MUX-PLMs: Data Multiplexing for High-throughput Language Models

Vishvak Murahari[1] **Ameet Deshpande**[1] **Carlos E. Jimenez**[1]
Izhak Shafran[2] **Mingqiu Wang**[2] **Yuan Cao**[2] **Karthik Narasimhan**[1]

[1]Princeton University [2] Google Brain
murahari@cs.princeton.edu

Abstract

The widespread adoption of large language models such as ChatGPT and Bard has led to unprecedented demand for these technologies. The burgeoning cost of inference for ever-increasing model sizes coupled with hardware shortages has limited affordable access and poses a pressing need for efficiency approaches geared towards high throughput and performance. Multi-input multi-output (MIMO) algorithms such as data multiplexing, offer a promising solution with a many-fold increase in throughput by performing inference for multiple inputs at the cost of a single input. Yet these approaches are not currently performant enough to be deployed in modern systems. We change that by developing MUX-PLMs, a class of deployable high throughput pre-trained language models (PLMs) trained with data multiplexing, that can be fine-tuned on any downstream task. Our novel multiplexing and demultiplexing modules proficiently entangle and disentangle inputs, and enable high-performance high throughput MUX-PLMs that are competitive with vanilla PLMs while achieving 2x/5x inference speedup with only a $1-4\%$ performance drop on a broad suite of tasks.

1 Introduction

Language models like ChatGPT OpenAI [2023], PaLM Chowdhery et al. [2022], T5 Raffel et al. [2020], and CM3 Aghajanyan et al. [2022], have seen unprecedented adoption in diverse sectors ranging from education and healthcare to manufacturing and marketing. The proficiency of these tools has led to unprecedented demand for these models, with users facing frequent outages and capacity limits. Additionally, ever-increasing model sizes and hardware shortages have constrained models' ability to handle a very high load of requests, thus limiting large-scale affordable access to these models. These trends bring into focus the need for high-throughput, high-performance, and environmentally responsible models that can be deployed at scale.

Multi-input Multi-output architectures (MIMO) Havasi et al. [2021], Ramé et al. [2021], Murahari et al. [2022] are a promising hardware-agnostic and architecture-agnostic paradigm that perform inference for multiple inputs *simultaneously* at the cost of a single input. This efficiency paradigm is natively geared towards yielding high-throughput models, in addition to being complementary in approach and motivation to current efficiency methods such as pruning, quantization, and distillation. Interestingly, MIMO approaches are partly inspired by the human brain's extraordinary ability to process multiple inputs and propagate information at a high bandwidth with a few neural codes Blumhagen et al. [2011], Akam and Kullmann [2014], Pirschel and Kretzberg [2016], Hong et al. [2016].

Murahari et al. [2022] introduced data multiplexing, a MIMO technique that can enable a many-fold increase in throughput. The method compresses N different instances into a single "multiplexed"

hidden representation before decompressing it into N independent predictions. While they show the plausibility of MIMO training, their method leads to a significant drop in performance ($20 - 30\%$ points) compared to state-of-the-art models.

In this work, we introduce MUX-PLMs, a class of high-throughput pre-trained language models trained in a MIMO fashion with data multiplexing to process multiple inputs (2-10) simultaneously with a forward pass over a single instance. MUX-PLMs offer up to 400% improvement in throughput over baseline pre-trained models while only being ~ 4 points and ~ 2 points worse than baseline pre-trained language models for text classification and token classification tasks, respectively. MUX-PLMs, like other pre-trained language models, provide general model initialization that can be fine-tuned for *any* downstream task. We demonstrate the effectiveness and generality of our MUX-PLMs class of pre-trained models by training MUX-BERT and MUX-ELECTRA models, which are trained with pre-trained objectives adapted from BERT Devlin et al. [2019] and ELECTRA Clark et al. [2020] respectively, although in a MIMO fashion with data multiplexing.

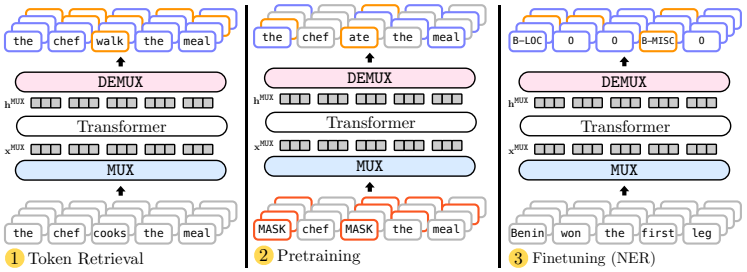

Figure 1: Illustrating the training process for MUX-PLMs. MUX-PLMs are first primed for MIMO style training with a token-retrieval auto-encoding task, where the model is trained to output the tokens in the N inputs. MUX-PLMs are then pre-trained by adapting standard pre-training objectives (BERT in this example), to MIMO style training with data multiplexing. The resulting MUX-BERT model, similar to standard PLMs, provides a general model initialization that can be fine-tuned on any downstream task (NER in this example).

2 Related Work

Efficient Inference with Transformers The ubiquity of pre-trained language models, their growing size, and over-parameterization has inspired extensive research on improving inference efficiency. This includes methods such as structured pruning Liu et al. [2019], Wang et al. [2020], Lagunas et al. [2021], Xia et al. [2022], Yang et al. [2022], knowledge distillation Hinton et al. [2015], Sanh et al. [2019], Sun et al. [2020], Jiao et al. [2020], Yin et al. [2021], quantization Zafrir et al. [2019], Shen et al. [2020], and data multiplexing Murahari et al. [2022]. These approaches assume that PLMs are highly over-parametrized and attempt to approximate a large function by learning a smaller, compressed, version of the original model.

Multi-input Multi-output Models Multi-input Multi-output (MIMO) architectures Havasi et al. [2021], Ramé et al. [2021], Murahari et al. [2022] train models using mixed-instance representations, i.e. Zhang et al. [2018], in order to obtain predictions for multiple instances simultaneously. Unlike efficiency methods, Havasi et al. [2021] and Ramé et al. [2021] try to obtain better performance by inducing multiple subnetworks in a single convolutional model to perform "ensembling for free" during inference. Data multiplexing, introduced in DataMUX Murahari et al. [2022], aims to improve model efficiency by training Transformer models with mixed-instance representations to perform simultaneous inference for language tasks, thereby improving inference throughput many-fold. Our work training PLMs with a potent MIMO architecture, data multiplexing, dramatically improves inference throughput while preserving high accuracy for downstream tasks.

3 MUX-PLMs: Data multiplexing for high-throughput language models

We propose MUX-PLMs, a class of high-throughput pre-trained Transformer-based language models trained in a MIMO fashion with data multiplexing. To demonstrate the viability and the generality of

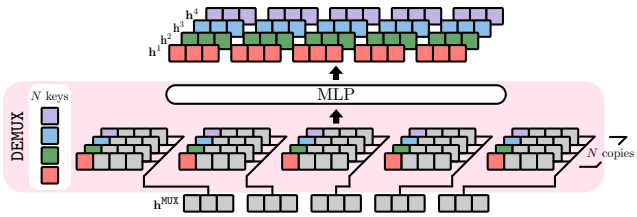

Figure 2: Illustrating our novel RSA-inspired demultiplexing module. The module is initialized with N key vectors which are used to demultiplex the transformed multiplexed representations (h^{MUX}). The keys are concatenated with h^{MUX} and are processed with an MLP to generate the demultiplexed output representations ($h_1 \cdots h_4$).

this class of models, we pre-train Transformer models with objectives based on BERT and ELECTRA, to get MUX-BERT and MUX-ELECTRA respectively. MUX-PLMs are trained with our three stage training algorithm (Figure 1). Firstly, MUX-PLMs are trained with the token retrieval task in T-MUX, which is an auto-encoding setup to decode all the tokens in the multiplexed input. This simple auto-encoding task is critical to prime the model for MIMO-style data multiplexing. The MUX-PLMs are then pre-trained with standard pre-training objectives but adapted to MIMO-fashioned training with data multiplexing. MUX-PLMs show significant throughput improvement over standard pre-trained LMs while matching their downstream task accuracies. Finally, MUX-PLMs, like other pre-trained language models, provide general model initialization that can be fine-tuned for *any* downstream task.

Contextual multiplexer T-MUX's multiplexer multiplexes tokens independent of 1) tokens in the same position in other instances and 2) other tokens in the instance, which could lead to suboptimal representations. We, therefore, explore a contextual multiplexing scheme that aggregates context both from tokens in the same instance and tokens in the same position of other instances. We first use a single transformer layer $\text{TRANS}_{\text{ctx}}$ to get contextual representations $\mathbf{h}^i_{\text{ctx}} = \text{TRANS}_{\text{ctx}}\left(\mathbf{x}^i_1, \ldots, \mathbf{x}^i_L\right))$ of length L. We apply a hadamard product with a multivariate gaussian $\mathbf{v}^i$ to all L positions.

$$\mathbf{g}^i_{\text{ctx}} = \mathbf{h}^i_{\text{ctx}} \odot \mathbf{v}^i \tag{1}$$

We generate multiplexed representations, $\mathbf{x}^{\text{MUX}}$, by applying another transformer layer $\text{TRANS}_{\text{inst}}$ across encoded representations from N instances at each position from 1 to L. This is done by transposing $\mathbf{g}_{\text{ctx}}$ and applying $\text{TRANS}_{\text{inst}}$.

$$\mathbf{x}^{\text{MUX}} = \text{TRANS}_{\text{inst}}\left(\mathbf{g}^{\top}_{\text{ctx}}\right) \tag{2}$$

RSA demultiplexer The *demultiplexer* in T-MUX requires a prefix whose length scales linearly with the number of instances (N), thus reducing the effective context length during pre-training, which degrades performance Ainslie et al. [2020], Zaheer et al. [2020], Beltagy et al. [2020]. Furthermore, it decreases throughput during inference for large N because the model must process an extra prefix of length N for each of the N instances. To address these issues, we draw inspiration from the RSA cryptosystem Rivest et al. [1978] to randomly initialize and learn N (private) key vectors $(\mathbf{k}_1, \ldots, \mathbf{k}_N, \mathbf{k}_i \in \mathbb{R}^d)$ which are keys that can be used to demultiplex the output representation (Figure 2).

$$\begin{aligned} \mathbf{h}^i &= \text{DeMUX}\left(\mathbf{h}^{\text{MUX}}, \mathbf{k}^i\right) \\ \mathbf{h}^i_j &= \text{DeMUX}\left(\mathbf{h}^{\text{MUX}}{}_j, \mathbf{k}^i\right) \end{aligned} \tag{3}$$

Akin to RSA, $\mathbf{v_i}$ and $\mathbf{k_i}$ can be treated as the keys for multiplexing (encryption) and demultiplexing (decryption) while ensuring that, unlike T-MUX, the input sequence length does not change and thereby leading to an improvement in throughput. Importantly, this architecture ensures that the keys better transfer across the different stages of training as they are no longer conditioned on the input instances.

Table 1: Average GLUE and token-level classification scores for the BASE (L=12, H=768) configuration, across ELECTRA, BERT, and MUX-PLMs for $N \in \{1, 2, 5, 10\}$. ‡ indicates our models and ↗ indicates throughput increase w.r.t. to a vanilla $\text{BERT}_{\text{BASE}}$ model. All models are evaluated on 5 seeds with mean and max scores reported.

Model	N	GLUE		Token		↗
		Mean (std)	Max	Mean (std)	Max	
BERT	**1**	**85.4 (0.0)**	**85.4**	**95.8 (0.0)**	**95.8**	1.0×
ELECTRA		82.1 (0.0)	82.1	95.3 (0.0)	95.3	1.0×
T-MUX	**2**	60.4 (0.6)	61.8	81.4 (0.1)	81.5	1.9×
MUX-BERT‡		**82.5 (0.6)**	**83.4**	**95.2 (0.1)**	**95.4**	**2.0**×
MUX-ELEC‡		**82.5 (0.4)**	83.1	95.0 (0.0)	95.1	**2.0**×
T-MUX	**5**	59.7 (0.6)	60.6	81.3 (0.2)	81.5	4.4×
MUX-BERT‡		**80.3 (0.4)**	**80.9**	**93.6 (0.1)**	**93.6**	**4.9**×
MUX-ELEC‡		79.8 (0.6)	80.5	93.4 (0.0)	93.5	**4.9**×
T-MUX	**10**	58.1 (0.5)	59.1	79.7 (0.2)	80.0	8.4×
MUX-BERT‡		77.8 (0.6)	78.8	91.6 (0.1)	**91.8**	**9.8**×
MUX-ELEC‡		**78.2 (0.6)**	**79.0**	**91.7 (0.1)**	**91.8**	9.7×

Table 2: MUX-PLMs (variants are underlined) are complementary to existing efficiency methods, while being competitive standalone. Contrary to existing methods, MUX-PLMs *do not use additional unlabelled and task-specific data* and can be easily fine-tuned for *any* downstream task without architectural modifications.

Model	↗	QNLI	QQP	SST2
BERT	1.0×	90.5	91.2	91.7
MUX-BERT (N=2)	2.0×	88.2	90.4	90.6
MUX-BERT (N=5)	4.9×	85.6	88.8	86.9
Use additional unlabelled or task-specific data				
DistilBERT_6	2.0×	89.2	88.5	91.3
Block Pruning	2.7×	89.7	-	91.2
Prune OFA	1.0×	90.3	91.2	91.5
Hybrid Approaches				
MUX-BERT (N=2) + CoFi	4.0×	88.4	90.4	90.0
TinyBERT_6	2.0×	91.1	91.1	93.0
CoFi	2.7×	91.3	-	93.0
AutoTinyBERT	4.3×	89.7	89.9	91.4
MobileBERT	2.3×	91.0	-	92.1

4 Results

4.1 MUX-PLMs outperform PLMs and T-MUX

Table 1 shows that both **MUX-BERT and MUX-ELECTRA outperform T-MUX at all levels of multiplexing (N)**, with improvements between 12 and 20 points on GLUE and token-classification tasks respectively. Furthermore, MUX-PLMs' efficient RSA-inspired demultiplexing method allows it to achieve faster throughput than T-MUX, increasing it by over 16% for $N = 10$.

Moreover, **MUX-PLMs provide a significant boost in throughput (N times faster) when compared to PLMs, without a significant loss in performance.** For example, MUX-ELECTRA ($N = 2$) is 0.4 points better and only 0.3 points worse than ELECTRA for GLUE and TOKEN tasks respectively, while being $2\times$ faster. Similarly, MUX-BERT ($N = 2$) is within 3 and 0.6 points of BERT for GLUE and TOKEN tasks respectively, while being significantly faster. We also observe that as N increases, MUX-PLMs' throughput is significantly better, though performance compared to PLMs can degrade. This is because a large N implies that MUX-PLMs must parallelly process more instances, thus having to share network parameters and activations with a larger number of instances, thus improving throughput and degrading performance. For example, the gap between ELECTRA and MUX-ELECTRA on TOKEN for $N = 2$ is 0.2 points and increases to 3.5 points for $N = 10$, which shows that N serves as a parameter to control the performance-throughput trade-off.

4.2 Comparing MUX-PLMs with other model compression methods

We compare our MUX-PLM models with other efficient learning methods, such as pruning and distillation, in Table 2. Contrary to other methods, our *vanilla* MUX-PLMs achieve competitive performance and significant throughput improvement *without* additional unlabeled and task-specific data, and can be easily fine-tuned on *any* downstream task without any architectural modifications (unlike pruning). For instance, when compared to DistilBERT, MUX-BERT ($N = 2$) does 1 point worse on QNLI and 2 points better on QQP while being equally fast.

More broadly, methods like CoFi, AutoTinyBERT, and MobileBERT show that combining qualitatively different efficiency paradigms is a promising approach towards efficient models. For example, CoFi combines structured pruning and knowledge distillation, and AutoTinyBERT combines knowledge distillation and neural architecture search. This places importance on discovering new efficiency paradigms that dovetail with existing ones We demonstrate its complementary nature by combining it with pruning. We perform structured pruning using CoFi (sparsity of 0.6) on our MUX-PLMs. The resulting pruned model is $2\times$ faster than the original MUX-PLM while being as performant. We believe that the best combination of efficiency techniques is largely hardware and use-case dependent and we hope that MIMO architectures evolve in tandem with other approaches.

References

Quora. data.quora.com/First-Quora-Dataset-Release-Question-Pairs. Accessed: 2022-10-15.

A. Aghajanyan, B. Huang, C. Ross, V. Karpukhin, H. Xu, N. Goyal, D. Okhonko, M. Joshi, G. Ghosh, M. Lewis, et al. Cm3: A causal masked multimodal model of the internet. arXiv preprint arXiv:2201.07520, 2022.

J. Ainslie, S. Ontañón, C. Alberti, P. Pham, A. Ravula, and S. Sanghai. ETC: encoding long and structured data in transformers. CoRR, abs/2004.08483, 2020. URL https://arxiv.org/abs/2004.08483.

T. Akam and D. M. Kullmann. Oscillatory multiplexing of population codes for selective communication in the mammalian brain. Nature Reviews Neuroscience, 15(2):111–122, 2014.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150, 2020.

F. Blumhagen, P. Zhu, J. Shum, Y.-P. Z. Schärer, E. Yaksi, K. Deisseroth, and R. W. Friedrich. Neuronal filtering of multiplexed odour representations. Nature, 479(7374):493–498, 2011.

D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), pages 1–14, 2017.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311, 2022.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In International Conference on Learning Representations, 2020.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. 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, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

W. B. Dolan and C. Brockett. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005.

W. Foundation. Wikipedia. URL https://dumps.wikimedia.org.

S. Grünewald, P. Piccirilli, and A. Friedrich. Coordinate constructions in english enhanced universal dependencies: Analysis and computational modeling. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 795–809, 2021.

M. Havasi, R. Jenatton, S. Fort, J. Z. Liu, J. Snoek, B. Lakshminarayanan, A. M. Dai, and D. Tran. Training independent subnetworks for robust prediction. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=OGg9XnKxFAH.

G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. ArXiv, abs/1503.02531, 2015.

S. Hong, M. Negrello, M. Junker, A. Smilgin, P. Thier, and E. De Schutter. Multiplexed coding by cerebellar purkinje neurons. Elife, 5:e13810, 2016.

X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. TinyBERT: Distilling BERT for natural language understanding. pages 4163–4174, 2020.

F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush. Block pruning for faster transformers. arXiv preprint arXiv:2109.04838, 2021.

H. Levesque, E. Davis, and L. Morgenstern. The winograd schema challenge. In Thirteenth international conference on the principles of knowledge representation and reasoning, 2012.

Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=rJlnB3C5Ym.

V. Murahari, C. E. Jimenez, R. Yang, and K. R. Narasimhan. DataMUX: Data multiplexing for neural networks. In Thirty-Sixth Conference on Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=UdgtTVTdswg.

OpenAI. Introducing chatgpt, 2023. URL https://openai.com/blog/chatgpt.

F. Pirschel and J. Kretzberg. Multiplexed population coding of stimulus properties by leech mechanosensory cells. Journal of Neuroscience, 36(13):3636–3647, 2016.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21:1–67, 2020.

A. Ramé, R. Sun, and M. Cord. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 823–833, October 2021.

R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120–126, 1978.

E. T. K. Sang and F. D. Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In CoNLL, 2003.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108, 2019.

S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In AAAI, 2020.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 conference on empirical methods in natural language processing, pages 1631–1642, 2013.

Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. MobileBERT: a compact task-agnostic bert for resource-limited devices. pages 2158–2170, 2020.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In International Conference on Learning Representations, 2018.

Z. Wang, J. Wohlwend, and T. Lei. Structured pruning of large language models. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.496. URL https://doi.org/10.18653%2Fv1%2F2020.emnlp-main.496.

A. Warstadt, A. Singh, and S. Bowman. Neural network acceptability judgments. Transactions of the Association for Computational Linguistics, 7:625–641, 2019.

A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In NAACL-HLT, 2018.

M. Xia, Z. Zhong, and D. Chen. Structured pruning learns compact and accurate models. In Association for Computational Linguistics (ACL), 2022.

Z. Yang, Y. Cui, and Z. Chen. TextPruner: A model pruning toolkit for pre-trained language models. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 35–43, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-demo.4. URL https://aclanthology.org/2022.acl-demo.4.

Y. Yin, C. Chen, L. Shang, X. Jiang, X. Chen, and Q. Liu. AutoTinyBERT: Automatic hyper-parameter optimization for efficient pre-trained language models. pages 5146–5157, 2021.

O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat. Q8bert: Quantized 8bit bert. 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS), pages 36–39, 2019.

M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. Big bird: Transformers for longer sequences. Advances in Neural Information Processing Systems, 33:17283–17297, 2020.

H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In International Conference on Learning Representations, 2018.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In The IEEE International Conference on Computer Vision (ICCV), December 2015.

A Appendices

B Experimental Setup

Datasets We pre-train all models on Wikipedia Foundation and Bookscorpus Zhu et al. [2015]. We evaluate on all datasets from the GLUE benchmark Wang et al. [2018], which are CoLA Warstadt et al. [2019], SST-2 Socher et al. [2013], MRPC Dolan and Brockett [2005], QQP qqp, STS-B Cer et al. [2017], MNLI Williams et al. [2018], QNLI Wang et al. [2018], RTE Wang et al. [2018], and WNLI Levesque et al. [2012]. We also evaluate on token classification tasks such as named entity recognition Sang and Meulder [2003] and POS tagging Grünewald et al. [2021]. We don't report average over WNLI and CoLA as these are the two smallest tasks in GLUE and we observe high variance across different seeds. All numbers are reported on the dev split. We report scores for all tasks in Appendix F.

Models We experiment with ELECTRA and BERT pre-training objectives and present the pre-trained multiplexed models **MUX-BERT** and **MUX-ELECTRA** for $N = 2, 5$ and 10. To simplify training, we use a random generator to train MUX-ELECTRA models, presented as an ablation in Clark et al. [2020], instead of using a smaller masked LM. Except where otherwise noted, we do not use the contextual MUX module, but instead, use the RSA demultiplexing module. Refer to Appendix C and D for implementation details.

Baselines We run experiments to compare our models against T-MUX, from Murahari et al. [2022] and baseline PLMs - ELECTRA and BERT, across three different model configurations (SMALL, BASE, and LARGE). We also provide a comparison to results reported in recent PLM pruning and distillation papers in Table 2.

Multi-run evaluation We evaluate all models across 5 random seeds to reduce variance for smaller datasets which is caused by the randomized order in which we multiplex instances in the batch. We report both the average and maximum scores across seeds in Table 1 to understand the importance of ordering the multiplexed instances and report average across seeds for all other results.

C Pre-training Details

We report all pre-training related hyper-parameters in Table 3. We primarily use the HuggingFace Transformers implementations for BERT and ELECTRA based models. All pre-training experiments were run on 8 A100 GPUs with distributed training. We run a small hyper-parameter search over over two learning rates. All pre-trained models are primed with the token retrieval task introduced in Murahari et al. [2022]. We train on the Wikipedia and Bookscorpus datasets for up to 10000 training steps with a learning rate of $1e-4$, and with a sequence length of 512.

For MUX-ELECTRA models, we don't train a generator as in the original ELECTRA work, but only use uniform-random token replacement. This is similar to what was used in ablations in ELECTRA Clark et al. [2020]. The generator randomly replaces 15% of tokens in the input with other tokens in the vocabulary.

D Fine-tuning Details

We report all the fine-tuning related hyper-parameters in Table 4. We run a small hyper-parameter search on the learning rate, batch size and number of training steps for different tasks. All models were trained with half-precision. We report numbers on the validation split. For GLUE tasks, we use the default metrics in Wang et al. [2018] and use F1 for the token-level tasks. All fine-tuning experiments were trained on 1 V100 GPU.

Speedup calculation For all models, we calculate throughput (samples/second) on a single V100 GPU and report throughput gains with respect to the $\text{BERT}_{\text{BASE}}$ model. We calculate throughput by averaging across 3 different trials (1 trial = 200 mini-batches) and use a batch size of 128 and a

Hyperparameter	MUX-BERT			MUX-ELECTRA
	SMALL	BASE	LARGE	BASE
Number of layers	4	12	24	12
Hidden Size	512	768	1024	768
FFN intermediate hidden size	2048	3072	4096	3072
Attention heads	8	12	16	12
Attention head size	64	64	64	64
Mask percent	15	15	15	N/A
Learning Rate Decay	Linear	Linear	Linear	Linear
Warmup steps	10000	10000	10000	10000
Learning Rate	[1e-4, 5e-5]	[1e-4, 5e-5]	[1e-4, 5e-5]	[1e-4, 5e-5]
Adam ϵ	1e-6	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999	0.999
Attention Dropout	0.1	0.1	0.1	0.1
Dropout	0.1	0.1	0.1	0.1
Batch Size	256	256	256	256
Sequence Length	512	512	512	512
Train Steps	1M	1M	1M	1M

Table 3: Pre-train hyper-parameters for MUX-BERT and MUX-ELECTRA models. We only report results for the Base configuration for MUX-ELECTRA models.

Hyperparameter	Value
Learning Rate	[2e-5, 5e-5]
Adam ϵ	1e-8
Adam β_1	0.9
Adam β_2	0.999
Learning rate decay	Linear
Warmup fraction	0.1
Attention Dropout	0.1
Dropout	0.1
Weight Decay	0
Batch Size	[32, 128] for SMALL/ BASE, [16, 64] for LARGE
Train Steps	2000 for RTE and WNLI 10000 for MRPC, COLA and STSB 20000 for NER, SST2, QNLI and POS [20000, 100000] for MNLI and QQP
Sequence Length	128

Table 4: Fine-tune hyperparameters

sequence length of 128 following prior work Xia et al. [2022]. We measure throughput for sequence-classification tasks on QQP and measure throughput for token-level classification tasks on named entity recognition.

E Analysis details

E.1 Ensembling results setup

We find that multiplexing the same instance by duplicating the instance N times leads to worse performance. This is likely because this input configuration is very out of distribution from what the multiplexed models are trained on. To address this, we randomly permute the instances in the batch after duplicating the instances N times. This ensures that the input to the multiplexer lies in a similar distribution to what the model was trained on.

Model Size	N	MNLI	QQP	QNLI	MRPC	WNLI	STSB	RTE	SST2	COLA	GLUE	GLUE$_{-\text{WNLI, COLA}}$
SMALL	1	**77.86**$_{\pm 0.0}$	**88.99**$_{\pm 0.0}$	84.00$_{\pm 0.0}$	77.70$_{\pm 0.0}$	**56.34**$_{\pm 0.0}$	**84.25**$_{\pm 0.0}$	**62.45**$_{\pm 0.0}$	**88.88**$_{\pm 0.0}$	**43.48**$_{\pm 0.0}$	**73.77**	**80.59**
	2	75.09$_{\pm 0.1}$	88.88$_{\pm 0.1}$	**84.31**$_{\pm 0.2}$	**79.75**$_{\pm 0.7}$	50.99$_{\pm 8.1}$	82.65$_{\pm 0.3}$	55.52$_{\pm 1.5}$	87.04$_{\pm 0.7}$	30.64$_{\pm 1.7}$	70.54	79.03
	5	70.50$_{\pm 0.1}$	86.39$_{\pm 0.1}$	81.23$_{\pm 0.2}$	74.26$_{\pm 1.0}$	**54.65**$_{\pm 3.3}$	79.90$_{\pm 0.2}$	58.56$_{\pm 1.9}$	82.57$_{\pm 0.3}$	12.78$_{\pm 1.6}$	66.76	76.20
	10	61.98$_{\pm 0.1}$	80.85$_{\pm 0.1}$	63.47$_{\pm 0.3}$	70.69$_{\pm 0.9}$	**56.62**$_{\pm 4.3}$	36.93$_{\pm 1.0}$	53.57$_{\pm 1.8}$	80.39$_{\pm 0.4}$	1.10$_{\pm 2.2}$	56.18	63.98
BASE	1	**84.24**$_{\pm 0.0}$	**91.19**$_{\pm 0.0}$	**90.54**$_{\pm 0.0}$	**87.75**$_{\pm 0.0}$	**56.34**$_{\pm 0.0}$	**89.18**$_{\pm 0.0}$	**63.18**$_{\pm 0.0}$	**91.74**$_{\pm 0.0}$	**58.79**$_{\pm 0.0}$	**79.22**	**85.40**
	2	80.59$_{\pm 0.1}$	90.36$_{\pm 0.1}$	88.17$_{\pm 0.1}$	83.77$_{\pm 1.4}$	50.70$_{\pm 7.0}$	85.84$_{\pm 0.1}$	58.19$_{\pm 1.6}$	90.62$_{\pm 0.6}$	55.61$_{\pm 1.6}$	75.98	82.51
	5	77.18$_{\pm 0.2}$	88.79$_{\pm 0.1}$	85.58$_{\pm 0.1}$	80.10$_{\pm 0.6}$	53.52$_{\pm 2.5}$	84.28$_{\pm 0.2}$	59.13$_{\pm 1.2}$	86.88$_{\pm 0.4}$	12.33$_{\pm 2.4}$	69.75	80.28
	10	73.62$_{\pm 0.3}$	86.94$_{\pm 0.1}$	82.08$_{\pm 0.3}$	78.63$_{\pm 0.6}$	52.68$_{\pm 6.0}$	81.62$_{\pm 0.2}$	58.27$_{\pm 2.4}$	83.44$_{\pm 0.6}$	0.00$_{\pm 0.0}$	66.36	77.80
LARGE	1	**85.79**$_{\pm 0.0}$	**91.46**$_{\pm 0.0}$	**92.29**$_{\pm 0.0}$	83.82$_{\pm 0.0}$	**56.34**$_{\pm 0.0}$	**89.53**$_{\pm 0.0}$	**66.06**$_{\pm 0.0}$	**91.40**$_{\pm 0.0}$	**57.79**$_{\pm 0.0}$	**79.39**	**85.76**
	2	83.23$_{\pm 0.2}$	90.85$_{\pm 0.1}$	90.66$_{\pm 0.2}$	**84.90**$_{\pm 0.8}$	**56.34**$_{\pm 0.0}$	88.22$_{\pm 0.2}$	59.21$_{\pm 0.9}$	91.38$_{\pm 0.4}$	**57.89**$_{\pm 1.5}$	78.08	84.06
	5	79.55$_{\pm 0.2}$	89.37$_{\pm 0.1}$	87.41$_{\pm 0.2}$	83.77$_{\pm 1.1}$	**54.93**$_{\pm 0.0}$	85.86$_{\pm 0.3}$	57.26$_{\pm 2.0}$	88.65$_{\pm 0.7}$	46.66$_{\pm 0.9}$	74.83	81.70
	10	35.45$_{\pm 0.0}$	63.18$_{\pm 0.0}$	50.54$_{\pm 0.0}$	68.38$_{\pm 0.0}$	**56.90**$_{\pm 5.2}$	82.81$_{\pm 0.2}$	52.13$_{\pm 1.9}$	50.92$_{\pm 0.0}$	1.87$_{\pm 4.6}$	51.35	57.63

Table 5: We show the full GLUE results for MUX-BERT. We report the mean accuracy and standard deviation over 5 seeds. Extrema and values within their standard deviation are emphasized for each model size.

Model Size	N	MNLI	QQP	QNLI	MRPC	WNLI	STSB	RTE	SST2	COLA	GLUE	GLUE$_{-\text{WNLI, COLA}}$
SMALL	1	**77.86**	**88.99**	84.00	77.70	56.34	**84.25**	**62.45**	**88.88**	**43.48**	**73.77**	**80.59**
	2	75.21	**89.01**	**84.61**	**80.64**	61.97	82.97	58.12	87.84	33.08	72.61	79.77
	5	70.66	86.46	81.60	75.74	61.97	80.24	60.65	83.49	15.57	68.49	76.98
	10	62.17	80.93	63.85	71.81	**63.38**	38.20	55.96	80.96	2.63	57.77	64.84
BASE	1	**84.24**	**91.19**	**90.54**	**87.75**	56.34	**89.18**	**63.18**	**91.74**	**58.79**	**79.22**	**85.40**
	2	80.82	90.47	88.28	86.03	**66.20**	86.06	60.65	91.51	56.93	78.55	83.40
	5	77.66	88.89	85.70	81.13	59.15	84.47	60.65	87.50	15.79	71.22	80.86
	10	74.04	87.03	82.45	79.41	63.38	81.89	62.45	84.29	0.00	68.33	78.79
LARGE	1	**85.79**	**91.46**	**92.29**	83.82	56.34	**89.53**	**66.06**	91.40	57.79	**79.39**	**85.76**
	2	83.40	90.94	90.96	**86.27**	56.34	88.50	60.29	**91.86**	**60.50**	78.78	84.60
	5	79.69	89.43	87.81	84.80	57.75	86.49	60.65	89.45	47.56	75.96	82.62
	10	35.46	63.18	50.89	68.38	**61.97**	83.04	55.60	50.92	7.55	53.00	58.21

Table 6: We show the full GLUE results for MUX-BERT. We report the *maximum* accuracy over 5 seeds. Extrema are emphasized.

F Task performance breakdown for all variants

N	MNLI	QQP	QNLI	MRPC	WNLI	STSB	RTE	SST2	COLA	GLUE	GLUE$_{-\text{WNLI, COLA}}$
1	**81.49**$_{\pm 0.0}$	**90.73**$_{\pm 0.0}$	**89.73**$_{\pm 0.0}$	75.98$_{\pm 0.0}$	**56.34**$_{\pm 0.0}$	**87.73**$_{\pm 0.0}$	57.76$_{\pm 0.0}$	**91.51**$_{\pm 0.0}$	**56.79**$_{\pm 0.0}$	**76.45**	82.13
2	80.29$_{\pm 0.2}$	90.58$_{\pm 0.1}$	88.39$_{\pm 0.2}$	**83.73**$_{\pm 0.7}$	**57.18**$_{\pm 2.1}$	86.80$_{\pm 0.1}$	**58.77**$_{\pm 1.1}$	88.65$_{\pm 0.4}$	51.92$_{\pm 1.7}$	76.26	**82.46**
5	76.99$_{\pm 0.2}$	89.08$_{\pm 0.0}$	85.40$_{\pm 0.3}$	80.25$_{\pm 1.6}$	**56.90**$_{\pm 4.5}$	84.27$_{\pm 0.2}$	57.26$_{\pm 1.0}$	85.09$_{\pm 1.0}$	26.89$_{\pm 1.2}$	71.35	79.76
10	74.62$_{\pm 0.2}$	87.63$_{\pm 0.1}$	82.70$_{\pm 0.2}$	77.89$_{\pm 0.7}$	50.99$_{\pm 4.9}$	81.96$_{\pm 0.5}$	**59.86**$_{\pm 2.1}$	82.71$_{\pm 0.5}$	27.76$_{\pm 2.3}$	69.57	78.20

Table 7: We show the full GLUE results for MUX-ELECTRA$_{\text{BASE}}$. We report the mean accuracy and standard deviation over 5 seeds. Extrema and values within their standard deviation are emphasized for each model size.

N	Retreival Rate	MNLI	QQP	QNLI	MRPC	WNLI	STSB	RTE	SST2	COLA	GLUE	GLUE$_{-\text{WNLI, COLA}}$
2	0.0	83.23$_{\pm 0.2}$	90.85$_{\pm 0.1}$	**90.66**$_{\pm 0.2}$	**84.90**$_{\pm 0.8}$	**56.34**$_{\pm 0.0}$	**88.22**$_{\pm 0.2}$	**59.21**$_{\pm 0.9}$	**91.38**$_{\pm 0.4}$	**57.89**$_{\pm 1.5}$	**78.08**	**84.06**
	0.1	**83.55**$_{\pm 0.3}$	**90.90**$_{\pm 0.1}$	**90.58**$_{\pm 0.2}$	**85.49**$_{\pm 1.1}$	**56.34**$_{\pm 0.0}$	**88.28**$_{\pm 0.2}$	**57.76**$_{\pm 1.4}$	90.69$_{\pm 0.8}$	**59.36**$_{\pm 1.4}$	**78.11**	83.89
	0.2	**83.50**$_{\pm 0.1}$	**90.96**$_{\pm 0.1}$	**90.69**$_{\pm 0.2}$	**84.95**$_{\pm 0.5}$	**56.34**$_{\pm 0.0}$	**88.28**$_{\pm 0.2}$	**58.34**$_{\pm 1.6}$	90.69$_{\pm 0.5}$	**59.17**$_{\pm 1.5}$	**78.10**	83.92
	0.5	**83.41**$_{\pm 0.2}$	**90.91**$_{\pm 0.0}$	90.47$_{\pm 0.1}$	**85.25**$_{\pm 0.5}$	**56.34**$_{\pm 0.0}$	88.02$_{\pm 0.1}$	**59.35**$_{\pm 1.6}$	89.52$_{\pm 0.6}$	**59.41**$_{\pm 2.0}$	**78.08**	83.85
5	0.0	**79.55**$_{\pm 0.2}$	**89.37**$_{\pm 0.1}$	**87.41**$_{\pm 0.2}$	**83.77**$_{\pm 1.1}$	54.93$_{\pm 0.0}$	**85.86**$_{\pm 0.3}$	**57.26**$_{\pm 2.0}$	**88.65**$_{\pm 0.7}$	**46.66**$_{\pm 0.9}$	**74.83**	**81.70**
	0.1	**79.49**$_{\pm 0.1}$	**89.34**$_{\pm 0.1}$	**87.25**$_{\pm 0.3}$	81.81$_{\pm 1.3}$	53.24$_{\pm 1.6}$	**85.80**$_{\pm 0.2}$	**55.60**$_{\pm 2.4}$	**88.19**$_{\pm 0.7}$	**47.60**$_{\pm 1.0}$	74.26	81.07
	0.2	**79.37**$_{\pm 0.1}$	**89.42**$_{\pm 0.1}$	**87.23**$_{\pm 0.3}$	82.40$_{\pm 1.1}$	54.93$_{\pm 0.0}$	**85.85**$_{\pm 0.2}$	**55.38**$_{\pm 2.6}$	87.84$_{\pm 0.8}$	43.58$_{\pm 1.2}$	74.00	81.07
	0.5	79.24$_{\pm 0.1}$	89.30$_{\pm 0.1}$	87.21$_{\pm 0.3}$	82.06$_{\pm 1.7}$	**56.34**$_{\pm 0.0}$	**85.97**$_{\pm 0.2}$	52.27$_{\pm 4.0}$	**88.58**$_{\pm 0.6}$	**47.01**$_{\pm 2.3}$	74.22	80.66
10	0.0	**35.45**$_{\pm 0.0}$	**63.18**$_{\pm 0.0}$	**50.54**$_{\pm 0.0}$	**68.38**$_{\pm 0.0}$	**56.90**$_{\pm 5.2}$	**82.81**$_{\pm 0.2}$	52.13$_{\pm 1.9}$	50.92$_{\pm 0.0}$	**1.87**$_{\pm 4.6}$	**51.35**	**57.63**
	0.1	**35.45**$_{\pm 0.0}$	**63.18**$_{\pm 0.0}$	**50.65**$_{\pm 0.2}$	**68.38**$_{\pm 0.0}$	**54.93**$_{\pm 5.0}$	4.45$_{\pm 1.5}$	51.48$_{\pm 2.4}$	50.92$_{\pm 0.0}$	**1.34**$_{\pm 1.8}$	42.31	46.36
	0.2	**35.45**$_{\pm 0.0}$	**63.18**$_{\pm 0.0}$	50.21$_{\pm 0.5}$	**68.43**$_{\pm 0.8}$	**54.65**$_{\pm 4.2}$	0.23$_{\pm 1.5}$	52.35$_{\pm 2.0}$	**51.72**$_{\pm 0.4}$	**0.29**$_{\pm 2.7}$	41.83	45.94
	0.5	**35.45**$_{\pm 0.0}$	**63.18**$_{\pm 0.0}$	50.43$_{\pm 0.4}$	**68.38**$_{\pm 0.0}$	**56.06**$_{\pm 0.6}$	82.01$_{\pm 0.6}$	**52.71**$_{\pm 0.0}$	50.92$_{\pm 0.0}$	**1.51**$_{\pm 1.7}$	51.18	**57.58**

Table 8: GLUE results for MUX-BERT$_{\text{LARGE}}$ when using a retrieval auxiliary objective during MLM pretraining with different trade-off rates to the MLM objective. We report the average accuracy over 5 seeds. Extrema and values within their standard deviation are emphasized for each value of N.

N	Mux Strategy	MNLI	QQP	QNLI	MRPC	WNLI	STSB	RTE	SST2	COLA	GLUE	GLUE$_{-\text{WNLI, COLA}}$
2	MUX-BERT	80.59$_{\pm 0.1}$	90.36$_{\pm 0.1}$	88.17$_{\pm 0.1}$	83.77$_{\pm 1.4}$	50.70$_{\pm 7.0}$	85.84$_{\pm 0.1}$	58.19$_{\pm 1.6}$	90.62$_{\pm 0.6}$	**55.61**$_{\pm 1.6}$	75.98	82.51
	DataMUX	**81.64**$_{\pm 0.2}$	**90.67**$_{\pm 0.1}$	88.39$_{\pm 0.2}$	**84.17**$_{\pm 0.4}$	**56.34**$_{\pm 0.0}$	**86.36**$_{\pm 0.2}$	**60.87**$_{\pm 0.7}$	90.50$_{\pm 0.4}$	53.74$_{\pm 1.0}$	**76.96**	**83.23**
	Attention	81.32$_{\pm 0.2}$	**90.65**$_{\pm 0.0}$	**88.77**$_{\pm 0.1}$	80.88$_{\pm 0.6}$	**56.34**$_{\pm 0.0}$	**86.25**$_{\pm 0.1}$	56.90$_{\pm 1.2}$	**91.06**$_{\pm 0.2}$	47.15$_{\pm 1.1}$	75.48	82.26
5	MUX-BERT	**77.18**$_{\pm 0.2}$	88.79$_{\pm 0.1}$	**85.58**$_{\pm 0.1}$	**80.10**$_{\pm 0.6}$	53.52$_{\pm 2.5}$	**84.28**$_{\pm 0.2}$	**59.13**$_{\pm 1.2}$	**86.88**$_{\pm 0.4}$	12.33$_{\pm 2.4}$	69.75	**80.28**
	DataMUX	76.32$_{\pm 0.1}$	**89.13**$_{\pm 0.1}$	84.22$_{\pm 0.3}$	78.38$_{\pm 0.9}$	**59.44**$_{\pm 3.5}$	81.78$_{\pm 0.4}$	54.15$_{\pm 1.3}$	86.17$_{\pm 0.4}$	28.32$_{\pm 0.8}$	**70.88**	78.59
	Attention	**77.16**$_{\pm 0.1}$	88.71$_{\pm 0.0}$	84.33$_{\pm 0.1}$	70.49$_{\pm 0.6}$	54.08$_{\pm 3.2}$	80.37$_{\pm 0.3}$	54.44$_{\pm 2.5}$	81.95$_{\pm 0.3}$	**34.67**$_{\pm 1.2}$	69.58	76.78
10	MUX-BERT	**73.62**$_{\pm 0.3}$	86.94$_{\pm 0.1}$	82.08$_{\pm 0.3}$	**78.63**$_{\pm 0.6}$	52.68$_{\pm 6.0}$	81.62$_{\pm 0.2}$	**58.27**$_{\pm 2.4}$	**83.44**$_{\pm 0.6}$	0.00$_{\pm 0.0}$	66.36	**77.80**
	DataMUX	72.74$_{\pm 0.1}$	87.88$_{\pm 0.1}$	**82.28**$_{\pm 0.2}$	77.30$_{\pm 0.5}$	**56.34**$_{\pm 0.0}$	78.07$_{\pm 0.4}$	55.31$_{\pm 1.2}$	82.36$_{\pm 0.3}$	13.56$_{\pm 3.0}$	67.32	76.56
	Attention	71.83$_{\pm 0.2}$	**88.00**$_{\pm 0.0}$	81.46$_{\pm 0.2}$	73.53$_{\pm 0.5}$	53.24$_{\pm 5.4}$	**82.95**$_{\pm 0.2}$	52.71$_{\pm 0.0}$	81.28$_{\pm 0.4}$	**32.84**$_{\pm 0.6}$	**68.65**	75.97

Table 9: GLUE results for MUX-BERT$_{\text{BASE}}$ using alternative multiplexing-demultiplexing strategies. We report the average accuracy over 5 seeds. Extrema and values within their standard deviation are emphasized for each value of N.

Model Size	N	MNLI	QQP	QNLI	MRPC	WNLI	STSB	RTE	SST2	COLA	GLUE	GLUE$_{-\text{WNLI, COLA}}$
SMALL	2	**61.48**$_{\pm 0.2}$	**80.33**$_{\pm 0.0}$	**60.05**$_{\pm 0.2}$	**68.43**$_{\pm 0.5}$	**56.34**$_{\pm 0.0}$	**15.02**$_{\pm 0.4}$	**51.12**$_{\pm 0.6}$	**79.75**$_{\pm 0.3}$	**8.22**$_{\pm 0.7}$	**53.42**	**59.45**
	5	58.35$_{\pm 0.2}$	77.50$_{\pm 0.1}$	57.17$_{\pm 0.3}$	**68.38**$_{\pm 0.0}$	**56.34**$_{\pm 0.0}$	11.31$_{\pm 0.3}$	**51.70**$_{\pm 1.3}$	77.78$_{\pm 0.3}$	6.02$_{\pm 0.7}$	51.62	57.46
	10	53.63$_{\pm 0.2}$	77.03$_{\pm 0.1}$	51.22$_{\pm 0.3}$	**68.38**$_{\pm 0.0}$	**57.46**$_{\pm 6.3}$	12.40$_{\pm 1.3}$	**52.35**$_{\pm 2.7}$	50.92$_{\pm 0.0}$	0.00$_{\pm 0.0}$	47.04	52.28
BASE	2	**63.29**$_{\pm 0.3}$	**81.42**$_{\pm 0.1}$	**60.35**$_{\pm 0.4}$	68.38$_{\pm 0.2}$	56.90$_{\pm 5.8}$	**17.65**$_{\pm 1.0}$	51.19$_{\pm 1.7}$	**80.78**$_{\pm 0.5}$	**9.62**$_{\pm 1.5}$	**54.40**	**60.44**
	5	60.67$_{\pm 0.2}$	79.42$_{\pm 0.1}$	59.77$_{\pm 0.2}$	**69.61**$_{\pm 0.8}$	53.80$_{\pm 7.3}$	14.92$_{\pm 1.8}$	**52.71**$_{\pm 0.8}$	**81.15**$_{\pm 0.6}$	**10.35**$_{\pm 1.7}$	53.60	59.75
	10	59.07$_{\pm 0.2}$	78.22$_{\pm 0.1}$	57.99$_{\pm 0.5}$	68.38$_{\pm 0.0}$	**60.28**$_{\pm 3.0}$	11.83$_{\pm 0.6}$	**53.07**$_{\pm 1.1}$	78.35$_{\pm 1.1}$	7.40$_{\pm 1.7}$	52.73	58.13
LARGE	2	**64.64**$_{\pm 0.2}$	**82.10**$_{\pm 0.1}$	**60.21**$_{\pm 0.2}$	**69.95**$_{\pm 0.9}$	**56.34**$_{\pm 0.0}$	**21.62**$_{\pm 0.4}$	52.71$_{\pm 0.0}$	**80.34**$_{\pm 0.9}$	**8.72**$_{\pm 2.1}$	**55.18**	**61.65**
	5	60.78$_{\pm 0.3}$	78.56$_{\pm 0.1}$	**60.19**$_{\pm 0.3}$	**69.51**$_{\pm 0.5}$	**56.34**$_{\pm 0.0}$	17.33$_{\pm 1.1}$	52.71$_{\pm 0.0}$	78.28$_{\pm 0.8}$	**10.63**$_{\pm 2.7}$	53.81	59.62
	10	48.79$_{\pm 0.6}$	68.41$_{\pm 0.1}$	55.76$_{\pm 0.8}$	68.58$_{\pm 0.6}$	**58.59**$_{\pm 3.3}$	8.38$_{\pm 1.1}$	**54.95**$_{\pm 0.9}$	64.82$_{\pm 1.0}$	3.48$_{\pm 3.9}$	47.97	52.81

Table 10: GLUE results for T-MUX with the original training recipe and implementation from Murahari et al. [2022]. We report the average accuracy and standard deviation over 5 seeds. Extrema and values within their standard deviation are emphasized for each model size.