21.4*	*-47.16%*	*22.3*	*-46.14%*	*240.4*	*1.49*
Student Parallelism-12L1S	12L1024D1S (109 M)	**11.8**	**-70.86%**	**12.6**	**-69.57%**	**621.5**	**3.86**
Student Parallelism-12L3S	12L1024D3S (327 M)	**12.1**	**-70.12%**	**12.8**	**-69.08%**	226.5	1.41

Table 2: Experimental results on inference efficiency. * denotes the teacher model with full depth and width; the suffix -pLqDrS denotes models with p layers, a hidden dimension of q, and r students; results in bold have the best efficiency among all compared; a result in *italic* indicates that the method fails to satisfy the target prediction quality.

increases with the number of layers, the expected latency of all models with over 6 layers has already exceeded the overall time budget. Other baselines like BERT-2L, PKD-BERT, TinyBERT, DistilBERT, and DynaBERT are within the time budget, but they have to wait longer to collect data batches. I-BERT suffers from the original computation complexity as BERT-base and small batch sizes from online workload, thus having higher latency. Because different samples exit from different layers, DeeBERT has the largest difference between the average and tail latency. The latency of Cocktail is determined by the largest model like 24-layer BERT-large to approach the accuracy of DeBERTa-xlarge.

4.2.3 Throughput. In information retrieval tasks such as text similarity search, hundreds of document candidates should be scored to obtain the most relevant ones to the query. Hence, we compare the throughput per GPU of best-pruned and single-student Student Parallelism and various baselines. We set the batch size as 64 for all baselines. The last column of Table 2 shows the throughput per GPU for all methods on the MRPC dataset, where the single-student Student Parallelism, trained with adaptive student pruning, improves throughput over baselines by up to 22.27× while maintaining prediction accuracy (§ 4.2.1). Meanwhile, even the best-pruned Student Parallelism (with 3 students) achieves comparable throughput with competitive baselines like TinyBERT with higher prediction accuracy. Even with fast integer operation, the model width and depth of I-BERT limits its throughput improvement to 3×. The vanilla BERT-2L256D has high throughout but is far from the target prediction quality. Cocktail suffers from low throughput per GPU since it has to use multiple GPUs for its models.

4.3 Ablation Studies

We further conduct ablation studies, including hyper-parameter analysis, accuracy analysis, and latency analysis.

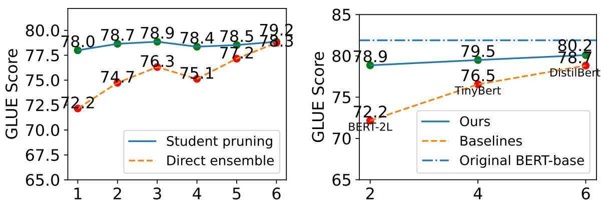
4.3.1 Hyper-parameter Analysis. We analyze two key hyper-parameters: the number of students and model depth, to show their impact on the prediction accuracy. Results in Figures 6(a) and 6(b) show that adaptive student pruning can improve the generalization of the remaining students and reduce the student number with minimal accuracy loss. Furthermore, we can further improve prediction accuracy with larger model depth with relaxed latency requirements.

Figure 6(a) shows that adaptive student pruning consistently outperforms direct boosting ensemble with the same number of students in terms of prediction accuracy, showing that adaptive student pruning effectively improves generalization. Due to the improved generalization of the remaining students, 3 students are sufficient to get the highest accuracy in most datasets (i.e. best-pruned). Even with only one remaining student, Student Parallelism can maintain a relatively high score. However, when we directly train the boosting ensemble without pruning, it is difficult for the limited number of students to the optimal solution.

As shown in Figure 6(b), Student Parallelism can further improve prediction accuracy with deeper students. With 4-layer students instead of 2-layer ones, the overall GLUE score improves from 78.9 to 79.5. Furthermore, Student Parallelism achieves a similar GLUE score to the original BERT-base with 6-layer students. Compared with other optimized baselines with the same number of layers (such as BERT-2L256D, TinyBERT, and DistilBERT), Student Parallelism

Ablation Variants	MRPC (F1)
Student Parallelism	86.9
w/o pruning	86.8
w/o stacking distillation	86.0
replacing boosting with bagging	85.6

Table 3: Ablation experiment results on prediction accuracy.



(a) Number of Students (b) Student Model Depth

Figure 6: Analysis of the impact of hyper-parameters on prediction accuracy.

Ablation Variants	avg (ms)	tail (ms)
Student Parallelism (3-layer students)	2.8	3.6
Student Parallelism (4-layer students)	4.5	5.2
with padding	5.1	5.3
with waiting queue	7.5	9.5

Table 4: Ablation experiment results on inference latency.

consistently outperforms them in terms of the overall GLUE score while having similar inference latency.

4.3.2 Accuracy Analysis. We first conduct ablation experiments on prediction accuracy to verify the effectiveness of our training methods on the MRPC dataset. Results are shown in Table 3. When we directly use all trained students without adaptive student pruning, we observe that the F1 score decreases a little from 86.9 to 86.8. After removing the stacking distillation, the F1 score further drops to 86.0, showing the effectiveness of the virtual stacking in improving model capacity. We further replace boosting ensemble with bagging ensemble, and observe a further decrease of F1 score to 85.6, showing that boosting more effective in ensembling small and shallow students in Student Parallelism.

4.3.3 Latency Analysis. We perform ablation experiments to analyze the impact of our designs on inference latency. Results in Table 4 show that latency increases by over 40% when we increase the depth of student models from 3 to 4 layers, showing the necessity to reduce layer-wise sequential computation. After we pad all data samples to the maximum length, the average latency gets close to tail latency. It shows that as Student Parallelism require less padding, the inference costs for shorter data samples are reduced. We further employ waiting queues to batch enough inference samples. The increase in both average and tail latency verifies the effectiveness of Student Parallelism in reducing the overhead of batching.

5 Discussion

Extention to Generative Models. Generative models like GPT [6] and LLaMA [14] generate tokens step by step, each of which involves classification of the next token. Therefore, our work can be naturally extended to speed up token classification for generative models. However, we do not conduct such experiments for three main reasons. First, evaluating the quality of open-ended generations is complex, and thus harder to verify if it maintains prediction quality. Second, we did not find 2-layer pre-trained GPT like BERT-2L [46] to initialize the generative students. We cannot afford to collect large datasets and pretrain LLMs neither. Finally and most importantly, we believe that speeding up BERT is non-trivial. They have been widely adopted by online services related to text mining and information retrieval, which will remain important in tasks like RAG [28] to complement large generative models.

Training Cost. Student Parallelism has a similar training cost to other model compression methods (e.g. 1.35× as DynaBERT with comparable accuracy). Although we train multiple students, each student is much smaller (e.g., 25% depth and 50% width) compared with other baselines. Each student is only trained on subsets of data (e.g., 20%) based on the residual error in the boosting ensemble. Furthermore, training of multiple students can be easily accelerated by multiple GPUs. For example, we can use one GPU to train the current student and distribute the inference of previous students on different GPUs. Most importantly, online services like search engines have much more inference samples (e.g., billions per day) than training, which amortizes the training costs.

6 Conclusion and Future Work

We propose Student Parallelism for efficient GPU inference of BERT-like models. It distills the large BERT-like teacher model into a group of small and homogeneous student models with fewer layers to achieve low inference latency. We train the group of students with boosting ensemble and virtual stacking distillation to maintain the prediction accuracy. We perform adaptive student pruning to reduce the number of students, thus achieving better generalization and dynamic adjustment of the number of students upon workload bursts. Finally, we make specialized designs for Student Parallelism to minimize the overhead of batching and padding. Comprehensive experiments with real-world online workloads verify the effectiveness of Student Parallelism in terms of accuracy, latency, and throughput. We plan to extend it to generative models like GPT for future work.

Acknowledgment

We thank the anonymous reviewers for their constructive comments. We thank Dr. Lu Hou for all the insightful discussions and suggestions for this paper. This work is supported in part by the Hong Kong RGC TRS T41-603/20R, the National Natural Science Foundation of China (62441220), and the Guangdong Basic and Applied Basic Research Foundation (2024A1515011667).

References

[1] Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016.
[2] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. Batch: Machine learning inference serving on serverless platforms with adaptive batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.
[3] Matteo Aquilina, Eric Budish, and Peter O'neill. Quantifying the high-frequency trading "arms race". *The Quarterly Journal of Economics*, 137(1):493–564, 2022.
[4] Umar Asif, Jianbin Tang, and Stefan Harrer. Ensemble knowledge distillation for learning improved and efficient networks. In *ECAI 2020*, pages 953–960. IOS Press, 2020.
[5] Microsoft Azure. Machine learning inference during deployment. https://github.com/triton-inference-server/server.
[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[8] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, 2017*, pages 613–627. USENIX Association, 2017.

[9] Weihao Cui, Han Zhao, Quan Chen, Hao Wei, Zirui Li, Deze Zeng, Chao Li, and Minyi Guo. Dvabatch: Diversity-aware multi-entry multi-exit batching for efficient processing of DNN services on gpus. In Jiri Schindler and Noa Zilberman, editors, *2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 183–198. USENIX Association, 2022.

[10] Don Dennis, Abhishek Shetty, Anish Prasad Sevekari, Kazuhito Koishida, and Virginia Smith. Progressive ensemble distillation: Building ensembles for efficient inference. *Advances in Neural Information Processing Systems*, pages 43525–43543, 2023.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Pratik Fegade, Tianqi Chen, Phillip B. Gibbons, and Todd C. Mowry. The cora tensor compiler: Compilation for ragged tensors with minimal padding. In *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022.

[13] Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. Compressing BERT: studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2020, Online, July 9, 2020*, pages 143–155. Association for Computational Linguistics, 2020.

[14] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[15] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. Cocktail: A multidimensional optimization for model serving in cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1041–1057, Renton, WA, April 2022. USENIX Association.

[16] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R Das. Cocktail: A multidimensional optimization for model serving in cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1041–1057, 2022.

[17] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2021.

[18] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793, 2020.

[19] Furong Huang, Jordan Ash, John Langford, and Robert Schapire. Learning deep resnet blocks sequentially using boosting theory. In *International Conference on Machine Learning*, pages 2058–2067. PMLR, 2018.

[20] Sylvain Jeaugey. Nccl 2.0. In *GPU Technology Conference (GTC)*, volume 2, 2017.

[21] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[22] Masahiro Kaneko, Masato Mita, Shun Kiyono, Jun Suzuki, and Kentaro Inui. Encoder-decoder models can benefit from pre-trained masked language models in grammatical error correction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*.

[23] Abeer Abdel Khaleq and Ilkyeun Ra. Cloud-based disaster management as a service: A microservice approach for hurricane twitter data analysis. In *2018 IEEE Global Humanitarian Technology Conference (GHTC)*, pages 1–8. IEEE, 2018.

[24] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

[25] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 784–794, 2022.

[26] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[28] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[29] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. Fastbert: a self-distilling BERT with adaptive inference time. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020*, pages 6035–6044. Association for Computational Linguistics, 2020.

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

[31] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pages 6682–6691. PMLR, 2020.

[32] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. *Advances in neural information processing systems*, 12, 1999.

[33] Llew Mason, Jonathan Baxter, Peter L Bartlett, Marcus Frean, et al. Functional gradient techniques for combining hypotheses. *Advances in Neural Information Processing Systems*, pages 221–246, 1999.

[34] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[35] Nvidia. Multi-process service. https://docs.nvidia.com/deploy/mps/index.html.

[36] Automatic Differentiation In Pytorch. Pytorch, 2018.

[37] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020.

[38] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. Infaas: Automated model-less inference serving. In Irina Calciu and Geoff Kuenning, editors, *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 397–411. USENIX Association, 2021.

[39] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[40] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 2017*.

[41] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020.

[42] Liyuan Sun, Jianping Gou, Baosheng Yu, Lan Du, and Dacheng Tao. Collaborative teacher-student learning via multiple knowledge transfer. *arXiv preprint arXiv:2101.08471*, 2021.

[43] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

[44] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

[45] Hanlin Tang, Xipeng Zhang, Kai Liu, Jianchen Zhu, and Zhanhui Kang. Mkq-bert: Quantized bert with 4-bits weights and activations. *arXiv preprint arXiv:2203.13483*, 2022.

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

[47] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

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

[49] Guanhua Wang, Zhuang Liu, Brandon Hsieh, Siyuan Zhuang, Joseph Gonzalez, Trevor Darrell, and Ion Stoica. sensai: Convnets decomposition via class parallelism for fast inference on live data. *Proceedings of Machine Learning and Systems*, 3:664–679, 2021.

[50] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[51] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2246–2251. Association for Computational Linguistics, 2020.

[52] Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. Nas-bert: task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1933–1943, 2021.

[53] Ze Yang, Linjun Shou, Ming Gong, Wutao Lin, and Daxin Jiang. Model compression with two-stage multi-teacher knowledge distillation for web question answering system. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 690–698, 2020.

[54] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE, 2019.

[55] Alaa Zain, Yang Jian, and Jinjia Zhou. Collaborative multiple-student single-teacher for online learning. In *Artificial Neural Networks and Machine Learning - ICANN 2022 - 31st International Conference on Artificial Neural Networks, Bristol, UK, September 6-9, 2022, Proceedings, Part I*, Lecture Notes in Computer Science.

[56] Yujia Zhai, Chengquan Jiang, Leyuan Wang, Xiaoying Jia, Shang Zhang, Zizhong Chen, Xin Liu, and Yibo Zhu. Bytetransformer: A high-performance transformer boosted for variable-length inputs. *arXiv preprint arXiv:2210.03052*, 2022.

[57] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. {MArk}: Exploiting cloud services for {Cost-Effective},{SLO-Aware} machine learning inference serving. In *USENIX Annual Technical Conference (ATC)*, pages 1049–1062, 2019.

[58] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.

[59] Xiatian Zhu, Shaogang Gong, et al. Knowledge distillation by on-the-fly native ensemble. *Advances in neural information processing systems*, 31, 2018.

[60] Simiao Zuo, Qingru Zhang, Chen Liang, Pengcheng He, Tuo Zhao, and Weizhu Chen. Moebert: from BERT to mixture-of-experts via importance-guided adaptation. In *NAACL*, pages 1610–1623, 2022.

A Theory Analysis

In this section, we first conduct the convergence analysis on Student Parallelism. Then we make the quantitative analysis and comparison of the inference efficiency.

A.1 Convergence Analysis

To prove the convergence, we can first prove our sequential training of students is one special case of Anyboost [32, 33], then we can reuse the convergence properties and proofs of Anyboost. Specifically, we take a non-parametric approach to view our sequential training of students as the numerical optimization problem in function space. And we consider $B(x))$ evaluate at each point x as a "parameter" and seek to minimize the following MSE error:

$$L(T(x), B(x))) = \frac{1}{2}|T(x) - B(x))|^2, \tag{8}$$

to make the ensemble of students generates a similar final representation with the original teacher $T(x)$. In function space, there are an infinite number of such parameters, but in data sets only a finite number $\{B(x)_i)\}_1^N$ are involved. Following the numerical optimization paradigm we take the solution to be

$$F^*(x) = \sum_{i=0}^{M} f^{(i)}(x), \tag{9}$$

where $f^{(0)}(x)$ is the initial student $S^{(0)}(x)$ directly mimics the teacher, and any $f^{(i!=0)}(x)$ is the incremental function ("step" or "boost") defined by the optimization method as follows:

$$f^{(i)}(x) = -\alpha_i g^{(i)}(x). \tag{10}$$

And $g^{(i)}(x)$ is the gradient decent direction of $L(\cdot)$, namely:

$$g^{(i)}(x) = T(x) - B^{(i-1)}(x) \tag{11}$$

Therefore, our main loss L_{boost} just makes the added i-th student $S^{(i)}(x)$ fit the $g_i(x)$:

$$L_{\text{boost}}(x) = \frac{1}{2}\left\|T(x) - B^{(i-1)}(x) - S^{(i)}(x)\right\|^2 = \frac{1}{2}\left\|g^{(i)}(x) - S^{(i)}(x)\right\|^2 \tag{12}$$

Although we apply other regularization like stacking distillation, the final representation of the student is only related with the above loss. Since the added i-th student $S^{(i)}(x)$ is trained to fit $g^{(i)}(x)$ in the final output and α_i is determined by the line serach, the sequential training of students in Student Parallelism belongs to Anyboost [32, 33]. Therefore, the boosting ensemble of students to mimic the teacher should share a similar convergence theorem with Anyboost as follows:

Theorem A.1. *The MSE loss in Student Parallelism has the lower bound 0, and it is Lipschitz differentiable loss function (for any $L > 1$, we always have $|\nabla L_{boost}(x_1) - \nabla L_{boost}(x_1)| < L|x_1 - x_2|$). Let $F^{(0)}, F^{(1)}, \ldots$ be the sequence of combined hypotheses generated by the Student Parallelism training algorithm, using small enough step-sizes $\alpha_i := -\frac{\langle \nabla L_{boost}(T, B^{(i-1)}), S^{(i)} \rangle}{L|S^{(i)}|^2}$. Then our sequential training of students either halts on round T with $-\langle \nabla L_{boost}(T, F^{(i-1)}), S^{(i)} \rangle \leq 0$, or $L_{boost}(F^{(i)})$ converges to some finite value L^*, in which case*

$$\lim_{i} \langle \nabla L_{boost}(T, B^{(t)}), S^{(t+1)} \rangle = 0. \tag{13}$$

Because the detailed proof can be found in the previous theory work [32, 33], we skip the proof for the convergence theorem.

A.2 Efficiency Analysis for GPU Inference

To make the quantitative analysis and comparison, we first build a performance model for the inference latency and throughput. In this model, we consider all model and input factors, including model depth D, model width W, batch size B, and input length N. Moreover, we also consider the time cost of the waiting queue $Q(B)$ that collect the maximum B samples, and the parallel computation capability C, the PCI-E transferring bandwidth T from the host to the GPU, the GPU number G, and the total parallel model or student group number M on the cluster. Therefore, we can model the inference latency of BERT-like models as follows:

$$\text{Latency} \propto D\left\lceil \frac{WBN^2M}{CG} \right\rceil + Q(B) + \frac{BN}{T} \tag{14}$$

The first term is the inference computation time on the GPU, in which $\left\lceil \frac{WBN^2M}{CG} \right\rceil$ is computation time of one layer and the ceiling symbol means it is constant if there is enough parallel computation capability C. And the second term $Q(B)$ stands for the waiting time of collecting the data batch, which depends on the workload and batch size B. The last term means the PCI-E transferring time of the data size BM. Furthermore, we can further model the throughput per GPU based on the inference latency as follows:

$$\text{Throughput Per GPU} = \frac{BM}{\text{Latency} \cdot G} \tag{15}$$

.

To analyze the efficiency, we then make quantitative comparisons between Student Parallelism and some representative baselines in optimizing the BERT-base on the MRPC dataset. As all the factors shown in Table 5, Student Parallelism can outperform the other baselines due to its model architecture and system designs. Specifically, Student Parallelism can achieve the lowest model depth of only 2 layers to significantly decreases the latency. Although the overall model width of 3 students brings in more parallel operators, the GPUs have enough parallel computation capability to speed them up. Because Student Parallelism realizes the direct inference on different individual samples, there is no waiting Q and large batch B for PCI-E transferring. And the single sample and small batches of 2~4 samples in the same bin are in the similar length. In terms of throughput, the multiple students sharing the same GPU make the M can be several times larger than G, leading to large enough total concurrent sample number BM to fully utilize the GPUs. Additionally, adaptive student pruning can reduce the student number to only 1 to improve the student group number M.

In contrast, the other baselines suffer from the model architecture unfriendly to GPUs and extra costs for data parallelism. The knowledge distillation baseline TinyBERT has 4 layers fewer than any other baselines, but it is 2 times larger than Student Parallelism. In adaptive computing baseline DeeBERT, some "hard" samples still have to go through all layers. Because Cocktail is the bagging ensemble of different models, its model depth is determined by the straggler that have the largest depth $\max(D_i)$. Although the pruning baseline DynaBERT have the smallest model width of 192 hidden dimensions in all methods, it mainly reduces the parallel operators that can be efficiently accelerated by GPUs. All baselines need large enough data batches for data parallelism, bringing in considerable waiting queue costs Q and large PCI-E transferring time. Traditionally, they have to pad all short samples to have the same maximum length N as 128. Otherwise, they need to suffer from extra compiling costs or computation divergence for the ragged batch. Based on data parallelism, all the baselines only run one model on every GPU to process different data batches, resulting in $M = G$.

B Implementation Details

B.1 Motivation Experiment Settings

By default in Fig. 1, the model is a 12-layer BERT-base with 768 hidden dimensions, and the data batch consists of 8 samples with length of 256. In every sub-figure, we choose one as the changing variable and fix the others to measure the latency and throughput and show the influences of the changing variable.

B.2 Testbed and Environment

We use 4 servers each with 4 NVIDIA Geforce 3090 GPUs, Intel Xeon(R) E5-268340 CPU, and 96 GB memory for all evaluations. And it runs on the Ubuntu 18.04 operating system. Besides the global dispatcher, every server has its own logical controller. And we implement the Student Parallelism with Python and packages like PyTorch [36], HuggingFace Transformers [50], and NCCL [20].

B.3 Implementation Details

B.3.1 Training details. We consider all 24 compact BERT variants [46] and all DEBERTA variants as the student candidates.

	Depth	Width	Batch Size	Length	M	Q
BERT-base	12	768	~10	128	G	Yes
TinyBERT	4	312	~10	128	G	Yes
DynaBERT	6	192	~10	128	G	Yes
DeeBERT	1-12	768	~10	128	G	Yes
Cocktail	$\max(D_i)$	$\sum_i W_i$	~10	128	G	Yes
Student Parallelism	2	(1-3)*256	≤ 4	8(n%8+1)	4G	None

Table 5: The comparison on all factors about GPU efficiency.

In practice, we use the BERT-2L256D [46] for BERT-base, BERT-6L768D for BERT-large, and Deberta-12L768D for DEBERTA-xlarge as the initialization of student models respectively. Following previous works [21, 51], we conduct the same data augmentation on the datasets. We conduct a grid search on various training hyperparameters to find the best for every dataset, including epoch number, batch size, learning rate, λ weights, and data sampling ratio. In practice, we optimize the α_i together with the student, since they are both differentiable. The training epoch number is early stopped by the validation loss. lr=5e-5 and betas=(0.9, 0.999) for Adam optimizer, $\lambda = 1$ works fine in most cases, data sampling ratio is usually 0.2 (smaller in some large datasets). The student number is determined by the validation loss. When conducting student distillation, we continue to add new students until the validation loss does not decrease any more. Then we conduct adaptive student pruning and measure the validation loss to decide the number of students to maintain. To determine the student model depth, we conduct the binary search from one layer to 25% layers of the teacher model, to find the smallest model depth that can reach the target accuracy.

B.3.2 Student Allocation. In the initialization, resource management conducts profiling and grid searching to find the optimal student number sharing the same GPU and the maximum batch size of each student group (used in § B.3.3). Both hyperparameters are set to maximize the total throughput without increasing latency due to lacking computation capability. During the inference, it dynamically adjusts the student number to host different numbers of student groups according to the workload intensity and resource usage. It keeps tracking the size of the length-aware buffer (described in § B.3.3) and the idle group number as indicators of workload intensity and resource usage. If the buffer size reaches the threshold (i.e. the total student group number), Student Parallelism drops one last-trained student models to start more groups so that the waiting time of buffered samples will not exceed the inference time. And if the buffer is empty and the total student number of all idle groups is larger than the occupied group number for a certain time window (e.g., 2 minutes), it means there are enough free resources to increase the student number by one for all the occupied student groups. Then Student Parallelism can make better use of idle computation resources to improve accuracy.

B.3.3 the Length-aware Buffer. Figure 5 shows how the length-aware buffer efficiently generates small batches of samples in similar lengths on the fly. We equally split the full-length range (i.e., 0-128) into 16 bins at every step of 8 tokens. Then we set the bins and pointer as the key-value pair in the dictionary to quickly index samples of different length bins. When a new sample arrives, we check the dictionary to see if any sample belongs to the same length bin in the buffer. If so and the element size is not the maximum (e.g., 4, which is also determined by profiling), we can get the element pointer from the dictionary to merge them as one buffer element. Otherwise, we append the new sample as the tail element in the buffer and update its length bin and pointer in the dictionary. When the first buffer element is dispatched, we delete its key-value pair in the dictionary. Obviously, all the operators of the length-aware buffer only has $O(1)$ time complexity due to the hash map.