Enhanced TreeBERT: High-Performance, Computationally Efficient Multi-Task Model

Stanford CS224N Default Project
Mentor: Hamza El Boudali

Thanawan Atchariyachanvanit
Department of Computer Science
Stanford University
`thanawan@stanford.edu`

Pann Sripitak
Department of Computer Science
Stanford University
`pannsr@stanford.edu`

Abstract

Transfer learning has emerged as a valuable tool for addressing a wide range of downstream NLP tasks using a single pre-trained model (Zhuang et al., 2020). However, fully harnessing the potential of transfer learning in a multi-task setting remains a challenge. In this project, we address this challenge by comparing various multi-task model configurations and multi-task fine-tuning techniques to identify the best-performing and the most computationally efficient multi-task BERT models across three different downstream tasks. We further improve these models through non-architectural approaches. Our top-performing model, composed of three individually fine-tuned BERT models, totals 328M trainable parameters and achieves an overall test score of 0.791. Conversely, our most computationally efficient model with only one underlying BERT and 29M trainable parameters achieves an overall test score of 0.765 with multi-task learning. These results are attributed to the multi-task BERT model configuration, the SMART framework, and the lightweight input preprocessing.

1 Introduction

When addressing NLP tasks in low-resource target domains, transfer learning emerges as a valuable approach by leveraging data from high-resource domains (Zhuang et al., 2020). Within this transfer learning framework, the Bidirectional Encoder Representations from Transformers (BERT) achieved the state-of-art performance, primarily due to fine-tuning deep bidirectional pretrained representations on each individual downstream task (Devlin et al., 2019). However, within a multi-task setting, addressing multiple tasks in isolation with one BERT model per task can be computationally expensive in terms of both time and space. Moreover, this approach may overlook potential benefits derived from multi-task learning.

In this paper, we explore the multi-task setting with the main objective of constructing 1.) the best-performing (BESTMODEL) and 2.) the most computationally efficient (EFFICIENTMODEL) multi-task BERT models for three distinct downstream tasks: sentiment analysis (SST), paraphrase detection (QQP), and semantic textual similarity (STS). To achieve this goal, we compare various multi-task model configurations and multi-task fine-tuning techniques. Furthermore, we investigate the global effectiveness of Jiang et al. (2020)'s SMART framework across all tasks, as well as the local improvement achieved by implementing lightweight input preprocessing, varying the loss function, and exploring sentence-pair encoding methods for certain tasks.

Our BESTMODEL comprises three individually fine-tuned BERT models, totaling 328M trainable parameters, while our EFFICIENTMODEL is a single BERT model with the last 4 encoder layers unfrozen and 29M trainable parameters. Through the utilization of the SMART framework and lightweight SST input preprocessing, they attain overall test scores of 0.791 and 0.765, respectively.

2 Related Work

While the original BERT paper (Devlin et al., 2019) demonstrated BERT's capability in achieving state-of-the-art performance across eleven NLP tasks, it did not explore its effectiveness as a multi-task learning model. This aspect is crucial for comprehending BERT's versatility and potential in real-world applications spanning various NLP domains. Liu et al. (2019) filled this research gap by presenting a Multi-Task Deep Neural Network (MT-DNN) with the $BERT_{LARGE}$ model incorporated. They found that MT-DNN consistently outperformed the BERT baseline, showing BERT's applicability and high performance as a multi-task learning model. They believe that it is because MT-DNN can leverage large amounts of cross-task data and benefit from a regularization effect which leads to more general representations to help adapt to new tasks and domains. We extend our contribution by exploring efficient model configurations in addition to the optimal one, all while adhering to the more limited constraint of utilizing only $BERT_{BASE}$.

Another area of research concerning fine-tuning BERT models for multi-task settings addresses overfitting and excessive updates during the fine-tuning phase. To address this challenge, Jiang et al. (2020) introduced the SMART framework, which integrates smooth-inducing adversarial regularization into the loss function. Specifically, they added small random perturbations to each embedded input and penalized the model for any resulting changes in output. They also proposed a class of Bregman proximal point optimization methods to optimize their new training objective.

Furthermore, efforts were made to address the limitations of BERT in single-task settings, which also pose challenges when transitioning to a multi-task setting. In the sentence-pair tasks, the original BERT architecture takes the concatenation of a sentence pair as its input. Therefore, no independent sentence embeddings are computed, making it difficult to derive sentence embeddings from BERT alone. To address this limitation, Reimers and Gurevych (2019) introduced Sentence-BERT (SBERT), a modification of the BERT network that integrates siamese and triplet networks. This enhancement enables us to derive semantically meaningful sentence embeddings, which can be compared using cosine similarity.

3 Approach

The foundation of our research lies in minBERT, a minimal version of BERT described in the Default Project handout, along with the utilization of pretrained $BERT_{BASE}$ weights from Huggingface. We leverage the concept of transfer learning to tailor minBERT to tackle three downstream tasks: sentiment analysis (SST), paraphrase detection (QQP), and semantic textual similarity (STS).

Subsequently, with the objective of constructing the BESTMODEL and the EFFICIENTMODEL, we begin by exploring various multi-task model configurations and multi-task fine-tuning methods. Then, we adopt a combination of two approaches to enhance performance: global enhancement and local enhancement. Our global enhancement strategies aim to improve the overall multi-task performance by leveraging techniques like Jiang et al. (2020)'s SMART framework. Conversely, our local enhancement strategies focus on enhancing task-specific performance. This includes performing lightweight input preprocessing for SST, adjusting loss functions for STS, and investigating sentence-pair encoding methods for STS and QQP. Finally, we apply the subset of the methods mentioned that are proven to be advantageous to our multi-task models.

Baseline. For our baseline model, we integrate three trainable *minimal* heads (one dropout and one linear layer) onto the pretrained minBERT architecture introduced in the Default Project handout with each head dedicated to one of the tasks (See A.2). Other configurations adhere to the default settings described in Section 4.3. This baseline model establishes the initial performance benchmark of the fixed pretrained minBERT, demonstrating the inherent difficulty of tasks for the BERT architecture.

Preliminary Study: Dropout Rate in Head Dropout Layers. Prior to conducting our main experiment, we conduct a preliminary exploration to determine the default hidden dropout rate within the dropout layer of the heads. We try dropout rates of 0.1, 0.3, 0.5, 0.7, and 0.9 on a fully fine-tuned single-task BERT model for both the SST and STS tasks.[1] We then select the optimal dropout rate to be used for the main experiment.

[1]We did not conduct experiments on the QQP task due to computational and time limitations.

Main Study I: Multi-task Model Configuration. We primarily focus on two types of multi-task models (See A.1). *Both models were implemented by us using the provided MultitaskBERT class.*

- The TripleBERT model comprises three separate $BERT$ models, each equipped with a dedicated minimal head for executing a downstream task.[2]
- The TreeBERT model consists of a single BERT model and three heads, where the weights of the BERT model are shared across all tasks.

We are interested in these two configurations because, while a single BERT model has proven to perform exceptionally in various single tasks, fine-tuning one BERT model per task in a multi-task setting is computationally expensive. Moreover, each task's input in a multi-task setting is the same in some scenarios, prompting us to process the input through a BERT model only once to save inference time. Hence, we explore fine-tuning a single BERT to be used for all tasks. While a reduction in parameters might restrict the performance of this compact model, we posit that implicit knowledge sharing during training in multi-task learning can compensate for it.

Main Study II: Fine-Tuning Methods for TreeBERT. Traditionally, fine-tuning involves training all layers of a BERT model to achieve optimal performance. We instead propose a different approach with our TreeBERT model, where we unfreeze only the last k encoder layers of BERT and fine-tune them on multiple tasks. We expect this approach to better optimize the TreeBERT model towards downstream tasks compared to our baseline model, where all BERT layers are frozen. We also anticipate that this approach will conserve computational time and cost while maintaining, if not improving, performance levels compared to TripleBERT by leveraging the benefits of out-of-domain data. *Note that we implemented all the fine-tuning frameworks ourselves. While inspired by existing methods, to the best of our knowledge, these methods are novel within this specific context.*

Global Enhancement Approach: SMART Framework. Considering the potential risks of overfitting and aggressive updating during the fine-tuning stage, we investigate Jiang et al. (2020)'s smooth-inducing adversarial regularization as a means to address the complexity of the model. Jiang et al. suggest minimizing $\mathcal{L}(\theta) + \lambda_S \mathcal{R}_S(\theta)$ during fine-tuning, where $\mathcal{L}(\theta)$ is the typical loss, λ_S is a tuning parameter, and $\mathcal{R}_S(\theta)$ is the smoothness-inducing adversarial regularizer defined as $\mathcal{R}_S(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{||\tilde{x}_i - x_i||_p \leq \epsilon} l_s(f(\tilde{x}_i;\theta), f(x_i;\theta))$, where $\epsilon > 0$ is a tuning parameter, and $\tilde{x}$ is a sampled embedded input perturbation. Note that l_s is selected to be the symmetrized KL-divergence for the classification task and the squared loss for the regression task. By minimizing this objective, function f is encouraged to exhibit smoothness within the neighborhood of all x_i's. This property proves particularly beneficial in mitigating overfitting and enhancing generalization. However, rather than adopting Jiang et al. (2020)'s Bregman proximal point optimization, we utilize the ADAMW optimizer implemented as outlined in the Default Project handout to optimize the new objective function. *In terms of implementation, we integrated the available smart-pytorch[3] into our codebase.*

Local Enhancement Approach I: SST Lightweight Input Preprocessing. After establishing the main architecture, we proceed to enhance the performance of each task-specific head. For the sentiment analysis head, we explore lightweight input preprocessing in an attempt to improve the SST accuracy. We discovered placeholders like '-lrb-' and '-rrb-', which represent '(' and ')' respectively, are not tokenized appropriately by the BERT tokenizer. Additionally, we noticed that the tokenizer treats "don't" and "do n't" in the datasets differently. Thus, we standardize them by making the following replacements: ["" ", """],[" "", """], [" n't", "n't"], ["-lrb- ", "("] , [" -rrb-", ")"] and ["\/", "/"]. *We made these observations without any external guidance.*

Local Enhancement Approach II: STS Loss Function. Zhuang and Chang (2017) introduce the Pearson Correlation Coefficient loss and argue that it reflects invariance to changes in location and scale of the expected STS score, unlike other training objectives such as mean squared error (MSE). We decide to explore its application. *We implemented this loss ourselves using torch.corrcoef.*

$$\mathcal{L}_{PCC} = -\frac{\sum_{n=1}^{N}(y^n - \overline{y})(\hat{y}^n - \overline{\hat{y}})}{\sqrt{\sum_{n=1}^{N}(y^n - \overline{y})^2}\sqrt{\sum_{n=1}^{N}(\hat{y}^n - \overline{\hat{y}})^2}}$$

where y^n is the predicted expected score, and $\hat{y}^n$ is the annotated gold score.

[2]When we mention TripleBERT, we are referring to its fully fine-tuned variant, unless otherwise specified.

[3]`https://github.com/archinetai/smart-pytorch`

Local Enhancement Approach III: QQP & STS Sentence-Pair Encoding Method We investigate two approaches for encoding sentence-pair inputs introduced in the original BERT model (Devlin et al., 2019) and the SBERT model (Reimers and Gurevych, 2019). *We implemented both ourselves in the QQP and STS heads.*

- Cross-Encoder Method: In this approach, the BERT model processes the concatenation of the two sentences in the following format: [CLS] sentence$_1$ [SEP] sentence$_2$ [SEP]. The resulting BERT embedding is then forwarded through the task head as usual.
- Bi-encoder Method: In this approach, the BERT model processes one sentence at a time, generating the corresponding output embeddings u and v. Then, for the QQP head, we concatenate u, $|u - v|$ and v and pass them through a dropout and linear layer. For the STS head, we pass $\cos(u, v)$ to the ReLU activation layer and multiply the resulting value by 5 to transform the cosine value into a logit between 0 and 5. Finally, for both heads, we consider two types of pooling strategies: CLS pooling and MEAN pooling.

Final Step: Ensembling & Comparing Methods and Models. Based on these findings, we apply the advantageous global and local enhancements to the selected BESTMODEL and EFFICIENTMODEL candidates and compare the performance of all the model candidates.

4 Experiments

4.1 Data

As explained in the Default Project handout, we perform sentiment analysis on the Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013). The SST dataset consists of 11,855 single sentences from movie reviews with a label of negative, somewhat negative, neutral, somewhat positive, or positive represented by integers 0 to 4. We use the Quora dataset[4] for paraphrase detection, which has 400,000 question pairs with labels 0.0 and 1.0 indicating whether particular instances are paraphrases of one another. We use the SemEval STS Benchmark dataset (Agirre et al., 2013) for semantic textual analysis. The dataset consists of 8,628 different sentence pairs of varying similarity on a continuous scale from 0 (unrelated) to 5 (equivalent meaning).

4.2 Evaluation method

Aligned with the provided leaderboard metrics, we evaluate model performance using accuracy for SST and QQP (Acc_{SST} & Acc_{QQP}), and Pearson correlation for STS (Cor_{STS}). The overall score is computed as follows: $\frac{(Acc_{SST}+Acc_{QQP}+0.5\cdot Cor_{STS}+0.5)}{3}$. We also track the number of total and trainable parameters to evaluate the trade-off between performance and computational cost.[5]

4.3 Experimental details

The following is our default setup unless a variable is the independent variable for a particular experiment. Our setup follows the default hyperparameter values provided in the codebase (e.g. $epochs$ = 10, $batch\_size$ = 8, $lr = 1e^{-5}$, $hidden\_dropout\_prob$ = 0.3). The only exception is the baseline model, which uses $lr = 1e^{-3}$. Additionally, we introduce early stopping with a patience of 5. Regarding the architecture, our default BERT pooling strategy is CLS pooling, and our default sentence-pair encoding method is the cross-encoder approach. For our training procedure, we employ different loss functions based on the task: cross entropy for SST, binary cross entropy for QQP, and mean squared error for STS. We utilize the ADAMW optimizer, implemented as per the guidelines provided in the Default Project handout. In terms of training steps, within each epoch, the model will be trained on one task at a time in the following order: SST, QQP, STS.[6]

[4]`https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`

[5]We initially tracked the fine-tuning time when running on a single GPU via Google Cloud Platform. However, upon rerunning it, we noticed a significant discrepancy, likely stemming from network traffic fluctuations.

[6]For the baseline model and TripleBERT variants, shuffling single-task batches is inconsequential since there are no shared trainable parameters. For TreeBERT with the k last encoder layers unfrozen, shared trainable parameters do exist, but we did not observe any improvement and sometimes encountered worse results when experimenting with shuffling single-task batches. Therefore, we opted not to perform shuffling. Moreover, the training task scheduling choices are beyond the scope of our project, so we did not delve deeply into this aspect.

4.4 Results

Preliminary Study Results: Dropout Rate in Head Dropout Layers

p_{dropout}	Acc_{SST}	Cor_{STS}
0.1	0.511	**0.880**
0.3	**0.518**	**0.875**
0.5	0.517	0.874
0.7	0.516	**0.875**
0.9	**0.522**	0.873

Table 1: Performance of a single-task fully fine-tuned BERT on SST and STS with different head dropout rates

Table 1 presents the performance of a single-task fully fine-tuned BERT model on the SST and STS tasks across various dropout rates (p_{dropout}). For the SST task, $p_{\text{dropout}} = 0.9$ produces the best SST model, while $p_{\text{dropout}} = 0.3$ yields the second-best. For the STS task, $p_{\text{dropout}} = 0.1$ results in the best model. Interestingly, both $p_{\text{dropout}} = 0.3$ and $p_{\text{dropout}} = 0.7$ produce the second-best. Conversely, $p_{\text{dropout}} = 0.9$ results in the worst-performing model. To strike a balance between the performance on the two tasks, we opt for $p_{\text{dropout}} = 0.3$ since it yields a model that performs relatively well on both tasks.

Main Study Results: Multi-task Model Configuration & Fine-Tuning Methods for TreeBERT

Table 2 displays the results of our model configuration experiments. The overall results align closely with our expectations. We anticipated that TripleBERT would perform best in all tasks, given it is fine-tuning towards specific tasks, and that it would require more time for fine-tuning due to its 328M trainable parameters. Specifically, it achieves the highest overall score of 0.781, making it our BESTMODEL candidate. Additionally, all variants of TreeBERT with some unfrozen layers outperform the baseline due to its increased flexibility as expected.

The results demonstrate a notable enhancement across all tasks when the last encoder layer is unfrozen, with overall performance increasing from 0.604 to 0.742 with 8M trainable parameters in total. Nevertheless, it appears that the marginal benefits of unfreezing the second layer (0.744), third layer (0.761), and fourth layer (0.763) are not as substantial, yet still represent an improvement. Additionally, we observe that the performance of the TreeBERT model begins to decline once $k > 4$. This outcome is expected, as the reduction in non-trainable parameters and the simultaneous increase in shared trainable parameters could potentially destabilize the model during joint training.

Considering the outcomes of varying k, we opt for $k = 4$ for our EFFICIENTMODEL candidate. This is motivated by the fact that they have more than ten times fewer trainable parameters, yet still achieve a commendable overall score of 0.763. This overall score stands higher by 0.159 compared to the baseline, while being only 0.018 lower than that of the fully fine-tuned TripleBERT.

Multi-task Model Configuration	Overall Score	Acc_{SST}	Acc_{QQP}	$Corr_{STS}$	# of Parameters (Trainable / Total)
Baseline	0.604	0.398	0.690	0.448	5,383 / 5,383
TripleBERT	**0.781**	**0.518**	**0.886**	**0.875**	**328M / 328M**
TreeBERT with $k = 1$	0.742	0.480	0.839	0.814	8M / 109M
TreeBERT with $k = 2$	0.744	0.493	0.820	0.836	15M / 109M
TreeBERT with $k = 3$	0.761	0.502	0.858	0.845	22M / 109M
TreeBERT with $k = 4$	**0.763**	**0.503**	0.863	0.848	**29M / 109M**
TreeBERT with $k = 5$	0.757	0.496	**0.881**	0.810	36M / 109M
TreeBERT with $k = 6$	0.732	0.404	0.859	**0.864**	43M / 109M

Table 2: Performance and efficiency of multi-task model under various configurations

Global Enhancement Results: SMART Framework

Table 3 shows the impact of adopting Jiang et al. (2020)'s smooth-inducing adversarial regularization with different weights λ_s on a single-task BERT model fully fine-tuned on the SST and STS tasks, while using the default hyperparameters of smart-pytorch[7]. Results reveal that $\lambda_s = 0.1$ yields the best performance on both tasks, with an increase of 0.11 in SST accuracy and 0.05 in STS correlation compared to when SMART is not applied. We believe that this regularization yields superior outcomes compared to dropout as it mitigates overfitting across the entire model rather than solely on the head.

λ_s	Acc_{SST}	$Corr_{STS}$
0	**0.518**	**0.875**
0.01	0.521	**0.880**
0.05	0.518	0.878
0.1	**0.529**	**0.880**
1.0	0.515	0.877

Table 3: Performance of a single-task fully fine-tuned BERT on SST and STS with different SMART weights λ_s

[7]`https://github.com/archinetai/smart-pytorch`

Local Enhancement I Results: SST Lightweight Input Preprocessing

SST IP	ACC_{SST}
✗	0.518
✓	**0.522**

Table 4: Effectiveness of SST input preprocessing (SST IP)

With lightweight input preprocessing, we observe a notable enhancement in the performance of a single-task fully fine-tuned BERT model for the SST task, with its performance increasing from 0.518 to 0.522 as shown in Table 4. This underscores the significance of standardizing the input format, as it might enable the pretrained model to leverage prior knowledge and learn more effectively.

Local Enhancement II Results: STS Loss Function

$\mathcal{L}_{STS}$	Cor_{STS}
MSE	**0.875**
Pearson	0.870

Table 5: Effectiveness of each STS loss function

Based on Table 5, it is evident that the Pearson Correlation Coefficient loss yields worse performance for a single-task fully fine-tuned BERT model on the STS task compared to the mean squared error loss, contrary to what Zhuang and Chang (2017) suggested. We attribute this unexpected result to the notion that the Pearson Correlation Coefficient loss may not be as widely generalizable as the more commonly used MSE loss function.

Local Enhancement III Results: QQP & STS Sentence-Pair Encoding Method

Encoding Method	Pooling Method	Acc_{QQP}	Cor_{STS}
Cross-enc	**CLS**	**0.886**	**0.875**
Bi-enc	CLS	0.852	0.756
Bi-enc	MEAN	0.880	0.872

Table 6: Effectiveness of each sentence-pair encoding method

Table 6 presents a comparison of the performance of a single-task fully fine-tuned BERT model using different sentence-pair encoding methods for the QQP and STS tasks. Our findings indicate that our default cross-encoder method outperforms the bi-encoder method on both tasks, irrespective of the pooling strategy employed. This aligns with our expectations, as cross-encoders are known to offer superior performance compared to bi-encoders, albeit at a higher computational cost. Additionally, the observation that bi-encoders with MEAN pooling exhibit better performance than those with CLS pooling is consistent with Reimers and Gurevych (2019)'s finding.

Final Step Results: Ensembling & Comparing Methods and Models

Based on the results of our primary study on multi-task model configuration and fine-tuning methods, we believe that the fully fine-tuned TripleBERT and the TreeBERT models with the last 4 encoder layers unfrozen are the top candidates for achieving superior performance and efficiency, respectively. Additionally, our investigations into global and local enhancements demonstrate that the SMART framework and SST lightweight input preprocessing have the potential to enhance our multi-task model's performance, while the default settings of the dropout rate, the STS loss function, and the sentence-pair encoding method are already optimal. We thus apply SMART with weight 0.1 and SST input preprocessing to our two model candidates and introduce a TripleBERT model with the last 4 encoder layers unfrozen for comparison.

Our results in Table 7 demonstrate that the combination of the SMART framework and SST lightweight input preprocessing leads to an increase in the overall performance of the fully fine-tuned TripleBERT by 0.006 and the TreeBERT with 4 layers unfrozen by 0.011. However, this combination does not improve the performance of the TripleBERT model with the last 4 encoder layers unfrozen. Furthermore, although the enhanced TreeBERT with 4 last encoder layers unfrozen still performs slightly worse than the fully fine-tuned TreeBERT, its overall score increases to 0.774. Additionally, it outperforms the TripleBERT model with the last 4 encoder layers by 0.007 while having only one-third of the total and trainable parameters. Finally, we experiment with training the model for 30 epochs instead of 10. We observe a small improvement in TripleBERT performance (+0.001), but a significant decline in TreeBERT performance (-0.006), possibly due to multi-task learning conflicts.

We evaluate our BESTMODEL, the fully fine-tuned TripleBERT trained for 30 epochs with SMART and SST input preprocessing applied, and our EFFICIENTMODEL, TreeBERT with the last 4 encoder layers unfrozen trained for 10 epochs with SMART and SST input preprocessing applied, on the test set. The BESTMODEL achieves an overall test score of 0.791, while the EFFICIENTMODEL achieves an overall test score of 0.765 as shown in Table 8.

Multi-task Model Configuration	SMART	SST IP	Overall Score	Acc_{SST}	Acc_{QQP}	$Corr_{STS}$	# of Parameters (Trainable / Total)
Baseline (ST)	✗	✗	0.604	0.398	0.690	0.448	5,383 / 5,383
TripleBERT (ST)	✗	✗	0.781	0.518	0.886	0.875	328M / 328M
TripleBERT (ST)	✓	✓	0.787	0.533	0.890	0.874	328M / 328M
TripleBERT with $k = 4$ (ST)	✗	✗	0.768	0.507	0.876	0.842	87M / 328M
TripleBERT with $k = 4$ (ST)	✓	✓	0.767	0.507	0.875	0.837	87M / 328M
TreeBERT with $k = 4$ (ST)	✗	✗	0.763	0.503	0.863	0.848	29M / 109M
TreeBERT with $k = 4$ (ST)	✓	✓	**0.774**	**0.516**	**0.872**	**0.868**	**29M / 109M**
TripleBERT (LT)	✓	✓	**0.788**	**0.533**	**0.890**	**0.880**	**328M / 328M**
TripleBERT with $k = 4$ (LT)	✓	✓	0.768	0.507	0.875	0.842	87M / 328M
TreeBERT with $k = 4$ (LT)	✓	✓	0.768	0.504	0.866	0.867	29M / 109M

Table 7: Model performance on dev set when non-architectural methods are applied
[ST = short training (10 epochs) / LT = long training (30 epochs)]

Multi-task Model	Overall Score	Acc_{SST}	Acc_{QQP}	$Corr_{STS}$	# of Parameters (Trainable / Total)
BESTMODEL (TripleBERT + SMART + SST IP + LT + Rounding)	**0.791**	0.543	0.891	0.876	328M / 328M
EFFICIENTMODEL (TreeBERT with $k = 4$ + SMART + SST IP + ST + Rounding)	**0.765**	0.492	0.873	0.862	29M / 109M

Table 8: Performance of our BESTMODEL and EFFICIENTMODEL on test set
[ST = short training (10 epochs) / LT = long training (30 epochs)]
[Rounding = Replace STS prediction that is out of [0.0, 5.0] range with 0.0 or 5.0]

5 Analysis

5.1 Overall Experimental Result Interpretation

It is evident that the fully fine-tuned TripleBERT model excels across all tasks due to its extensive trainable parameters and the capacity for task-specific fine-tuning. These supplementary enhancements like the SMART framework and SST lightweight input preprocessing can further elevate its performance to achieve notable improvements.

Conversely, the performance of the TreeBERT model with the last 4 encoder layers unfrozen is initially constrained by its fewer trainable parameters, resulting in inferior performance compared to TripleBERT with the last 4 encoder layers unfrozen under similar conditions. However, upon applying the two enhancements, the TreeBERT model surpasses the performance of the frozen TripleBERT. This underscores the benefits of multi-task learning over isolated training approaches, particularly when overfitting concerns are addressed and the input is standardized, enabling the model to leverage prior knowledge and learn more effectively.

5.2 Ablation Study

We conduct ablation experiments to dissect the impact of the two key enhancements on our BEST-MODEL and EFFICIENTMODEL. The results are shown in A.3. They indicate that SST Input preprocessing contributes to an increase in SST accuracy for both models. Conversely, while SMART enhances the performance of TripleBERT on STS and QQP tasks, it leads to a decline in performance for TreeBERT on those tasks, as well as in the overall score. However, when both methods are used in combination, we observe a significant improvement in both models, particularly for TreeBERT.

We attribute this success of the combined approach to its ability to prevent TreeBERT from overfitting excessively to one task during joint training, especially when encountering tasks with varying input formats. This synergy effectively ensures that TreeBERT can adapt to diverse task requirements without compromising its performance.

5.3 Model Performance Analysis

Examples referenced in the model performance analysis are provided in A.4.

Sentiment Analysis (SST). We observe that while the SST dataset exhibits class imbalance, our BESTMODEL and EFFICIENTMODEL cannot capture this imbalance well (See A.5). Moreover, we find that, out of 514 misclassifications made by our BESTMODEL on the dev set, 440 instances were off by just one class. In these cases, the sentiment of the sentence is often subjective, and many individuals might agree with the predictions (Ex.1). When the model is off by 2, it is often either because the sentence can be viewed as positive in a haunting way (Ex.2), the model misunderstands nuances of language and sarcasm (Ex.3), or the sentence has a lot of negative words, but the message is overall positive (Ex.4). Finally, there are only 6 sentences where the model is off by 3 or more. In these edge cases, there is no discernable pattern as to why the model predicts the sentiment incorrectly. Our EFFICIENTMODEL exhibits a similar trend, with a higher frequency of misclassifications across all levels of error (See the confusion matrices in A.6 for further insights).

Paraphrase Detection (QQP). The confusion matrices for both models are shown in A.7. Our investigation of both models' predictions reveals frequent false positives for very similar sentences with subtle, crucial differences such that even a person quickly reading it can think they are paraphrases of one another (Ex.5). False negatives occur in sentences that require inference beyond simple language (Ex.6), require better parsing (Ex.7) and instances where the sentences are arbitrary on whether or not they are paraphrases of one another (Ex.8). In comparing the two models, we observe that our BESTMODEL demonstrates a stronger ability to discern differences between two sentences that may appear in similar contexts, while our EFFICIENTMODEL may misinterpret these sentences as paraphrases due to their contextual similarity (Ex.9)

Semantic Textual Similarity (STS). By plotting the histogram of the STS similarities in the training and dev sets, we observe that the labels frequently cluster around integer values rather than being uniformly distributed across the interval. However, our models struggled to effectively capture this clustering pattern as shown in A.8. Additionally, for both models, the majority of errors are less than 1.0, with a bias towards being somewhat similar (See A.9). The model's misclassification of sentences being too similar often occurs due to numerous shared words and a similar structure with a small but significant difference (Ex.10). Conversely, misclassifications as too dissimilar typically resulted from drastically different expressions conveying the same idea (Ex.11). We examine the number of predictions falling within error ranges of 0.01 - 1.0, 1.0 - 2.0, and 3.0+. We observe that our EFFICIENTMODEL exhibits more errors in the 1.0 - 2.0 range than BESTMODEL (135 compared to 114 instances), whereas a similar number of mispredictions are observed in the 3.0+ range.

6 Conclusion

Our findings highlight the pivotal role of the multi-task BERT model configuration in determining performance. Supplementary factors such as the SMART framework and SST lightweight input preprocessing bolsters model performance, particularly in multi-task learning scenarios where overfitting to one task or non-standardized inputs may impede overall performance.

Our key contributions include our BESTMODEL, which achieves an overall test score of 0.791, and our EFFICIENTMODEL, which achieves an overall test score of 0.765. Despite comprising only one BERT with four unfrozen layers and 29M trainable parameters, our EFFICIENTMODEL outperforms the model with three underlying BERTs, equal numbers of unfrozen layers, and 87M parameters on the dev set. Furthermore, despite its inferior performance compared to BESTMODEL, it notably diminishes training time. Additionally, with only one underlying BERT, this model configuration offers a valuable foundation for minimizing inference time when applying identical inputs across multiple tasks, as the input only needs to pass through the BERT layers once.

Moving forward, we aim to explore task scheduling choices for multi-task learning, as this could potentially enhance our shared training outcomes. Furthermore, performing cross-validation can mitigate overfitting to our dev set and ensure the generalizability of our model.

Team Contributions: Pann was responsible for the minBERT implementation, the ADAMW optimizers, and the integration of SMART. Thanawan led the development of TreeBERT and implemented the SST lightweight input preprocessing, STS loss function, and sentence-pair encoding methods. We conducted experiments, performed analysis, and collaborated on writing the paper together.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning.

WenLi Zhuang and Ernie Chang. 2017. Neobility at SemEval-2017 task 1: An attention-based sentence similarity model. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 164–169, Vancouver, Canada. Association for Computational Linguistics.

A Appendix

A.1 Multi-task Model Types

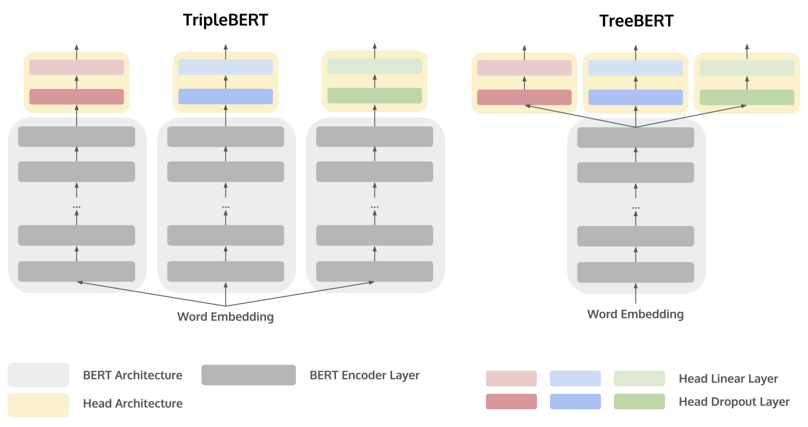


Figure 1: Multi-task Model Types: TripleBERT vs. TreeBERT

A.2 Multi-task Model Configurations

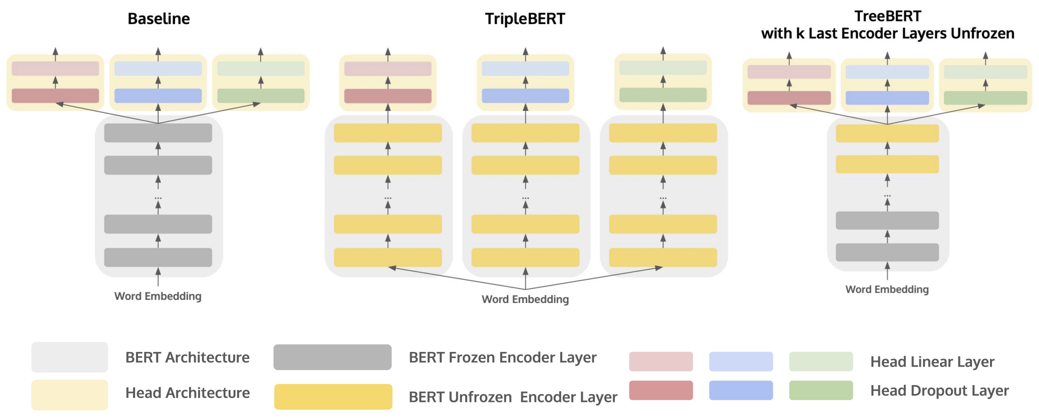


Figure 2: Multi-task Model Configurations: Baseline - TripleBERT - TreeBERT with k Last Encoder Layers Unfrozen

A.3 Ablation Study Results

Multi-task Model Configuration	SMART	SST IP	Overall Score	Acc_{SST}	Acc_{QQP}	$Corr_{STS}$	# of Parameters (Trainable / Total)
TripleBERT (ST)	✗	✗	0.781	0.518	0.886	0.875	328M / 328M
TripleBERT (ST)	✓	✗	0.782	0.518	0.890	0.876	328M / 328M
TripleBERT (ST)	✗	✓	0.782	0.522	0.886	0.875	328M / 328M
TripleBERT (ST)	✓	✓	0.787	0.533	0.890	0.874	328M / 328M
TreeBERT with $k = 4$ (ST)	✗	✗	0.763	0.503	0.863	0.848	29M / 109M
TreeBERT with $k = 4$ (ST)	✓	✗	0.760	0.504	0.855	0.844	29M / 109M
TreeBERT with $k = 4$ (ST)	✗	✓	0.769	0.506	0.873	0.857	29M / 109M
TreeBERT with $k = 4$ (ST)	✓	✓	0.774	0.516	0.872	0.868	29M / 109M

Table 9: Ablation Study on the effects of SMART regularization and SST input preprocessing on our BESTMODEL and EFFICIENTMODEL [ST = short training (10 epochs)]

A.4 Examples Referenced in the Model Performance Analysis Section

Ex.	Task	Sample	Actual	Predicted
1	SST	“It’s a lovely film with lovely performances by Buy and Accorsi.”	3	4
2	SST	“A gripping, searing portrait of a lost soul trying to find her way through life.”	2	4
3	SST	“The film takes the materials of human tragedy and dresses them in lovely costumes, Southern California locations and star power.”	2	4
4	SST	“You’ll gasp appalled and laugh outraged and possibly, watching the spectacle of a promising young lad treading desperately in a nasty sea, shed an errant tear.”	3	1
5	QQP	“What are some facts that everyone knows?” & “What are some facts that everyone should know?”	0.0	1.0
6	QQP	“What does the small blue icon of a man with a plus sign mean in the upper right side of Quora answers?” & “What is the blue human figure button for on right top of quora answers?”	1.0	0.0
7	QQP	“How much does wolfram alpha cost?” & “How much did Wolfram|Alpha cost?”	1.0	0.0
8	QQP	“Why do you learn foreign languages?” & “Why should I learn foreign languages?”	1.0	0.0
9	QQP	“What can you get as a customer of Star Alliance?” & “What are some ways to register with Star Alliance?”	0.0	1.0
10	STS	“Syrian fighter pilot defects to Jordan” & “Syrian PM defects to Jordan”	1.4	3.7
11	STS	“In these days of googling, it’s sloppy to not find the source of a quotation.” & “I agree with Kate Sherwood, you should be able to attribute most quotes these days by simple fact checking.”	3.2	0.83

Table 10: Examples referenced in the model performance analysis section with their actual labels and the predicted labels made by BESTMODEL [Note: We reformatted the spacing for improved readability.]

A.5 Class Imbalance Issues in the SST Dataset

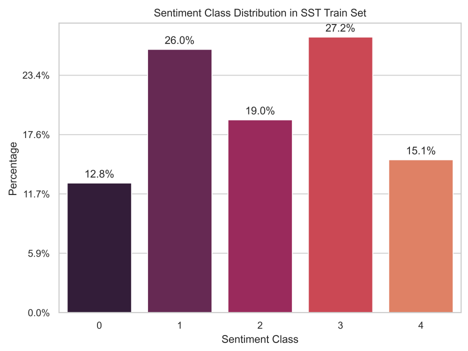
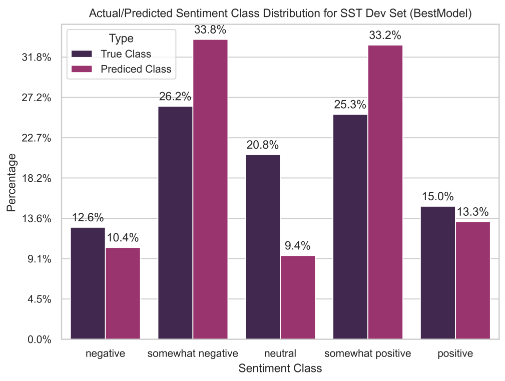


Figure 3: Sentiment Class Distribution in SST Train Set



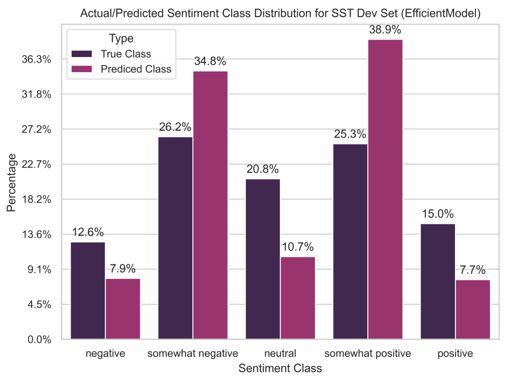
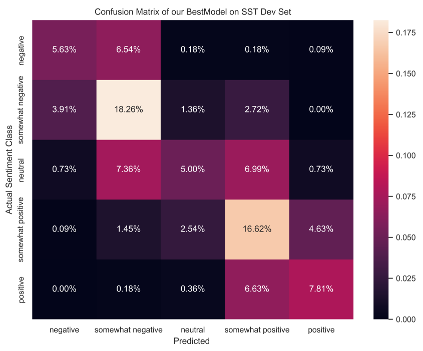


Figure 4: Actual/Predicted Sentiment Class Distribution for SST Dev Set (BESTMODEL)

Figure 5: Actual/Predicted sentiment class distribution for SST dev set (EFFICIENTMODEL)

A.6 Confusion Matrices Illustrating the Performance of the BESTMODEL and the EFFICIENTMODEL on SST Dev Set



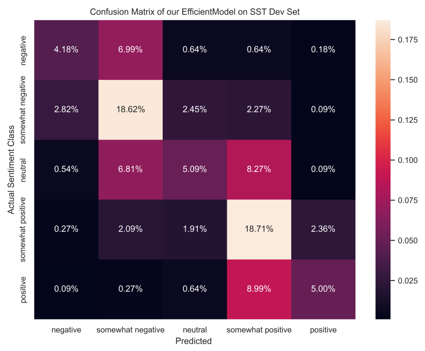


Figure 6: Confusion matrix illustrating the performance of the BESTMODEL on SST dev set

Figure 7: Confusion matrix illustrating the performance of the EFFICIENTMODEL on SST dev set

A.7 Confusion matrix illustrating the performance of the BESTMODEL and the EFFICIENTMODEL on QQP dev set

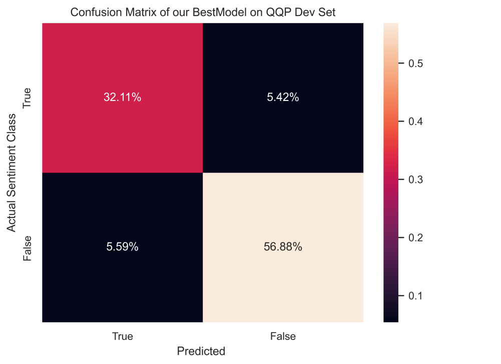


Figure 8: Confusion matrix illustrating the performance of the BESTMODEL on QQP dev set

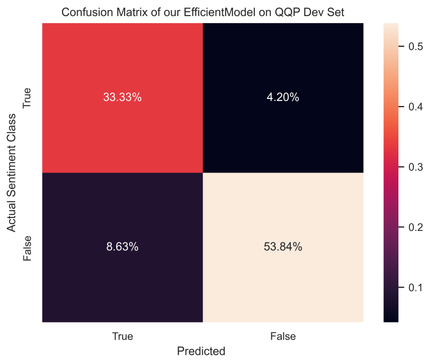


Figure 9: Confusion matrix illustrating the performance of the EFFICIENTMODEL on QQP dev set

A.8 Data Non-uniformity in the STS Dataset

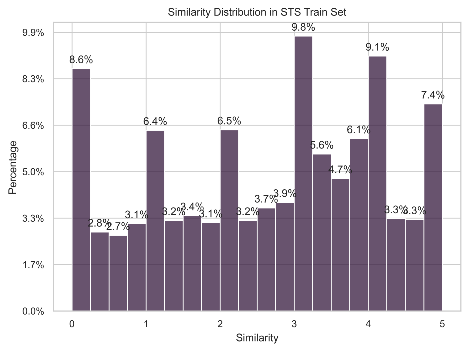


Figure 10: Similarity Distribution in STS Train Set

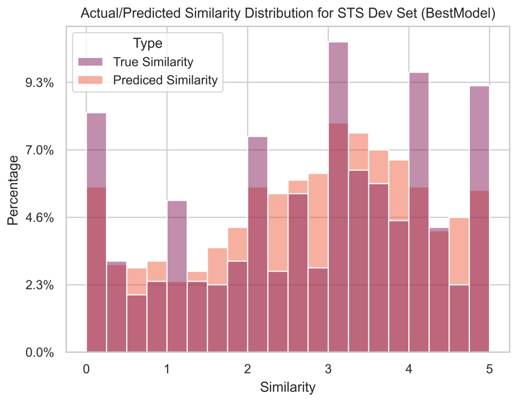


Figure 11: Actual/Predicted similarity distribution for STS dev set (BESTMODEL)

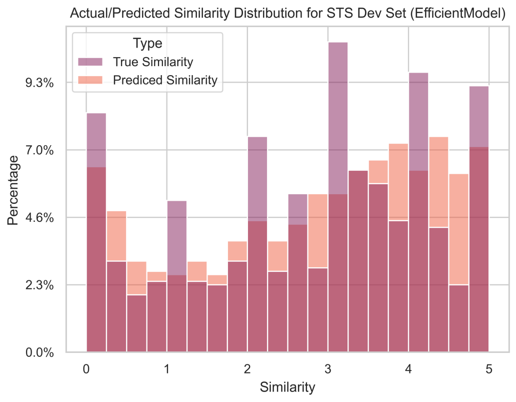


Figure 12: Actual/Predicted similarity distribution for STS dev set (EFFICIENTMODEL)

A.9 Confusion matrix illustrating the performance of the BESTMODEL and the EFFICIENTMODEL on STS dev set

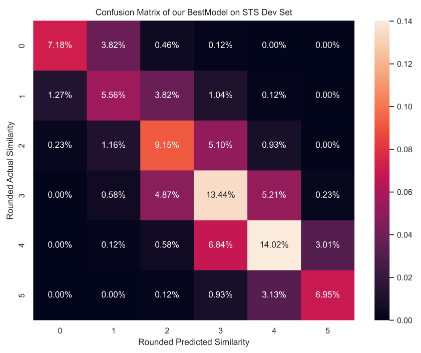


Figure 13: Confusion matrix illustrating the performance of the BESTMODEL on STS dev set (created by rounding similarity values to the nearest integer value)

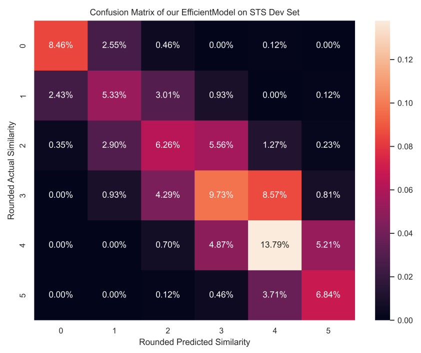


Figure 14: Confusion matrix illustrating the performance of the EFFICIENTMODEL on STS dev set (created by rounding similarity values to the nearest integer)