Một tiếp cận học đa nhiệm dựa trên mô hình ngôn ngữ nhỏ cho một số bài toán xử lý ngôn ngữ tự nhiên

TÓM TẮT

Trong lĩnh vực Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP), các mô hình dựa trên Transformer đã mang lại những kết quả hấp dẫn về khả năng hiểu và sinh ngôn ngữ. Tuy nhiên, số lượng tham số cực lớn và chi phí tính toán cao của các mô hình này gây khó khăn cho việc triển khai trên các thiết bị hạn chế về tài nguyên. Điều này thúc đẩy việc nghiên cứu phát triển các phương pháp hiệu quả, mang lại sự cân bằng giữa kích thước mô hình và hiệu suất. Bài báo này đề xuất một tiếp cận học đa nhiệm mới, dựa trên mô hình ngôn ngữ nhỏ như TinyBERT cho một số bài toán (tác vụ) NLP, bao gồm: phân tích cảm xúc, phát hiện diễn giải và độ tương đồng ngữ nghĩa của văn bản. Chúng tôi thực hiện huấn luyện ban đầu thông qua cơ chế học đa nhiệm để nắm bắt các đặc trưng ngôn ngữ chung. Sau đó, mô hình huấn luyện trước này được tinh chỉnh cho từng tác vụ cụ thể. Chúng tôi tiến hành các thực nghiệm để đánh giá hiệu quả của phương pháp này trên bộ dữ liệu chuẩn GLUE. Kết quả chứng minh tính hiệu quả của phương pháp của chúng tôi trong thiết lập học đa nhiệm cho các tác vụ NLP khác nhau. Chúng tôi cung cấp cài đặt thực nghiệm của nghiên cứu này trên kho lưu trữ Github (*https://github.com/Tuyenle2/Project-code-2025.1143.93*).

Từ khóa: *Xử lý ngôn ngữ tự nhiên, học đa nhiệm, mô hình ngôn ngữ nhỏ, tinh chỉnh, TinyBERT.*

A Multi-Task Learning Approach Based on Small Language Model for Natural Language Processing Tasks

ABSTRACT

In the field of Natural Language Processing (NLP), Transformer-based models have yielded appealing results in language understanding and generation. However, these models' extremely high number of parameters and computationally expensive are challenging to deployment on resource-constrained devices. This motivates research in the development of efficient methods that offer an attractive balance between model size and performance. This paper proposes a novel multi-task learning approach, which relies on a small language model like TinyBERT for NLP tasks, including: sentiment analysis, paraphrase detection, and semantic textual similarity. We perform initial training through a Multi-Task Learning (MTL) mechanism to capture general language features. Subsequently, this pre-trained model is fine-tuned for each specific task. We conduct experiments to evaluate the effect of this approach on the GLUE benchmark dataset. The results demonstrate the effectiveness of our method within a multi-task learning setup for diverse NLP tasks. We provide an implementation of this work on Github repository (*https://github.com/Tuyenle2/Project-code-2025.1143.93*).

Keywords: *Natural language processing, multi-task learning, small language model, fine-tuning, TinyBERT.*

1. INTRODUCTION

Large language models such as BERT[1], RoBERTa, T5[2], XLNet[3], ELECTRA[4], and GPT[5] have revolutionized the field of NLP, achieving state-of-the-art performance on a wide range of tasks. However, their massive size and high computational requirements pose significant barriers to deployment on resource-constrained devices. TinyBERT[6], a compact version of BERT created through knowledge distillation, has emerged as a potential solution, offering an attractive balance between model size and performance. To fully exploit the potential of these compact models, multi-task fine-tuning (MT-FT) has become an increasingly popular research direction in the field of NLP[7]. The primary goal of MTL is to improve the performance of multiple related tasks by leveraging useful information shared among them[8]. Instead of training separate models for each task, MTL allows a single model to learn simultaneously from multiple tasks, thereby promoting knowledge sharing and improving generalization capabilities[7]. Various multi-task architectures and strategies have been proposed[8], from shared encoder layers with task-specific output layers as in multi-task deep neural networks[7], to unified "text-to-text" approaches like T5[2,9], or efficient adapter-based techniques[10].

Although modern NLP models continually push the boundaries of performance, the effective application of MT-FT, especially on compact models like TinyBERT, still faces many challenges. Key challenges in-clude selecting and balancing tasks to avoid negative transfer, designing optimal network architectures for efficient parameter sharing[7], dealing with catastrophic forgetting when learning multiple tasks, and particularly addressing the limited capacity of small models when handling the complexity of multiple data sources and learning objectives[10]. Building on these considerations, we pose three research questions (RQ) and present main contributions to address them.

- **RQ1:** How can TinyBERT be leveraged in a multi-task learning architecture to simultaneously learn these three tasks, exploiting their commonalities to improve generalization and computational efficiency?
- **RQ2:** After initial multi-task training, to what extent can individual fine-tuning for each task improve

performance compared to traditional single-task training?

- **RQ3:** How does this approach balance performance and computational efficiency compared to larger models (like BERTbase) or other approaches in current research?

To answer these questions, we design a multi-task learning framework built upon TinyBERT[6] to tackle multiple NLP tasks including: sentiment analysis, paraphrase detection, and semantic textual similarity. Our main contributions are summarized as follows:

- First, we propose a novel multi-task learning approach, which relies on a small language model for NLP tasks, aiming to address the efficiency-performance trade-off under resource constraints.
- Second, we demonstrate that leveraging shared representations through MTL followed by individual task-specific finetuning significantly enhances the overall performance on sentiment analysis, paraphrase detection, and semantic textual similarity tasks.
- Third, we provide empirical evidence showing that despite its compact size, TinyBERT can achieve competitive performance with considerably lower computational cost, offering a practical alternative to larger models such as BERT-base in real-world applications.

The rest of this paper is organized as follows. Section 2 provides preliminaries on problem statement, multi task learning, fine tuning, and TinyBERT that form the basis for the development of the proposed method detailed in Section 3. Section 4 describes the experimental setup, results and comparing the result with BERT-base in Section 5. Section 6 briefly recalls some related work. Finally, Section 7 wraps up the paper with conclusions and future work.

2. BACKGROUND

2.1. Multi-Task Learning

Multi-task learning (MTL for short) is a machine learning approach in which a single model is trained to perform multiple related tasks simultaneously. The core idea is that different tasks can share information or common representations, which helps to improve overall performance, enhance generalization capabilities, and reduce the risk of overfitting, especially when data for individual tasks is limited[11]. In NLP, MTL is often applied by sharing the encoder layers of the model across multiple tasks, while having separate output layers for each specific task[7].

Formally, the total loss in MTL can be defined as:

$$L_{\text{total}} = \sum_{i=1}^{T} L_i \quad (1)$$

where L_i denotes the task-specific loss for the ith task, and T is the number of tasks. To better balance task learning, dynamic weighting strategies have been proposed. One such formulation is:

$$L_{\text{total}} = \sum_{i=1}^{T} w_i L_i \quad \text{with} \quad w_i = \frac{L_{\text{total}}}{T \times L_i} \quad (2)$$

This joint learning allows the model to capture fundamental language features useful for various objectives.

2.2. Problem Statement

Let D be the set of input texts (or text pairs, depending on the specific task), where each element $x \in D$ is a sentence, a text passage, or a pair of sentences/words. For each text x (or text pair), there will be corresponding labels for each task:

- Sentiment analysis (SA): $y_{sa} \in (0; 1)$ where $y_{sa} = 0$ if the text has negative sentiment, and $y_{sa} = 1$ if positive.
- Paraphrase detection (PD): For a pair of texts $(x_1; x_2)$, the label $y_{pd} \in (0; 1)$, where $y_{pd} = 0$ if the two texts are not paraphrases of each other, and $y_{pd} = 1$ if they are paraphrases.
- Semantic textual similarity (STS): For a pair of words $(w_1; w_2)$, the label $y_{sts} \in (0; 1)$, where $y_{sts} = 0$ if the two words are antonyms, and $y_{sts} = 1$ if they are synonyms (in some cases, this task might involve determining if a word in a specific context can be replaced by another word without significantly changing the sentence's meaning).

The MTL problem with TinyBERT can be modeled as a function f_{mtl}:

$$f_{mtl} : D \rightarrow (y_{sa}, y_{pd}, y_{sts}) \quad (3)$$

Here, $f_{mtl}(x)$ includes the model's predicted labels $(y_{sa}, y_{pd}, y_{sts})$ or the input x across all three

tasks. (y_{sa}, y_{pd}, y_{sts}) represents the set of possible labels for each task.

2.3. Fine-Tuning

Fine-tuning (FT for short) is a popular transfer learning technique[12] where a model that has been pre-trained on a large amount of data (often self-supervised tasks like language modeling) is then adjusted or "fine-tuned" on a smaller, task-specific dataset for a target task[13]. The pretraining process helps the model learn rich and contextual language representations[14].

The fine-tuning step allows the model to adapt this learned knowledge to the nuances of the specific task, often leading to much better performance than training a model from scratch on limited task data[1]. In classification tasks, the typical loss function used during fine-tuning is the cross-entropy loss:

$$L_{\text{CE}} = -\sum_{c=1}^{C} y_c \log(p_c) \tag{4}$$

where C is the number of classes, yc is the true label distribution, and pc is the predicted probability for class c. In more recent setups, supervised contrastive learning[15,16] is also applied to encourage discriminative feature learning:

$$L_{\text{SCL}} = -\frac{1}{N}\sum_{i=1}^{N}\frac{1}{|P(i)|}\sum_{p\in P(i)}\log\frac{\exp(s_{i,p}/\tau)}{\sum_{a\in A(i)}\exp(s_{i,a}/\tau)} \tag{5}$$

where:

- N: total samples in a batch.
- $P(i)$: set of positives (samples with the same label as i, excluding i).
- $|P(i)|$: number of positives for anchor i.
- $A(i)$: set of all samples in the batch excluding i (positives and negatives).
- si,p: cosine similarity between embedding of i and p.
- τ: temperature parameter controlling sharpness of the distribution.

2.4. TinyBERT

While large models like BERT[1,9] achieve impressive performance, their size and computational cost are significant barriers to deployment on resource-constrained devices. To address this issue, more compact models have been developed. TinyBERT[6] utilizes *knowledge distillation* to reduce the size of BERT while retaining a large portion of its performance.

This distillation process involves transferring knowledge from a large *teacher* model (the original BERT) to a smaller *student* model (TinyBERT) during both the pre-training and taskspecific fine-tuning stages. The knowledge distillation loss in TinyBERT typically integrates several components:

$$L_{\text{KD}} = \alpha L_{\text{emb}} + \beta L_{\text{hidden}} + \gamma L_{\text{logit}} \tag{6}$$

where L_{emb}, L_{hidden}, and L_{logit} refer to the embedding-level, hidden-state-level, and outputlevel distillation losses, respectively.

3. METHODOLOGY

This section presents the architecture and training process of our proposed approach, which combines MTL and fine-tuning strategies using the TinyBERT model. First, we detail the architecture that supports shared learning across multiple tasks. Then, we describe how the TinyBERT backbone is trained in a multi-task setting. Finally, we explain the individual fine-tuning phase applied after MTL for task-specific optimization.

3.1. Model Architecture

Our architecture utilizes TinyBERT as a shared backbone to extract contextual representations across all tasks. As shown in Figure 1, raw text data is fed into the system as character strings. This text is pre-processed by the TinyBERT-Tokenizer to convert it into corresponding tokens. The encoded tokens from each task are then passed to the TinyBERT backbone. This is the shared architectural component, sharing weights across all tasks. TinyBERT processes these token sequences and generates contextual representations at the last hidden layer. Specifically, the embedding vector corresponding to the special [CLS] token from the last hidden layer is usually taken as a representation for the entire input sequence.

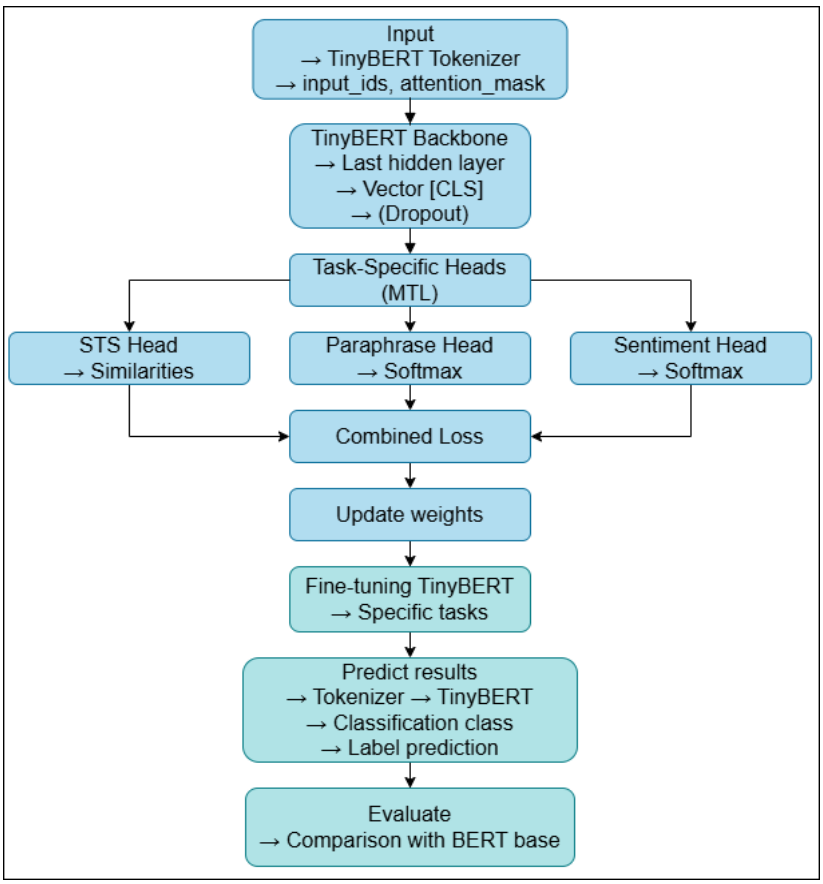

Figure 1. Overview of proposed method.

This vector is expected to capture the composite information of the sentence. A dropout layer is applied after extracting the [CLS] vector to help the model prevent overfitting during the MTL training process. The [CLS] representation (after the dropout layer) is then fed into separate heads, each designed for a specific task within the set of MTL tasks. The loss function from each task head is computed independently. These loss functions are then combined to form a joint loss function for the entire MTL model.

$$L_{total} = w_{sa}L_{sa} + w_{pd}L_{pd} + w_{si}L_{si} \quad (7)$$

where w_{sa}, w_{pd}, w_{si} are weights to balance the contribution of each task. The optimization process relies on this joint loss function to simultaneously update the weights of the TinyBERT backbone and the task-specific heads. This allows TinyBERT to learn robust and flexible language representations from various information sources. After TinyBERT has been pre-trained with MTL and possesses good language understanding capabilities, it is then fine-tuned for the specific task of interest. In here, we fine-tune for all three aforementioned tasks to compare the model's performance after fine-tuning with the model after multi-task training. After finetuning, the complete TinyBERT model (including the TinyBERT backbone and the classification layer for the target task) is used to make predictions. With a new input text, the system will process it through the tokenizer, feed it into TinyBERT, pass it through the classification layer, and finally predict the label based on the class with the highest probability.

The training process comprises two main phases:

- **Phase 1: Multi-task Training.** The input text x (or its components, such as x_1; x_2 for PD, or words w_1; w_2 in context for SI) is fed through the TinyBERT model. TinyBERT generates contextual embeddings (or a set of embeddings). The key point of MTL is that most of TinyBERT's parameters (the Transformer layers) are shared across all tasks, allowing the model to learn general representation useful for multiple tasks.
- **Phase 2: Fine-tuning for Specific NLP Tasks.** From the shared

representation ex obtained from TinyBERT separate output layers are used for each task. A classification layer is applied to the representation of the [CLS] token from ex to predict the sentiment label *ysa*. The representation of the sentence pair from TinyBERT (typically the [CLS] token's representation when the two sentences are concatenated and fed into the model) is passed through a classification layer to predict the label y_{pd}. Similarly, the representation of the word pair from TinyBERT is passed through a classification layer to predict the label y_{sts}.

This architecture enables TinyBERT to learn generalized features that are beneficial across tasks, while still allowing task-level specialization through individual fine-tuning.

3.2. Multi-task TinyBERT Training

Algorithm 1 describes the training procedure for the multi-task TinyBERT model. First, the TinyBERT model is initialized with pre-trained weights. Then, appropriate labels and classification layers are added for each task. Data is divided into batches, and each batch is tokenized using TinyBERT's tokenizer to convert text into a suitable tokenized format. Subsequently, the TinyBERT model extracts an embedding representing the entire sentence from the [CLS] token. Next, the [CLS] token (after the dropout layer) is fed into separate heads, each designed for a specific task within the set of MTL tasks. Finally, the loss for each task is calculated based on its type, and model parameters are updated through a gradient descent algorithm. This process is repeated over multiple epochs, and the AdamW optimizer is used to optimize the model's. The TinyBERT backbone is shared among tasks, allowing the model to learn general representations useful for multiple tasks, while separate outputs enable the model to adjust predictions for each specific task. During training, the model receives input as input-ids and attention-mask, along with a task parameter specifying the current task. Based on the task, the model uses the corresponding classification or regression layer to produce output. With the provided labels, the model calculates the appropriate loss function (crossEntropyLoss for classification (3)(4), MSELoss for regression (5)) for use during training. This architecture allows the model to learn shared general representations through the shared TinyBERT layer and then specialize for each task through separate output layers.

Algorithm 1 Multi-task TinyBERT Training

Require: Multi-task dataset D for SST-2, QQP, and STS-B; pre-trained TinyBERT model; number of tasks N_C; task-specific weights $W = \{w_{sa}, w_{pd}, w_{sts}\}$, initially set to 1.0; hyperparameters: number of epochs E, batch size B, embedding size f, hidden layer size H

Ensure: Trained multi-task TinyBERT model with shared backbone and task-specific heads; optimized combined loss function L_{total}

1: Load pre-trained TinyBERT as shared backbone

2: Add task-specific heads:

-SST-2: Linear + Softmax for binary classification ($y_{sa} \in \{0, 1\}$)

-QQP: Linear + Softmax for binary classification ($y_{pd} \in \{0, 1\}$)

-STS-B: Linear layer for regression ($y_{sts} \in [0, 5]$)

3: Load and preprocess dataset D

4: Tokenize text using TinyBERT-Tokenizer 5:

5: **for** epoch $e = 1$ to E **do**

6: Divide D into mini-batches of size B

7: **for** each mini-batch **do**

8: Pass input through TinyBERT to obtain [CLS] embeddings

9: Apply dropout to [CLS] embeddings

10: Compute predictions from task-specific heads

$$\mathcal{L}_{sa} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{c=0}^{1} y_{i,c}\log(P_{i,c})$$

$$\mathcal{L}_{pd} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{c=0}^{1} y_{i,c}\log(P_{i,c})$$

$$\mathcal{L}_{sts} = \frac{1}{B}\sum_{i=1}^{B}(\text{score}_{sts,i} - y_{sts,i})^2$$

11: Combined Loss:

12: $L_{total} = w_{sa}L_{sa} + w_{pd}L_{pd} + w_{sts}L_{sts}$

13: Compute gradients of L_{total}

14: Update model parameters using optimizer

15: **end for**

16: **end for**

17: **return** Trained TinyBERT model, L_{total}, task-specific head parameters.

Algorithm 2 Fine-tuning TinyBERT for

Specific NLP Tasks

Require: Pre-trained multi-task TinyBERT model, task-specific dataset D_{task}, loss function L_{task}, hyperparameters: number of epochs E, batch size B

Ensure: Fine-tuned TinyBERT model optimized for the specific task

1: **Initialize:** Load pre-trained TinyBERT (shared backbone + task-specific head)

2: **Prepare Dataset:** Load and preprocess D_{task} using TinyBERT tokenizer to get input IDs and attention masks

3: **for** epoch $e = 1$ to E **do**

4: Divide D_{task} into mini-batches of size B

5: **for** each mini-batch **do**

6: Forward pass:

-Pass tokenized inputs through Tiny- BERT to get [CLS] embeddings

-Feed embeddings to task-specific head to get predictions

7: Compute loss L_{task}:

-SST-2 or QQP (classification):

$$\mathcal{L}_{\text{task}} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{c=0}^{1} y_{i,c}\log(P_{i,c})$$

-STS-B (regression):

$$\mathcal{L}_{\text{task}} = \frac{1}{B}\sum_{i=1}^{B}(\text{score}_{\text{pred},i} - y_{\text{true},i})^2$$

8: Backward pass:

Compute gradients of L_{task}

9: Update model parameters using optimizer

10: **end for**

11: **end for**

12: **return** Fine-tuned TinyBERT model

3.3. Fine-tuning TinyBERT for Specific NLP Tasks

Algorithm 2 describes the process of fine-tuning TinyBERT for specific NLP tasks. After being pre-trained using MTL acquiring strong language understanding capabilities, TinyBERT is further fine-tuned on the target task of interest. Here, we fine-tune the model for all three aforementioned tasks to compare its performance after fine-tuning with its performance immediately after multi-task training. We use the multi-task trained TinyBERT model as the starting point and further train it on data specific to each individual task, utilizing the corresponding loss function. The model's weights are updated to optimize performance for that task. This fine-tuning process allows the model to adapt the representations learned during the MTL phase to better fit the unique characteristics and requirements of each task, thereby enhancing overall accuracy and performance.

4. EXPERIMENTS

In this section, we present the experimental implementation based on the method described in Section 3. Our aim is to explore the effectiveness of using a small language model like TinyBERT in a multi-task learning setup for diverse NLP tasks.

4.1. Dataset

The GLUE benchmark (General Language Understanding Evaluation) is a widely recognized collection of diverse NLP tasks designed to assess the language understanding capabilities of machine learning models. This benchmark comprises various tasks that test different aspects of natural language understanding. The study utilizes three datasets from GLUE benchmark:

- SST-2 (Stanford Sentiment Treebank): A sentiment classification dataset comprising sentences from movie reviews, labeled as either positive or negative. It includes approximately 67,000 sentences for training and about 872 sentences for development (validation) and 1,821 sentences for testing.
- QQP (Quora Question Pairs): A paraphrase detection dataset containing pairs of questions from the Quora website, labeled to indicate whether the two questions are semantically equivalent. This is a large dataset with over 363,780 question pairs for training and around 40,431 pairs for development and 390,965 pairs for testing. Question pairs can have different phrasing but still convey similar meanings, or vice versa. It uses binary labels: duplicate or non-duplicate.

Table 1. Validation performance at different training stages.

Task	Measure	After MTL	After FT
Sentiment analysis	Accuracy	0.8704	0.8922
Paraphrase detection	Accuracy	0.8737	0.8822
Semantic textual similarity	Pearson	0.8624	0.8690

- STS-B (Semantic Textual Similarity Benchmark): A semantic textual similarity dataset containing sentence pairs from various sources, labeled with scores from 1 to 5 indicating their degree of semantic similarity. It includes approximately 7,000 sentence pairs for training, 1,500 pairs for development, and 1,400 pairs for testing. Unlike SST-2 and QQP with discrete labels, STS-B uses a continuous scale, requiring the model to understand a more detailed level of semantic correlation.

4.2. Evaluation

We evaluate the model's performance on the validation sets for each task. For sentiment analysis and paraphrase detection, we use evaluate-classification:

$$\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{total number of samples}} \quad (8)$$

And for semantic textual similarity, we use evaluate-STS, which calculates the Pearson correlation coefficient:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \quad (9)$$

- x_i: Predicted score ($score_{si}$).
- y_i: Actual score (y_{si}).
- $\bar{x}, \bar{y}$: Mean values of predicted and actual scores.

4.3. Experimental Setup

The experiments were conducted using PyTorch and the Hugging Face Transformers library on a single GPU. The pre-trained TinyBERT model "huawei-noah/TinyBERT-General- 4L-312D"[7] was used as the backbone. The specific version TinyBERT-General-4L-312D refers to a TinyBERT model with a general configuration ("General"), comprising 4 Transformer layers and a hidden dimension size of 312[7]. Combining MTL with fine-tuning of a compact model like TinyBERT-General-4L-312D promises to leverage the advantages of both approaches: the ability to learn rich representations from multiple tasks and the deployment efficiency of a small model. The batch size was set to 16. The AdamW optimizer was employed with a learning rate of 1e-5 and a dropout rate of 0.01. The number of training epochs for both the initial multi-task training and individual fine-tuning for each task was 5. Early stopping was triggered if the validation score did not improve for 2 consecutive epochs. Mixed precision training was utilized to accelerate training and reduce memory consumption.

5. RESULTS AND DISCUSSION

The model's validation performance at different training stages is presented in Table 1 and visualized in the following Figure 2.

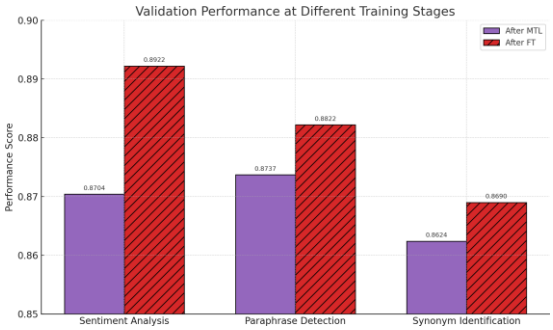

Figure 2. Performance comparison across tasks.

The results show that initial multi-task training enables the model to achieve significant performance across all three tasks. After individual fine-tuning, the accuracy for sentiment and paraphrase tasks improved, while the Pearson correlation for the semantic textual similarity task also slightly increased.

Table 2. Comparing TinyBERT and BERT-base.

Criteria	TinyBERT	BERT-base
Parameters	~14.5M	~110M
Transformer Layers	4	12
Hidden Dimensions	312	768
Inference Time	~4–6x faster	Standard
Memory Required	~1/7	High

Table 3. Comparing TinyBERT and BERT-base performance.

Tasks	TinyBERT (MTL)	TinyBERT (FT)	BERT-base (Single)	BERT-base (MTL)
SST-2	0.8704	0.8922	0.927	0.913
QQP	0.8737	0.8822	0.913	0.904
STS-B	0.8624	0.8690	0.894	0.885

The high performance achieved after the initial multi-task training phase indicates that the TinyBERT model can effectively learn shared representations beneficial for all three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. This suggests that some fundamental linguistic and semantic knowledge can be transferred among these tasks. Thus, the answer to **RQ1** is that TinyBERT can learn effectively from three tasks simultaneously, achieving good performance with a compact model.

The improvement in performance after individual fine-tuning for each task demonstrates that continuing to train the model on task-specific data allows it to adjust its weights to better capture the specific nuances of each task, leading to improved performance. From this, we derive the answer to **RQ2**: Individual fine-tuning after MTL is clearly effective, helping to optimize for each task and narrowing the gap with larger models.

The initial MTL phase appears to have facilitated some positive knowledge transfer among tasks, as evidenced by the reasonable performance achieved across all three tasks with a single shared model. The further improvements observed during individual fine-tuning indicate that additional specialization on each task is beneficial for maximizing performance.

Comparing performance across tasks, the paraphrase detection task achieved the highest accuracy in both the initial multi-task training phase and after individual fine-tuning. This could be due to the clear binary nature of the task (duplicate or non-duplicate) and the relatively large size of the QQP dataset. The semantic textual similarity task, being a regression task predicting a continuous score, had a slightly lower Pearson correlation compared to the accuracy achieved in the classification tasks. Table 2 compares TinyBERT and BERTbase models, TinyBERT is significantly more compact than BERT-base, featuring 7.6 times fewer parameters. This leads to faster inference speeds and lower memory requirements, making it ideal for resource-constrained devices.

We compare our method and results with related work on multi-task learning and TinyBERT on GLUE tasks: Table 3 compares performance.

- BERT-base (single task): Devlin et al.[1], fine-tuned individually on each GLUE task.
- BERT-base (MTL): Liu et al.[7] multi-task trained on similar tasks.

TinyBERT achieves quite good performance (87- 88%) even before fine-tuning, indicating that MTL helps leverage common knowledge among tasks. However, it is still about 3-4% lower than BERT-base MTL (90-91%), reflecting the difference in representational capacity due to model scale. This suggests that while TinyBERT can learn common features, it's not as robust as BERT-base due to its limited number of layers and parameters. After

fine-tuning, TinyBERT improves by 1-2% across all three tasks, narrowing the gap with BERT-base. However, it does not surpass BERT-base.

Compared to BERT-base, multitask TinyBERT achieves 2-4% lower performance across all three tasks (SST-2, QQP, STS-B), even after individual fine-tuning. However, it excels in computational efficiency, being 7.6 times smaller and 4-6 times faster in inference. This makes TinyBERT a reasonable choice in scenarios where efficiency is prioritized over maximum performance, such as deployment on edge devices or in real-time applications. The combined MTL and FT approach of this study also shows potential in leveraging a single model for multiple tasks, reducing training costs compared to individually training separate BERT-base models. This directly answers **RQ3**: Our method achieves a reasonable balance between performance and computational efficiency, which is particularly useful in practical applications requiring resource savings.

6. RELATED WORK

Recent advances in NLP have explored both large-scale pre-trained models and compact architectures optimized for efficiency. In particular, MTL and model compression have emerged as two key strategies to improve performance and reduce computational costs.

Stickland and Murray introduced projected attention layers to adapt BERT for MTL in[7]. By adding lightweight task-specific adapters, they achieved strong performance on the GLUE benchmark while maintaining efficiency. Their results confirmed that parameter-efficient adaptation strategies can make large models more scalable across multiple tasks.

Jiao et al. proposed TinyBERT in[6], which applied a two-stage knowledge distillation approach first at the pre-training level and then at the task-specific fine-tuning level to compress BERT-base into a smaller model while retaining competitive performance. TinyBERT-4L, in particular, achieved over 96% of BERT-base's performance on GLUE with only about 13% of its parameters.

While TinyBERT was initially designed for single-task fine-tuning, later studies have extended its application to MTL. A notable example is the work by Yu et al., which explored the effect of multi-task fine-tuning on small models such as Phi-3-Mini in the financial domain. Their study revealed that combining related tasks significantly improved performance even outperforming large models like GPT-4-o in domainspecific benchmarks.

In addition, Houlsby et al.[10] proposed adapter layers for parameter-efficient fine-tuning, enabling multitask training with minimal parameter overhead. The adapters can be selectively trained for new tasks, making them particularly useful in low-resource and multi-domain settings.

Furthermore, a growing body of literature focuses on dynamic task weighting to improve MTL stability and mitigate negative transfer. For instance, Kendall et al.[17] proposed weighting losses based on task uncertainty, while Lakkapragada et al. used exponential moving average loss strategies to maintain balance during training.

Some recent surveys, for example those by Zhang and Yu[7], Crawshaw, and Yu et al., Liu et al.[18] have analyzed the training strategies and task relatedness in MTL. These works emphasize the importance of sharing knowledge across tasks, selecting compatible task combinations, and using dynamic weighting schemes to mitigate negative transfer.

7. CONCLUSION

In this study, we proposed a novel multitask learning approach, which relies on a small language model like TinyBERT for NLP tasks, including: sentiment analysis, paraphrase detection, and semantic textual similarity. We first trained a shared TinyBERT model using a multitask learning framework, then fine-tuned it individually for each task. The experimental results showed that the initial MTL phase enabled the model to capture shared representations that achieved strong performance across all tasks. Subsequent task-specific fine-tuning further improved accuracy and correlation scores, demonstrating the value of combining MTL with finetuning. Specifically, TinyBERT achieved competitive performance while remaining significantly smaller and faster than BERT-base making it suitable for resource-constrained environments. These findings highlighted the potential of small language models like TinyBERT to serve as efficient and effective solutions for multi-task NLP scenarios. This approach also offers

practical benefits in deployment settings, such as reduced memory usage and training costs, without a significant compromise in task performance.

In the future, we will explore different task balancing strategies in the initial MTL phase to potentially improve learned shared representations. Investigating other fine-tuning techniques and hyper parameters for each task could also lead to better individually fine-tuned models, which could then benefit the aggregation process. Experimenting with more sophisticated methods to determine task similarity, possibly considering uncertainty or variance in individual model performance, could also be valuable. Evaluating the performance of all models on the held-out GLUE test sets would provide a more comprehensive assessment of their generalization capabilities.

REFERENCES

1. Devlin, M.-W. Chang, K. Lee, K. Toutanova. *BERT: Pre-training of deep bidirectional transformers for language understanding*, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), Minneapolis, USA, 2019.
2. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, *Journal of Machine Learning Research*, **2020**, *21*(140), 1–67.
3. Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, Q. V. Le. *XLNet: Generalized autoregressive pretraining for language understanding*, Advances in Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada, 2019.
4. K. Clark, M.-T. Luong, Q. V. Le, C. D. Manning. *ELECTRA: Pre-training text encoders as discriminators rather than generators*, International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 2020.
5. T. Brown, B. Mann, N. Ryder, M. Subbiah, et al. *Language models are few-shot learners*, Advances in Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada, 2020.
6. X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, Q. Liu. *TinyBERT: Distilling BERT for natural language understanding*, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020), Online Event, 2020.
7. P. Liu, X. Qiu, X. Huang. *Multi-task deep neural networks for natural language understanding*, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy, 2019.
8. Y. Zhang, Z. Yu. *A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods*, Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Dubrovnik, Croatia, 2023.
9. K. Clark, U. Khandelwal, O. Levy, C. D. Manning. *What does BERT look at? An analysis of BERT's attention*, Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy, 2019.
10. N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, S. Gelly. *Parameter-efficient transfer learning for NLP*, Proceedings of the 36th International Conference on Machine Learning (ICML), Long Beach, USA, 2019.
11. R. Caruana. Multitask learning, *Machine Learning*, **1997**, *28*(1), 41–75.
12. C. Sun, X. Qiu, Y. Xu, X. Huang. *How to fine-tune BERT for text classification?*, Chinese Computational Linguistics (CCL), Kunming, China, 2019.
13. J. Howard, S. Ruder. *Universal language model fine-tuning for text classification*, Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), Melbourne, Australia, 2018.
14. B. McCann, J. Bradbury, C. Xiong, R. Socher. *Learned in translation: Contextualized word vectors*, Advances in Neural Information Processing Systems (NeurIPS), Long Beach, USA, 2017.
15. B. Gunel, J. Du, A. Conneau, V. Stoyanov. *Supervised contrastive learning for pre-*

trained language model fine-tuning, International Conference on Learning Representations (ICLR 2021), Vienna, Austria, 2021.

16. P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, C. Isola, A. Maschinot, C. Liu, D. Krishnan. *Supervised contrastive learning*, Advances in Neural Information Processing Systems (NeurIPS), Vancouver, Canada, 2020.
17. A. Kendall, Y. Gal, R. Cipolla. *Multi-task learning using uncertainty to weigh losses for scene geometry and semantics*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, USA, 2018.
18. H. Liu, Y. Shen, D. Jin, et al. Multi-task learning for natural language processing: A survey, *ACM Computing Surveys*, **2021**, *54*(10s), 1–43