RevMUX: Data Multiplexing with Reversible Adapters for Efficient LLM Batch Inference

Yige Xu[1,2], **Xu Guo**[1]*, **Zhiwei Zeng**[1]*, **Chunyan Miao**[1,2]
[1]Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly
[2]College of Computing and Data Science
Nanyang Technological University, Singapore
{yige002,xu008}@e.ntu.edu.sg, {zhiwei.zeng,ascymiao}@ntu.edu.sg

Abstract

Large language models (LLMs) have brought a great breakthrough to the natural language processing (NLP) community, while leading the challenge of handling concurrent customer queries due to their high throughput demands. Data multiplexing addresses this by merging multiple inputs into a single composite input, allowing more efficient inference through a shared forward pass. However, as distinguishing individuals from a composite input is challenging, conventional methods typically require training the entire backbone, yet still suffer from performance degradation. In this paper, we introduce RevMUX, a parameter-efficient data multiplexing framework that incorporates a reversible design in the multiplexer, which can be reused by the demultiplexer to perform reverse operations and restore individual samples for classification. Extensive experiments on four datasets and three types of LLM backbones demonstrate the effectiveness of RevMUX for enhancing LLM inference efficiency while retaining a satisfactory classification performance.

1 Introduction

In recent years, Large Language Models (LLMs), such as GPT-3 (175B) (Brown et al., 2020), PaLM (540B) (Chowdhery et al., 2023), and GPT-4 (1.7T) (OpenAI, 2023) have emerged as a cornerstone in Natural Language Processing (NLP). The field has witnessed a dramatic increase in model sizes, which, although improving downstream performance, also poses considerable challenges. Inference with these LLMs has become increasingly resource-intensive, often confronting users with capacity limits (OpenAI, 2023). With the rise of "language model as a service" (Sun et al., 2022b), improving inference efficiency has become a key focus to accommodate these growing model sizes.

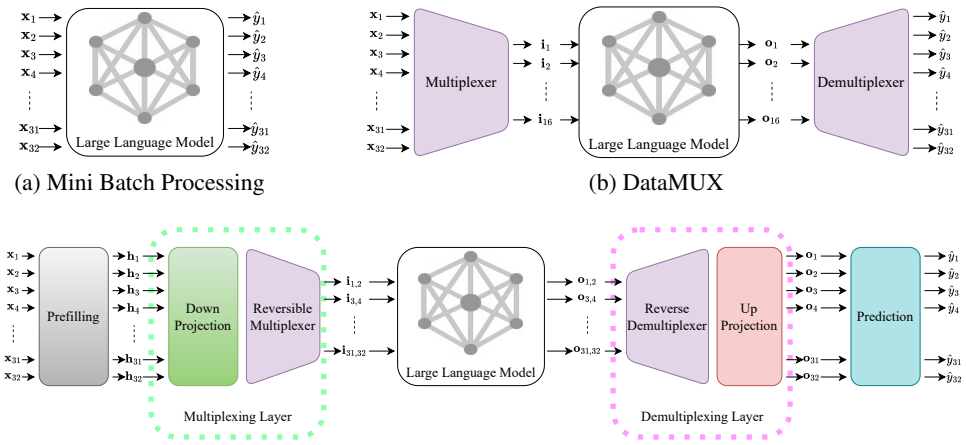
To explore efficient inference for LLMs, the community has mainly focused on model-centric or algorithm-centric approaches (Wan et al., 2023). Model-centric approaches, including quantization (Bhandare et al., 2019) and knowledge distillation (Hinton et al., 2015), aim to compress LLMs into smaller models while retaining the capabilities of the vanilla models. In contrast, algorithm-centric approaches, such as speculative decoding (Leviathan et al., 2023) and KV-Cache optimization (Zhang et al., 2023), aim to reduce latency and memory usage in sequence generation tasks. However, when processing batch queries in a single forward pass, these methods generally result in a significant increase in computational load, e.g., FLOPs, linear with the number of inputs.

The Multi-input Multi-output (MIMO) architecture (Ramé et al., 2021; Havasi et al., 2021; Murahari et al., 2022) has emerged as an effective approach for predicting multiple samples simultaneously in a single forward pass, incurring the computational cost of only a single input. This architecture requires jointly training a multiplexer and a demultiplexer alongside the entire base model: the multiplexer combines multiple inputs into one, and the demultiplexer separates the base model's outputs back into individual ones. Subsequent research (Murahari et al., 2023) applied MIMO-style training to enhance inference with LLMs, exemplified by BERT (Devlin et al., 2019). However, this method not only necessitates training the multiplexer and demultiplexer during pre-training phase but also requires fine-tuning the entire model, including BERT's parameters, thereby limiting its applicability to increasingly larger language models. Moreover, updating the backbone model's parameters necessitates maintaining multiple copies of the backbone model to accommodate dynamic inference budgets, further constraining its scalability.

In this paper, we explore MIMO training on a fixed LLM to improve its inference efficiency with-

*Corresponding authors.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 22072–22087



(a) Mini Batch Processing (b) DataMUX

(c) RevMUX

Figure 1: Illustration of our proposed RevMUX in comparison to traditional mini-batch processing and DataMUX.

out additional pre-training. A major challenge in implementing MIMO for fixed LLMs is to trace and preserve the uniqueness of each input, as the fixed LLMs can struggle to differentiate individual instances within the consolidated inputs, resulting in performance degradation (Murahari et al., 2023). Inspired by Reversible Neural Networks (Gomez et al., 2017; Behrmann et al., 2019), which split the input into two halves for parallel processing and enable reconstruction of lower-layer inputs from upper-layer outputs, we propose reversible adapters to achieve data multiplexing, dubbed RevMUX. The reversible multiplexer leans to map different samples into distinct feature spaces and the mapping function is shared with the demultiplexer to perform a reverse operation that dis-aggregates the output from the LLM for classification. We train these reversible adapters in a parameter-efficient manner (Lester et al., 2021) on downstream tasks and then apply them for batch inference.

Through extensive experiments on four datasets and three types of LLM backbones, we demonstrate the effectiveness of RevMUX in enhancing LLM inference efficiency while maintaining satisfactory classification performance. Notably, our RevMUX method, which freezes the entire backbone LLM, achieves performance comparable to or surpassing that of two fine-tuned baselines on $\text{BERT}_{\text{BASE}}$. We also extended our method to encoder-decoder architectures such as T5 and decoder-only architectures like LLaMA3-8B (Dubey et al., 2024). Results on all three architectures across five scales consistently show that the proposed reversible adapters significantly contribute to performance retention during data multiplexing. Moreover, the combined use of reversible adapters in both the multiplexing and demultiplexing processes creates a synergistic effect, amplifying performance benefits beyond those achieved by individual components.

2 Related Work

2.1 Efficient Inference for LLMs

The majority of recent efforts to enhance LLM inference efficiency have focused on either model-centric or algorithm-centric approaches.

Model-centric methods, also known as model compression techniques, aim to train smaller models that enable efficient inference while retaining the capabilities of the original, larger models. As summarized by Wan et al. (2023), recent model compression techniques for LLMs can be grouped into the following categories: (1) *Quantization*, which converts model weights and/or activations of high-precision data types (e.g., float32) into low-precision data types (e.g., int8) (Dettmers et al., 2023; Xiao et al., 2023; Shao et al., 2024); (2) *Parameter Pruning*, which removes redundant LLM weights (Ma et al., 2023; Frantar and Alistarh, 2023); and (3) *Knowledge Distillation*, which involves training a small student model to mimic the behavior of a large teacher model (Gu et al., 2024; Liu et al., 2024).

In contrast, algorithm-centric methods focus on optimizing the inference process through the design of time- or memory-efficient algorithms. For example, speculative decoding supports parallel token computation for autoregressive language models, thereby speeding up the decoding stage (Leviathan

et al., 2023; Santilli et al., 2023). Additionally, KV-Cache optimization, which reuses cached KV pairs, can reduce the computational cost of decoding (Zhang et al., 2023; Ge et al., 2024).

The above methods either compress the models or optimize the inference process but do not leverage data-specific strategies. When applied to batch queries in a single forward pass, they typically result in a significantly increased computational load, often proportional to the number of inputs. In contrast, our approach enhances inference efficiency through a data-centric strategy. We focus on data multiplexing techniques instead of modifying models or algorithms, allowing the model to perform batch inference with significantly reduced computational costs.

2.2 Multi-Input Multi-Output Training

To reduce both training and inference costs in ensemble models, the concept of Multi-Input Multi-Output (MIMO) (Havasi et al., 2021) has been introduced. MIMO enables the training of multiple independent subnetworks within a single network, thereby enhancing prediction robustness. This mechanism allows for multiple output predictions through a single forward pass, effectively simulating the ensemble process while conserving computational resources (Ramé et al., 2021; Sun et al., 2022a, 2024). Although previous MIMO works primarily focus on enhancing ensemble efficiency, their findings crucially substantiate the ability of deep neural networks to process multiple inputs in a single forward pass, laying the ground for subsequent works on data multiplexing.

Recent works have lent MIMO-style training to improve the batch inference efficiency of LLMs. Murahari et al. (2022) propose a data mixer to amalgamate multiple inputs and a corresponding demixer to disaggregate the combined output into individual ones. Specifically, within a batch of $N \times M$ instances, a multiplexing layer consolidates these $N \times M$ representations into M consolidated representations. Subsequently, the demultiplexing layer interprets these M outputs to generate predictions for the entire set of $N \times M$ instances. This approach, embodied in MIMONets (Menet et al., 2023), incorporates a distinctive key mechanism that serves not only to bind the inputs together but also facilitates their separation. Building upon this concept, MUX-PLMs (Murahari et al., 2023) have advanced the field by pre-training language models that leverage a contextual multiplexer coupled with an RSA demultiplexer, marking a significant step forward in the efficient inference on PLMs.

However, existing MIMO-style frameworks for LLM batch inference typically require end-to-end training, where the base model is trained alongside the data mixer and demixer. This would become impractical for large-scale language models due to their substantial size and complexity. Consequently, this paper focuses on scenarios where the backbone model is already trained and fixed, exploring strategies for effective data multiplexing without any additional pre-training.

3 Methodology

3.1 Overview of RevMUX

Given an input instance x and a LLM $f(\cdot)$, most existing LLM applications can be summarized as:

$$\hat{y} = f(x), \tag{1}$$

where $\hat{y}$ is the prediction. During the inference stage, take Figure 1 as an example, traditional mini-batch processing extends input vector into tensors to improve the throughput. DataMUX (Murahari et al., 2022) introduce a multiplexer to combine 32 input samples into 16 vectors and a demultiplexer to decompose 16 outputs to 32 labels, which saves the computational load because the LLM only infer "16 samples". Due to the challenge of fixing backbone LLM, the main difference between our proposed RevMUX and DataMUX are two folds:

Prefilling: The mixture of input samples may lead to a distribution shift (Navarro et al., 2024), which makes the gap on latent representation space between the backbone language model and the multiplexed input samples. Hereby the decision to tine-tune the backbone language model versus not fine-tuning it represents two distinct methodological approaches. Fine-tuning adapts the backbone language model to new tasks by mixing multiple input samples and learning their ralational representations. In contrast, not fine-tuning corresponds to learn how to mix the input samples by bridging the gap between different representation space. To tackle this problem, we use prefilling for transforming the feature space, to ensure the feature space becoming more similar to the feature space seen during pre-training.

Reversible Multiplex Adapter and Reverse Demultiplex Adapter: While combining feature vectors of multiple samples into a single vector can

reduce the computational load, such processing can result in significant information loss and model confusion. To preserve the distinction between different samples, it is essential to incorporate a traceable module. Such as module should effectively revert and separate the combine features, ensuring that each sample's unique characteristics are retained for accurate classification. Drawing inspiration from Reversible Neural Networks (Gomez et al., 2017; Behrmann et al., 2019), which divide the input into two halves to facilitate the reconstruction of lower layer activations from the upper layer outputs, we introduce reversible adapters to mix and demix different samples within a batch. These adapters are trained in a parameter-efficient manner on downstream tasks and are then employed for batch inference. In Section 3.2, we introduce our RevMUX pipeline when $N = 2$. Details of multiplexing layer and demultiplexing layer when $N > 2$ can be found in Appendix A.

3.2 The RevMUX Pipeline

3.2.1 Task-specific Backbone

Our work aims to address the problem where users, having a large language model already in place for their target task, seek to accelerate inference. In the initial step, it is crucial to have a backbone model capable of addressing the target task. In this paper, we selected T5 (Raffel et al., 2020) as the backbone to experimentally validate the effectiveness of our approach. Additionally, for comparison purposes, we also utilized BERT (Devlin et al., 2019) as the backbone in our comparative experiments. For T5, we fix the language model and use prompt tuning (Liu et al., 2022) to train a soft prompt, which simulates the scenarios that we cannot train a task-specific large language model. For BERT, we simply add a classfier and fine-tune the BERT to learn the task-specific backbone. For LLaMA3-8B (Dubey et al., 2024), we are not able to train the backbone, hereby we assume that the backbone is well trained and can be directly transferred to our classification tasks.

3.2.2 Prefilling

As shown in Section 3.1, the first step of RevMUX pipeline is prefilling. In this step, we convert the input instances $x_1, x_2, \cdots, x_N$ to dense representations, ensuring the feature space becoming more similar to the feature space seen during the backbone pre-training:

$$\mathbf{h}_k^l = \text{Encoder}_{0:l}(\mathbf{X}_k), \tag{2}$$

where l indicates that we use the first l encoder layers of the pre-trained language model for prefilling. For BERT and LLaMA, $\mathbf{X}_k = x_k$. For T5, $\mathbf{X}_k = \text{concat}[\mathbf{p}_0; x_k]$ where $\mathbf{p}_0$ is a pre-trained task-specific soft prompt.

3.2.3 Multiplexing Layer

With prefilling, we obtain N representations for N instances. Then we have a multiplexing layer $g : \mathbb{R}^{N\times d} \to \mathbb{R}^d$ to mix instances together, where d is the dimension of the backbone language model. As shown in Figure 1c, our multiplexing layer includes down projection and reversible multiplexer.

Down Projection Since the dimension of backbone language model is d and the representations of instances are under the space of $\mathbb{R}^{N\times d}$, we firstly employ a linear layer $f_{\text{down}} : \mathbb{R}^d \to \mathbb{R}^{\frac{d}{N}}$ as the down projection function:

$$\mathbf{i}_k^l = f_{\text{down}}(\mathbf{h}_k^l). \tag{3}$$

Reversible Multiplexer Motivated by Reversible Neural Network (Gomez et al., 2017) that the layers' activations can be reconstructed from the next layers, we employ a reversible multiplexer to combine the multiple input instances, which the demultiplexing layer can reconstruct.

As illustrated in Figure 2 when $N = 2$, we have:

$$\begin{aligned} \mathbf{o}_1^l &= \mathbf{i}_1^l + \mathcal{F}(\mathbf{i}_2^l), \\ \mathbf{o}_2^l &= \mathbf{i}_2^l + \mathcal{G}(\mathbf{o}_1^l), \\ \mathbf{o}^l &= \text{concat}[\mathbf{o}_1^l, \mathbf{o}_2^l], \end{aligned} \tag{4}$$

where $\mathcal{F}(\cdot)$ and $\mathcal{G}(\cdot)$ are learnable 2-layer MLP.

3.2.4 Language Modeling with Batched Instances

With the multiplexing layer, we obtain $\mathbf{o}_l$ that contains the representation of all batched instances. After that, we pass the batched representation throughout the backbone language model:

$$\hat{\mathbf{o}} = \text{Decoder}\Big(\text{Encoder}_{l+1:L}(\mathbf{o}^l)\Big), \tag{5}$$

where L is the number of encoder layers. Specially, for encoder-only backbone such as BERT, $\text{Decoder}(x) = x$.

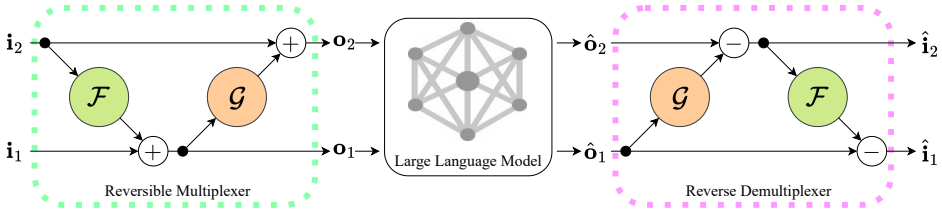


Figure 2: Illustration of the reversible multiplexer and reverse demultiplexer when $N = 2$.

3.2.5 Demultiplexing Layer

From Eq (5), we obtain the outputs of the language model. To demix the outputs, we have a demultiplexing layer $h : \mathbb{R}^d \to \mathbb{R}^{N\times d}$. Similar to the multiplexing layer mentioned in Section 3.2.3, our demultiplexing layer includes reverse demultiplexer and up projection.

Reverse Demultiplexer Given the necessity to decompose the language model's output to restore the outputs of the original samples, we employ a reversible multiplexer during the input content assembly process. Therefore, we use the reverse demultiplexer to decompose the output. Take $N = 2$ as an example, we have:

$$\begin{aligned}[\hat{\mathbf{o}}_1, \hat{\mathbf{o}}_2] &= \hat{\mathbf{o}}, \\ \hat{\mathbf{i}}_2 &= \hat{\mathbf{o}}_2 - \mathcal{G}(\hat{\mathbf{o}}_1), \\ \hat{\mathbf{i}}_1 &= \hat{\mathbf{o}}_1 - \mathcal{F}(\hat{\mathbf{i}}_2),\end{aligned} \tag{6}$$

where $\mathcal{F}(\cdot)$ and $\mathcal{G}(\cdot)$ share the same parameters with that in Eq (4).

Up Projection Considering that we obtain a $\frac{d}{N}$-dimensional sample representation through the reverse demultiplexer, it is necessary to employ an up projection to restore this representation to the original d-dimensional space for further processing by the backbone language model:

$$\hat{\mathbf{h}}_k = f_{\text{up}}(\hat{\mathbf{i}}_k), \tag{7}$$

where $k \in \{1, 2, \cdots, N\}$ that indicates the output of the k-th instance, $f_{\text{up}} : \mathbb{R}^{\frac{d}{N}} \to \mathbb{R}^d$ is a linear layer for up projection.

3.2.6 Prediction

The last step of RevMUX is prediction, which converts the output from the demultiplexing layer to target labels. For BERT, the encoder-only backbone, we reuse the trained task-specific classifier layer:

$$\hat{y}_k = \text{softmax}(W_c \hat{\mathbf{h}}_k), \tag{8}$$

where W_c is the task-specific parameter matrix trained in Section 3.2.1.

For T5 and LLaMA, we reuse the language model head to decode the target token and then use a verbalizer $\mathcal{V}$ to convert the target token to the target label:

$$\hat{y}_k = \mathcal{V}(\text{LM\_Head}(\hat{\mathbf{h}}_k)). \tag{9}$$

In summary, the overall framework can be abstracted as:

$$\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_N = h\Big(f\big(g(x_1, x_2, \cdots, x_N)\big)\Big), \tag{10}$$

where N indicates the number of mixed samples, $f(\cdot)$ indicates the backbone LLM, $g : \mathbb{R}^{N\times d} \to \mathbb{R}^d$ indicates the multiplexing layer, and $h : \mathbb{R}^d \to \mathbb{R}^{N\times d}$ indicates the demultiplexing layer. It is notably that traditional mini-batch processing of Eq (1) is a special case of Eq (10) under the condition of $N = 1$ and $g(x) = h(x) = x$.

3.3 Training Objectives

In this subsection, we will briefly introduce our training objectives.

Gold Label The first objective function is a task-specific loss function from gold label. In this paper, we use cross-entropy loss as:

$$\mathcal{L}_{\text{ce}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}), \tag{11}$$

where C is the number of labels.

InfoNCE On the other hand, considering the need to reconstruct the outputs of the original samples, the second objective function must impose constraints to ensure that the results obtained from multiplexed batch inference closely match those from the original one-by-one forward propagation. This ensures that the remaining components of the backbone language model function correctly. To address this problem, we employ Information

Noise-Contrastive Estimation (InfoNCE) (Oord et al., 2018) as the second objective function. InfoNCE is a loss function used in contrastive learning to maximize the mutual information between positive pairs of samples while minimizing it between negative pairs.

During the training stage, we compute the output representation by twice: one from the multiplexed batch inference, and the other from the original one-by-one forward pass. Within the same batch, we treat the output pairs corresponding to the same sample as positive examples and the remaining output pairs as negative examples. Hereby we compute the loss by:

$$\mathcal{L}_{\text{info}} = \sum_{k=1}^{N} \text{InfoNCE}(\hat{\mathbf{h}}_k, \mathbf{h}_k), \tag{12}$$

$$= \sum_{k=1}^{N} -\mathbb{E}[\log \frac{\exp(\hat{\mathbf{h}}_k \cdot \mathbf{h}_k)}{\exp(\hat{\mathbf{h}}_k \cdot \mathbf{h}_k) + \sum_{j \neq k}^{N} \exp(\hat{\mathbf{h}}_k \cdot \mathbf{h}_j)}]$$

where $\mathbf{h}_k = \text{LLM}(\mathbf{X}_k)$ is the output of one-by-one forward pass and $\hat{\mathbf{h}}_k$ is defined in Eq (7).

Thus, the overall objective is:

$$\mathcal{L} = \mathcal{L}_{\text{ce}} + \lambda \mathcal{L}_{\text{info}}, \tag{13}$$

where λ is the weight to control the importance of cross-entropy loss and the InfoNCE loss.

4 Experiments

4.1 Datasets and Evaluation Settings

We conduct experiments on four datasets across three tasks. For the sentiment classification task, we use SST-2 (Socher et al., 2013). For the paraphrase detection task, we use MRPC (Dolan and Brockett, 2005). For the natural language inference task, we use RTE (Wang et al., 2019) and QNLI (Wang et al., 2019). For fair comparisons with baseline methods, we use $\text{BERT}_{\text{BASE}}$ as the backbone. We further examined RevMUX on T5 across three different scales.

To better simulate real-world randomness, we conduct 10 tests for each model. In each test, we begin by dividing the samples into N distinct subsets. From each subset, we randomly select a sample to create a batch. This batch is then processed by the model. Given these testing parameters, it is possible for the same sample to yield varying prediction results across different tests. To account for this variability, we calculate the average of these multiple tests to serve as our final evaluation metric. This averaged metric is intended to represent the expected accuracy of the overall sample set in real-world inference scenarios. More details can be found in Appendix C.1.

4.2 Baselines

We consider the following baselines:

DataMUX (Murahari et al., 2022). A MIMO-style learning framework that trains a multiplexing layer to combine a group of N data samples into a single representation and a demultiplexing layer to separate this into N representations for classification. The two layers are typically linear layers trained together with the entire base model.

MUX-PLM (Murahari et al., 2023). Also a MIMO-style learning framework, particularly designed for enhancing the throughput for a pre-trained LLM. MUX-PLM requires training the multiplexing and demultiplexing layers during the pre-training stage for $\text{BERT}_{\text{BASE}}$ to learn the new task of "combining multiple input samples". In the experiment section, we use $\text{MUX-BERT}_{\text{BASE}}$ to indicate this baseline for clarity.

Vanilla Adapters It directly applies a vanilla three-layer Multilayer Perception (MLP) for multiplexing and demultiplexing respectively, akin to DataMUX. This baseline examines the effectiveness of the reversible design in RevMUX.

Only Multiplexer Reversible It keeps the reversible multiplexer of RevMUX but replaces its demultiplexer with a vanilla three-layer MLP. This baseline empirically examines whether the demultiplexer of RevMUX can restore individual inputs.

5 Results and Analysis

5.1 Comparison with Baselines

For a fair comparison with previous methods that involve fine-tuning the backbone model, we first experiment on $\text{BERT}_{\text{BASE}}$ (110M) and also report the fine-tuned results, as shown in Table 1.

(1) **RevMUX retains performance stably**: Overall, RevMUX (🔥) consistently outperforms $\text{MUX-BERT}_{\text{BASE}}$ (🔥) ($p = 0.015$) and DataMUX (🔥) ($p = 0.02$) across all four datasets. More importantly, our RevMUX (❄), which freezes the entire backbone LLM, achieves comparable or superior performance to the two fine-tuned baselines, albeit with some sacrifice in inference efficiency. Notably,

	Model	N	↗	Tuned	Params	SST-2	MRPC	RTE	QNLI	Avg. Score
Backbones	$\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019)	1	-	🔥	110M	92.20	87.01	62.96	90.55	83.18
	$\text{MUX-BERT}_{\text{BASE}}$ (Murahari et al., 2023)	1	100%	🔥	112M	91.74	87.75	63.18	90.54	83.30
Baselines	DataMUX (Murahari et al., 2022)	2	180%	🔥	166M	90.50	85.05	60.87	88.39	81.20
	$\text{MUX-BERT}_{\text{BASE}}$ (Murahari et al., 2023)	2	201%	🔥	112M	90.62	83.77	58.19	88.17	80.19
Ours	Vanilla Adapters	2	156%	❄️	16.53M	90.42	84.78	60.06	88.19	80.86
	Only Multiplexer Reversible	2	161%	❄️	20.07M	90.65	84.60	60.41	88.14	80.95
	RevMUX (❄️)	2	154%	❄️	9.45M	90.85	85.06	60.72	88.25	81.22
	RevMUX (🔥)	2	154%	🔥	120M	**91.21**	**85.78**	**61.41**	**88.72**	**81.78**

Table 1: Model comparison using $\text{BERT}_{\text{BASE}}$ as backbone model. "🔥" indicates fine-tune the BERT, "❄️" indicates freeze the BERT as feature extraction only. "Params" is the number of learnable parameters. Best results in **bold** and the second-best in underline. Inference speedups (↗) are reported against the $N = 1$ setting.

RevMUX (❄️) outperforms $\text{MUX-BERT}_{\text{BASE}}$ (🔥) which requires an additional pre-training stage ($p = 0.166$). These results highlight RevMUX's advantage in retaining classification performance during data multiplexing.

(2) **No fine-tune scenario is significantly more challenging**: Comparing RevMUX (❄️) with RevMUX (🔥), it is clear that finetuning the backbone LLM significantly enhances performance across all the datasets ($p < 0.01$). As the task is very challenging, fine-tuning LLMs proves to bring limited gains.

(3) **Components in RevMUX are effective**: Moreover, RevMUX (❄️) surpasses Vanilla Adapters (❄️), highlighting the effectiveness of reversible design in boosting accuracy. Vanilla Adapters (❄️) performed similarly to Only Multiplexer Reversible (❄️), suggesting that the reversible multiplexer alone offers limited benefits. The effectiveness of RevMUX (❄️) lies in the synergy between the reversible multiplexer and reverse demultiplexer, as shown by comparing RevMUX (❄️) against Only Multiplexer Reversible (❄️).

(4) **The trade-off between efficiency and accuracy**: Apart from accuracy, we also measure the total number of FLOPS required for each model to infer all four validation sets. For a fair comparison, we fix the batch size as 32 and the sequence length as 128. We compute the speedups (↗) against the baseline $\text{MUX-BERT}_{\text{BASE}}(N = 1)$, reported in the column ↗ in Table 1. We observe that $\text{MUX-BERT}_{\text{BASE}}(N = 2)$, halved the FLOPs consumption, achieving a speedup of 201% while our RevMUX achieved speedups ranging between 154% and 161%, demonstrating a trade-off between model accuracy and efficiency. We present the results in Figure 3, where the blue line indicates that the baseline model accuracy decreases as efficiency increases. For a given efficiency target, RevMUX (🔥) and DataMUX (🔥) are clearly above the blue line but RevMUX (🔥) results in higher accuracy, indicating that reversibility can help preserve the classification performance.

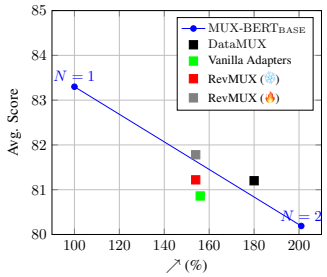


Figure 3: Trade-off between inference efficiency and model accuracy.

5.2 Scalability Tests on Larger Models

5.2.1 Encoder-Decoder Architecture

In this section, we focus on evaluating our proposed parameter-efficient RevMUX (❄️) on larger language models, specifically on T5. We conduct experiments on T5 with three model sizes: T5_{Small} (60M), T5_{Base} (220M), and T5_{Large} (770M). For each task, we use prompt tuning (Lester et al., 2021) to adapt each model to the task domain in advance and then train RevMUX for inference acceleration. The results are presented in Table 2, and we highlight the following observations:

(1) **RevMUX retains performance stably while improving efficiency**: We use the result of fine-tuning the entire backbone on each dataset and inference with a single input ($N = 1$) as the reference point. When inference with two inputs simultaneously ($N = 2$), RevMUX achieves about 45% speedups across all scales, while at the same

Backbone	Model	N	Tuned	↗	SST-2	MRPC	RTE	QNLI	Avg. Score
T5_{Small}	Task-specific Backbone	1	🔥	100%	90.34	84.31	64.62	89.34	82.15
	Vanilla Adapters	2	❄	138%	89.00	81.72	57.22	85.36	78.33
	Only Multiplexer Reversible	2	❄	146%	89.04	82.30	57.51	85.44	78.57
	RevMUX	2	❄	145%	**89.14**	**82.45**	**60.22**	**85.63**	**79.36**
T5_{Base}	Task-specific Backbone	1	🔥	100%	94.56	87.50	82.31	92.93	89.33
	Vanilla Adapters	2	❄	140%	92.36	82.94	63.28	87.58	81.54
	Only Multiplexer Reversible	2	❄	144%	92.54	83.19	64.01	88.14	81.98
	RevMUX	2	❄	144%	**92.70**	**83.80**	**64.73**	**88.65**	**82.47**
T5_{Large}	Task-specific Backbone	1	🔥	100%	95.96	90.44	87.36	93.94	91.93
	Vanilla Adapters	2	❄	141%	92.58	83.16	64.22	88.42	82.10
	Only Multiplexer Reversible	2	❄	143%	92.67	83.46	64.43	88.56	82.28
	RevMUX	2	❄	143%	**92.81**	**83.86**	**65.01**	**88.89**	**82.64**

Table 2: T5 results on the four datasets of GLUE benchmark. "🔥" indicates parameter-efficient fine-tune the T5, "❄" indicates freeze the whole backbone while training the adapters.

time, maintaining a satisfactory classification accuracy. This observation holds across the datasets and model scales, demonstrating the generalizability and scalability of RevMUX.

(2) **Both the reversible multiplexer and reverse demultiplexer are effective**: The findings with the batch inference results ($N = 2$) are consistent with those on $\text{BERT}_{\text{BASE}}$. The comparisons between RevMUX and Vanilla Adapters provide strong empirical evidence for the effectiveness of the reversible design in retaining performance. Furthermore, RevMUX consistently surpasses the Only Multiplexer Reversible method in all scenarios, highlighting the synergistic effect of the reversible multiplexer and the reverse demultiplexer.

(3) **The efficiency-performance trade-off is more pronounced for larger backbones**: The efficiency-performance trade-off is a well-known challenge in the community. Our experiments across various backbone sizes provide empirical evidence that, with a data multiplexing approach, larger backbones experience greater performance compromises in exchange for improved efficiency. Apart from QNLI, the amount of performance degradation on the other datasets follows the trend: $\text{T5}_{\text{Large}} > \text{T5}_{\text{Base}} > \text{T5}_{\text{Small}}$.

5.2.2 Decoder-Only Architecture

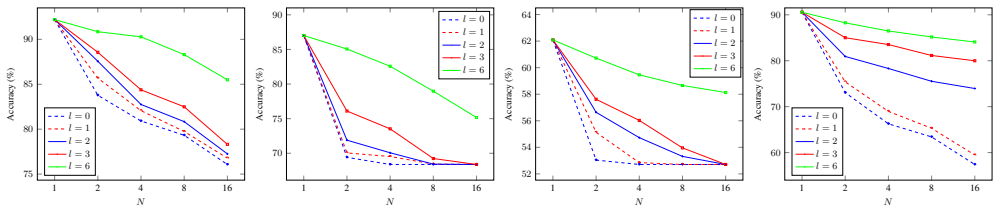
We now shift our focus to evaluating RevMUX (❄) on larger decoder-only language models, specifically LLaMA3-8B. Unlike our previous study with T5, we do not pre-adapt LLaMA3 using prompt tuning. Instead, we focus on zero-shot inference, which is commonly employed in billion-scale LLMs. In this study, we assess how RevMUX enhances inference efficiency in a zero-shot context. For each task, we curate a manual prompt and directly train RevMUX on top of LLaMA3 for inference. Additional details can be found in Appendix D. Based on the results presented in Table 3, we draw the following key observations:

(1) **RevMUX is scalable to billion-scale decoder-only LLMs**: Similar to the outcomes observed with $\text{BERT}_{\text{BASE}}$ and three T5 models, both the reversible multiplexer and the reverse demultiplexer demonstrate significant effectiveness when applied to LLaMA3-8B.

(2) **RevMUX significantly enhances both performance and inference efficiency**: Compared to Zero-Shot Prompting, RevMUX demonstrates a twofold increase in inference efficiency and improves accuracies by approximately $2\% - 10\%$ across the four datasets. Unlike the previous experiment, which established a strong baseline by training soft prompts for task domain adaptation, this study employs a manual prompt with a frozen LLaMA3 and demonstrates a clear overall performance gain brought by RevMUX. Our results suggest that during the training of reversible adapters, RevMUX also effectively learns to preserve the discriminative information that is helpful for classification tasks.

5.3 Model Analysis and More Studies

We analyze the performance and inference efficiency of RevMUX (❄) by varying the number of prefilling layers l, the batch size N, and the impact of InfoNCE loss λ. We use $\text{BERT}_{\text{BASE}}$ and report accuracy on MRPC and RTE in Figure 4 and speedups (↗) on SST-2 in Table 4.



(a) SST-2 (b) MRPC (c) RTE (d) QNLI

Figure 4: Results of different l and N on $\text{BERT}_{\text{BASE}}$.

Backbone	Model	N	SST-2	MRPC	RTE	QNLI	Avg. Score
Llama3-8B	Zero-Shot Prompting	1	92.64	70.10	72.92	76.99	78.16
	Vanilla Adapters	2	94.01	80.96	82.72	85.99	85.92
	Only Multiplexer Reversible	2	94.09	81.08	82.82	86.24	86.06
	RevMUX	2	**94.38**	**81.30**	**83.18**	**86.53**	**86.35**

Table 3: Llama3-8B results on the four datasets of GLUE benchmark.

l	0	1	2	3	6
↗	207%	198%	189%	181%	154%

Table 4: Inference efficiency improvement with different prefilling layers on SST-2 with $\text{BERT}_{\text{BASE}}$.

The impact of N and l on performance: Figure 4 shows a clear downward trend in classification accuracy as N increases. This is anticipated, as mixing more samples in a batch makes it more difficult for RevMUX to preserve the individual distinctiveness given the limited capacity of the reversible modules, $\mathcal{F}$ and $\mathcal{G}$. However, with a sufficient number of prefilling layers (e.g., $l = 6$), the model can maintain relatively high accuracy even when N is increased to 16. This suggests that increasing the number of prefilling layers is an effective strategy to enhance model performance, allowing it to sustain accuracy despite larger N values. More studies can be found in Table 10 in the Appendix.

The impact of prefilling on efficiency: While Figure 4 indicates that increasing the number of prefilling layers enhances classification accuracy, it also raises a concern about inference efficiency. As shown in Table 4, increasing the number of prefilling layers to 6 can reduce the speedup by 50% compared to not using any prefilling. However, as higher layers in LLMs typically provide a better representation space that may help in distinguishing different samples, choosing the optimal number of prefilling layers remains a trade-off to balance accuracy and efficiency.

6 Conclusion

In this paper, we introduce RevMUX, a parameter-efficient data multiplexing framework designed to enhance the batch inference efficiency of LLMs. RevMUX features a reversible multiplexer that combines multiple samples, allowing the demultiplexer to reverse this process and restore individual outputs for classification. We train RevMUX on downstream tasks while keeping the backbone LLM frozen, and apply it for batch inference. Extensive experiments on BERT-base, T5 across three scales, and LLaMA3-8B demonstrate the effectiveness of RevMUX in enhancing both accuracy and efficiency. Ablation studies confirm the synergistic function of the reversible multiplexer and the reverse demultiplexer.

Acknowledgements

This research is supported, in part, by the Joint NTU-WeBank Research Centre on Fintech (Award No. NWJ-2020-007), Nanyang Technological University, Singapore. This research is also supported, in part, by the National Research Foundation, Prime Minister's Office, Singapore under its NRF Investigatorship Programme (NRFI Award No. NRF-NRFI05-2019-0002). Xu Guo thanks Wallenberg-NTU Presidential Postdoctoral Fellowship. Zhiwei Zeng thanks the support from the Gopalakrishnan-NTU Presidential Postdoctoral Fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore.

Limitations

While RevMUX presents a promising step forward in improving LLM inference efficiency, several limitations need to be acknowledged.

Inference efficiency-performance trade-offs: Although RevMUX effectively improves inference efficiency, there is an inherent trade-off with potential loss in inference performance. While our experiments show that RevMUX can largely retain a satisfactory classification performance in the majority of scenarios, the performance compromises could vary with different datasets or tasks. For instance, we observe that the efficiency-performance trade-off is more pronounced on the RTE dataset with $\mathrm{T5}_{\mathrm{Large}}$ and $\mathrm{T5}_{\mathrm{Base}}$. This may be attributed to the inherent complexity and nuances of the RTE dataset, and the underlying causes warrant further investigation.

Potential for bias and fairness issues: As with many other AI and ML methods, there is a risk that the multiplexing strategy could inadvertently amplify existing biases in the data. Proper handling of fairness and bias relation issues in data multiplexing remains an area requiring further investigation.

Further empirical evidence on scalability: While RevMUX shows promise in enhancing efficiency, its scalability for extremely large-scale deployments or real-time applications needs thorough evaluation. Our experimental results suggest that larger backbones tend to experience greater performance compromises to gain efficiency. Understanding how RevMUX scales with even larger model sizes and deployment contexts is critical for broader applications.

Ethics Statement

In conducting this research, we have adhered to ethical standards and have not introduced any new ethical concerns:

- **Data usage**: We did not release any new datasets as part of this study. All datasets used are publicly available or have been licensed for academic purposes. We ensure compliance with the data usage policies of these sources.
- **Codes and artefacts**: The source code for baselines and other artefacts employed in our study are either open-sourced or licensed for academic use.
- **Transparency and accountability**: We strive for transparency in our research. All results and methodologies are clearly documented, and we encourage replication and scrutiny by the research community.

References

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. 2019. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 573–582. PMLR.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*. Asian Federation of Natural Language Processing.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*.

Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. 2017. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2214–2224.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran. 2021. Training independent subnetworks for robust prediction. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3045–3059. Association for Computational Linguistics.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Chengyuan Liu, Fubang Zhao, Kun Kuang, Yangyang Kang, Zhuoren Jiang, Changlong Sun, and Fei Wu. 2024. Evolving knowledge distillation with large language models and active learning. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 6717–6731. ELRA and ICCL.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 61–68. Association for Computational Linguistics.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Nicolas Menet, Michael Hersche, Geethan Karunaratne, Luca Benini, Abu Sebastian, and Abbas Rahimi. 2023. MIMONets: Multiple-input-multiple-output neural networks exploiting computation in superposition. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Vishvak Murahari, Ameet Deshpande, Carlos E. Jimenez, Izhak Shafran, Mingqiu Wang, Yuan Cao, and Karthik Narasimhan. 2023. MUX-PLMs: Data multiplexing for high-throughput language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 4540–4554. Association for Computational Linguistics.

Vishvak Murahari, Carlos E. Jimenez, Runzhe Yang, and Karthik Narasimhan. 2022. DataMUX: Data multiplexing for neural networks. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Madeline Navarro, Camille Little, Genevera I Allen, and Santiago Segarra. 2024. Data augmentation via subgroup mixup for improving fairness. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7350–7354. IEEE.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Alexandre Ramé, Rémy Sun, and Matthieu Cord. 2021. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 803–813. IEEE.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 12336–12355. Association for Computational Linguistics.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. 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, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

Rémy Sun, Clément Masson, Gilles Hénaff, Nicolas Thome, and Matthieu Cord. 2024. Semantic augmentation by mixing contents for semi-supervised learning. *Pattern Recognit.*, 145:109909.

Rémy Sun, Alexandre Ramé, Clément Masson, Nicolas Thome, and Matthieu Cord. 2022a. Towards efficient feature sharing in MIMO architectures. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2022, New Orleans, LA, USA, June 19-20, 2022*, pages 2696–2700. IEEE.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022b. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 20841–20855. PMLR.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, et al. 2023. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Appendix

A RevMUX ($N > 2$)

Similar with the RevMUX pipeline when $N = 2$, the pipeline of RevMUX ($N > 2$) has a slightly different multiplexer and demultiplexer to adapt the condition of N.

A.1 Reversible Multiplexer

In order to keep the reversible design, when $N > 2$, we can expand the Eq (4) as:

$$\begin{aligned}
\mathbf{o}_1^l &= \mathbf{i}_1^l + \mathcal{F}_1(\mathbf{i}_N^l), \\
\mathbf{o}_2^l &= \mathbf{i}_2^l + \mathcal{F}_2(\mathbf{o}_1^l), \\
\mathbf{o}_3^l &= \mathbf{i}_3^l + \mathcal{F}_3(\mathbf{o}_2^l), \\
&\cdots \\
\mathbf{o}_N^l &= \mathbf{i}_N^l + \mathcal{F}_N(\mathbf{o}_{N-1}^l), \\
\mathbf{o}^l &= \text{concat}[\mathbf{o}_1^l, \mathbf{o}_2^l, \mathbf{o}_3^l, \cdots, \mathbf{o}_N^l].
\end{aligned} \tag{14}$$

It is notably that $\mathcal{F}(\cdot)$ and $\mathcal{G}(\cdot)$ in Eq. (4) is the same as $\mathcal{F}_1(\cdot)$ and $\mathcal{F}_2(\cdot)$ in Eq (14).

Dataset	# Labels	# Train samples	# Evaluation samples
SST-2	2	67,349	872
MRPC	2	3,668	408
QNLI	2	104,743	5,463
RTE	2	2,490	277

Table 5: Summary statistics of four datasets from GLUE benchmark. We evaluate all models on the development set of all datasets.

A.2 Reverse Demultiplexer

Similar with Eq (14), we expand the Eq (6) when $N > 2$ as:

$$
\begin{aligned}
[\hat{\mathbf{o}}_1, \hat{\mathbf{o}}_2, \hat{\mathbf{o}}_3, \cdots, \hat{\mathbf{o}}_N] &= \hat{\mathbf{o}}, \qquad (15)\\
\hat{\mathbf{i}}_N &= \hat{\mathbf{o}}_N - \mathcal{F}_N(\hat{\mathbf{o}}_{N-1}),\\
\hat{\mathbf{i}}_{N-1} &= \hat{\mathbf{o}}_{N-1} - \mathcal{F}_{N-1}(\hat{\mathbf{o}}_{N-2}),\\
&\cdots\\
\hat{\mathbf{i}}_1 &= \hat{\mathbf{o}}_1 - \mathcal{F}_1(\hat{\mathbf{i}}_N).
\end{aligned}
$$

In summary, Eq (4) and Eq (6) are the special case ($N = 2$) of Eq (14) and Eq (15), respectively.

B Datasets

The detailed statistics of the datasets is shown in Table 5. We used four datasets from the GLUE benchmark to evaluate our models. The SST-2 dataset, with 67,349 training samples and 872 evaluation samples, is used for binary sentiment classification, labelling sentences as either positive or negative. The MRPC dataset consists of 3,668 training samples and 408 evaluation samples, involving sentence pairs labelled to indicate whether they are paraphrases of each other. The QNLI dataset includes 104,743 training samples and 5,463 evaluation samples, where the task is to determine if a given sentence correctly answers a question, derived from the Stanford Question Answering Dataset (SQuAD). Lastly, the RTE dataset, with 2,490 training samples and 277 evaluation samples, involves binary classification to determine whether one sentence entails another.

C Performance Testing

C.1 Testing Rounds

It is important to note that RevMUX does not adhere to the commutative property. For instance, $\mathrm{RevMUX}(x_1, x_2)$ is not necessarily equal to $\mathrm{RevMUX}(x_2, x_1)$. Unlike conventional mini-batch processing pipelines that eliminate randomness during inference, RevMUX is sensitive to the order of inputs. As a result, the same input instance can yield different predictions depending on the testing order. Therefore, to achieve a more robust and accurate evaluation, it is essential to assess RevMUX across multiple rounds, with varying input sequences (e.g., $[x_1, x_2, x_3, x_4]$ versus $[x_1, x_4, x_3, x_2]$).

To empirically explore the appropriate number of testing rounds, we fixed a RevMUX configuration and evaluated the model across multiple rounds, recording the cumulative distribution function (CDF) of accuracy. As illustrated in Figure 5, we observe that as the number of testing rounds, t, increases, the distribution of the model evaluation accuracy becomes smoother. Our analysis suggests that $t = 100$ provides a sufficiently robust evaluation. However, it is notable that the CDF curve for $t = 10$ closely approximates that of $t = 100$. Therefore, we selected $t = 10$ for our evaluations to achieve a balance between efficiency and accuracy.

Dataset	SST-2	MRPC	RTE	QNLI
T-statistic	-2.833	-3.712	-73.688	-5.603
p-value	0.011	< 0.005	< 0.0001	< 0.0001

Table 6: T-statistic and the p-value of RevMUX (🔥) outperforms RevMUX (❄).

C.2 The Effect of Fine-tuning on Performance during Data Multiplexing

To assess the impact of fine-tuning versus not fine-tuning the $\mathrm{BERT}_{\mathrm{BASE}}$ backbone on the performance, we conducted a t-test to evaluate the significance of fine-tuning (RevMUX 🔥) versus not fine-tuning (RevMUX ❄). As shown in Table 6, the results indicate significant performance improvements with fine-tuning the backbone model across all datasets. For SST-2 and MRPC, the p-values (0.011 and < 0.005, respectively) and negative t-statistics (-2.833 and -3.712) demonstrate that fine-tuning yields superior accuracy. The RTE dataset shows an exceptionally high t-statistic of -73.688 with a p-value <0.0001, highlighting a dramatic performance boost with fine-tuning. Similarly, for QNLI, the strong negative t-statistic of -5.603 and a p-value <0.0001 confirm the advantages of fine-tuning.

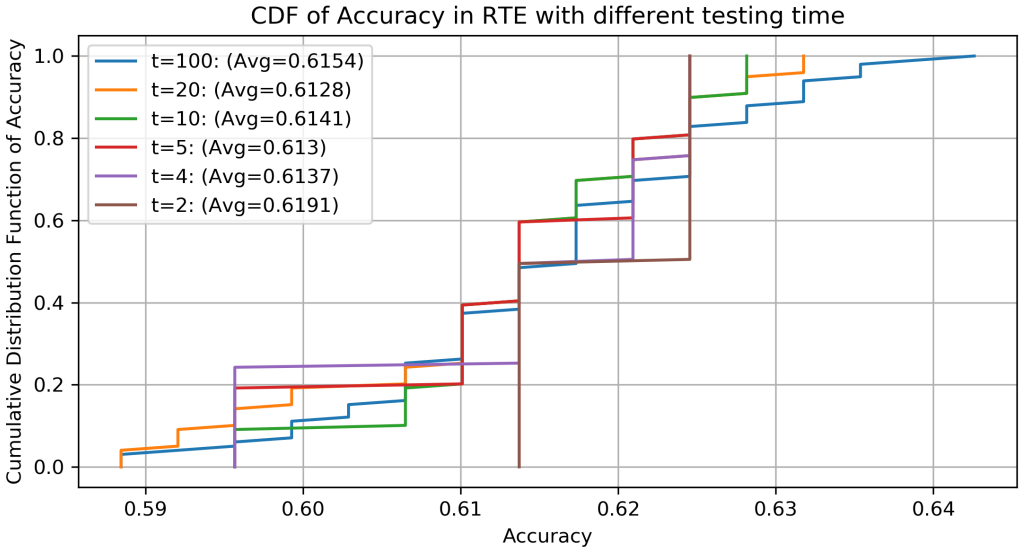


Figure 5: CDF of testing times t.

Dataset	Template
SST-2	**User**: You are require to predict the sentiment (positive or negative) to the following sentence. You should response positive or negative, only one token is accepted. **User**: <|start of the sentence|>: <sentence> <|end of the sentence|>. **Assistant**: **?**
RTE, MRPC, QNLI	**User**: You are require to predict the two following sentences are entailment or not (yes or no). You should response yes or no, only one token is accepted. **User**: <|start of the sentence1|>: <sentence1> <|end of the sentence1|> **User**: <|start of the sentence2|>: <sentence2> <|end of the sentence2|> **Assistant**: **?**

Table 7: Chat template for LLaMA3-8B-Instruct. Here "<sentence>" indicates single-sentence classification, "<sentence1>" and "<sentence2>" indicate the pair-wised sentence classification.

D Scalability Test

D.1 Scaling to Larger Model Size

D.1.1 Backbone Selection

To evaluate the effectiveness of RevMUX on larger backbone models, we selected the recently released and well-known open-source LLM, LLaMA3. Due to limited computational resources, we opted for the 8B model variant, which can be trained on a V100 GPU with 32GB of memory. To maintain consistency with the pre-training scenarios, we employed a chat template. Based on these considerations, we selected LLaMA3-8B-Instruct as the backbone for our experiments.

D.1.2 Implemtation Details

Given that SST-2 is a single-sentence classification task, while RTE, MRPC, and QNLI are pairwise sentence classification tasks, we utilized two different chat templates, as illustrated in Table 7. To simplify the verbalizer for answer prediction, we imposed a constraint that "only one token is accepted" and selected the language head prediction of the final token as the prediction for LLaMA. By applying this chat template for zero-shot transfer, the results presented in Table 3 validate the effectiveness of our approach.

D.2 Scaling to Larger N

To explore the scalability of RevMUX with varying values of N, we conduct a comparative experiment against MUX-PLM using the $\text{BERT}_{\text{BASE}}$ backbone. The results, presented in Table 10, lead to the following key observations:

(1) **RevMUX outperforms** MUX-PLM **when** $N = 2$: Under a fair comparison, RevMUX achieves an average score of 81.22 when $N = 2$,

	SST-2	MRPC	RTE	QNLI	Avg. Score
With InfoNCE	**89.14**	**82.45**	**60.22**	**85.63**	**79.36**
w.o. InfoNCE	89.03	82.11	58.45	85.40	78.75

Table 8: Ablation study results on with vs without InfoNCE loss on $\mathrm{T5}_{\mathrm{Small}}$.

	SST-2	MRPC	RTE	QNLI	Avg. Score
With InfoNCE	**90.85**	**85.06**	**60.72**	**88.25**	**81.22**
w.o. InfoNCE	90.58	84.04	58.59	87.85	80.27

Table 9: Ablation results about with vs without InfoNCE loss on $\mathrm{BERT}_{\mathrm{BASE}}$.

surpassing the 80.19 score of MUX-PLM.

(2) **RevMUX maintains comparable or superior performance with larger N values**: Notably, as N increases, RevMUX continues to demonstrate its scalability. For instance, the average score of RevMUX with $N = 8$ (77.78) is comparable to that of MUX-PLM with $N = 5$ (77.92). Furthermore, RevMUX with $N = 16$ achieves a higher average score (75.72) than MUX-PLM with $N = 10$ (75.61), highlighting the effectiveness and scalability potential of RevMUX.

E Hyperparameter Analysis

E.1 Impacts of λ for InfoNCE Loss

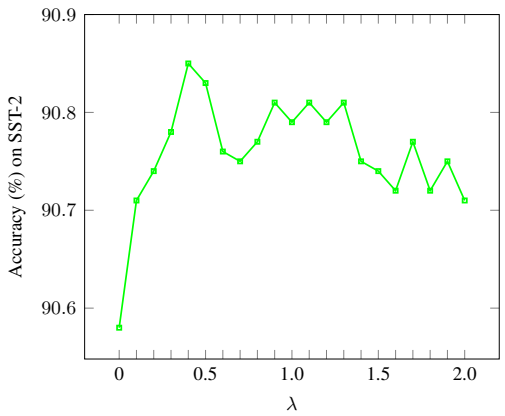


Figure 6: The impact of different λ for InfoNCE loss under the $\mathrm{BERT}_{\mathrm{BASE}}$ backbone.

Impacts of InfoNCE Loss: In order to explore the effectiveness of InfoNCE in our framework, we conduct ablation studies about with and without InfoNCE Loss with $\mathrm{BERT}_{\mathrm{BASE}}$ backbone. As shown in Table 9, the InfoNCE loss improves the average score from 80.27 to 81.22, demonstrates the effectiveness of the objective. More detailed analysis of the InfoNCE loss can be found in Appendix E.1.

In this section, we extend our experiments to further investigate the impact of the InfoNCE loss.

As shown in Table 8, we observe that incorporating the InfoNCE loss leads to improvements across all four datasets using the $\mathrm{T5}_{\mathrm{Small}}$ backbone. This aligns with the findings from the $\mathrm{BERT}_{\mathrm{BASE}}$ backbone discussed in Section 5.3, demonstrating the consistent effectiveness of the InfoNCE loss.

To gain deeper insights, we also conduct experiments varying the value of λ in Eq (13). As illustrated in Figure 6, we find that a value around 0.5 yields the best performance, and we adopt this setting for the subsequent experiments.

F Inference Efficiency Comparison

To compare inference efficiency, we report the FLOPs required for validation set inference. For a fair comparison, we set the batch size to 32 and the sequence length to 128, following the methodology of (Murahari et al., 2023). The efficiency improvement, denoted in column ↗, is calculated based on the average FLOPs used across all four datasets. The results are presented in Table 11 and Table 12.

Based on the inference efficiency results presented in Table 11, we evaluated various models using $\mathrm{BERT}_{\mathrm{BASE}}$ as the backbone. RevMUX (❄), despite achieving comparable efficiency, shows slightly higher average FLOPs (33.713 T) compared to DataMUX (28.799 T) and MUX-$\mathrm{BERT}_{\mathrm{BASE}}$ (25.834 T).

Based on the inference efficiency results presented in Table 12, using T5 as the backbone model, RevMUX achieves about 45% speedups across all scales. RevMUX shows average FLOPs of 8.188 T, 34.532 T, and 119.588 T on $\mathrm{T5}_{\mathrm{Small}}$, $\mathrm{T5}_{\mathrm{Base}}$, and $\mathrm{T5}_{\mathrm{Large}}$, respectively. The speed-up percentages on different T5 backbones are roughly around 140%, ranging from 138% to 144%.

Model	N	Tuned	SST-2	MRPC	RTE	QNLI	Avg. Score
MUX-PLM	1	🔥	91.74	87.75	63.18	90.54	83.30
RevMUX	2	❄️	90.85	85.06	60.72	88.25	81.22
MUX-PLM	2	🔥	90.62	83.77	58.19	88.17	80.19
RevMUX	4	❄️	90.28	82.57	59.46	86.48	79.70
MUX-PLM	5	🔥	86.88	80.10	59.13	85.58	77.92
RevMUX	8	❄️	88.30	78.97	58.66	85.17	77.78
MUX-PLM	10	🔥	83.44	78.63	58.27	82.08	75.61
RevMUX	16	❄️	85.50	75.17	58.13	84.08	75.72

Table 10: Model comparison of RevMUX and MUX-PLM (Murahari et al., 2023) using $\text{BERT}_{\text{BASE}}$ as backbone model with different N.

	Model	N	↗	Tuned	SST-2	MRPC	RTE	QNLI	Avg. FLOPs
Backbones	MUX-$\text{BERT}_{\text{BASE}}$ (Murahari et al., 2023)	1	100%	🔥	25.824	11.477	7.651	162.593	51.886
Baselines	DataMUX (Murahari et al., 2022)	2	180%	🔥	13.866	6.400	4.267	90.664	28.799
	MUX-$\text{BERT}_{\text{BASE}}$ (Murahari et al., 2023)	2	201%	🔥	12.439	5.741	3.827	81.330	25.834
Ours	Vanilla Adapters	2	156%	❄️	16.545	7.741	5.263	103.663	33.303
	Only Multiplexer Reversible	2	161%	❄️	16.019	7.495	5.096	100.363	32.243
	RevMUX	2	154%	❄️	16.749	7.837	5.328	104.938	33.713

Table 11: Inference efficiency comparison using $\text{BERT}_{\text{BASE}}$ as backbone model. (Unit: T FLOPs)

Backbone	Model	N	Tuned	↗	SST-2	MRPC	RTE	QNLI	Avg. FLOPs
T5_{Small}	Task-specific Backbone	1	🔥	100%	5.919	2.770	1.880	37.084	11.913
	Vanilla Adapters	2	❄️	138%	4.293	2.008	1.366	26.891	8.640
	Only Multiplexer Reversible	2	❄️	146%	4.058	1.899	1.291	25.424	8.168
	RevMUX	2	❄️	145%	4.068	1.903	1.294	25.487	8.188
T5_{Base}	Task-specific Backbone	1	🔥	100%	24.689	11.552	7.843	154.677	49.690
	Vanilla Adapters	2	❄️	140%	17.660	8.263	5.618	110.644	35.546
	Only Multiplexer Reversible	2	❄️	144%	17.133	8.016	5.451	107.344	34.486
	RevMUX	2	❄️	144%	17.156	8.027	5.458	107.486	34.532
T5_{Large}	Task-specific Backbone	1	🔥	100%	84.782	39.668	26.932	531.149	170.633
	Vanilla Adapters	2	❄️	141%	60.308	28.218	19.187	377.854	121.392
	Only Multiplexer Reversible	2	❄️	143%	59.372	27.777	18.888	371.987	119.506
	RevMUX	2	❄️	143%	59.412	27.798	18.901	372.239	119.588

Table 12: Inference efficiency comparison using T5 as backbone model. (Unit: T FLOPs)