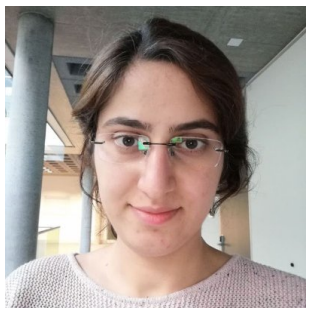
COMPACTER:

Efficient Low-Rank Hypercomplex Adapter Layers

Rabeeh Karimi Mahabadi[1, 2] James Henderson[2] Sebastian Ruder[3]

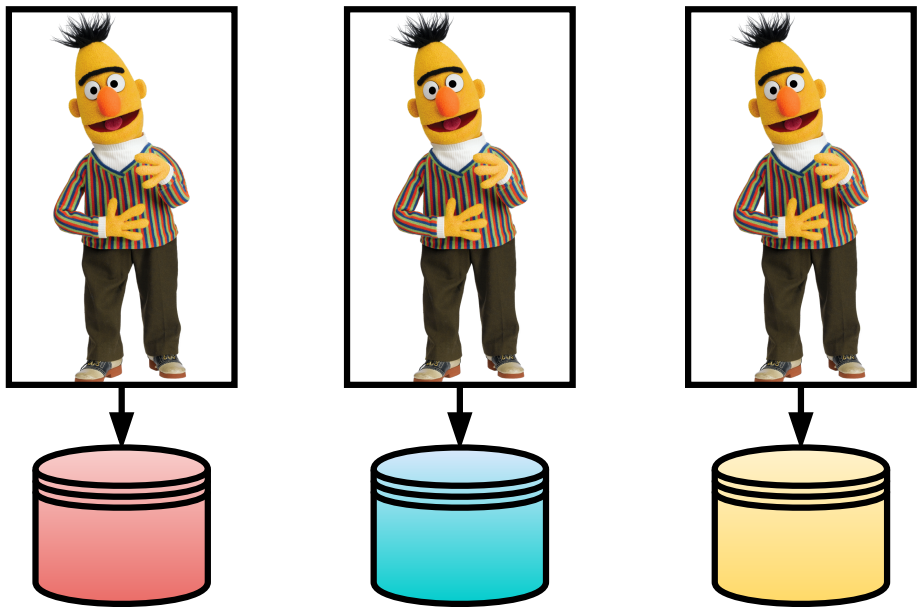
[1]EPFL University, [2]Idiap Research Institute, [3]DeepMind



Motivation

- Fine-tuning large-scale pretrained language models with millions and billions of parameters on downstream tasks is:
  - Sample-inefficient
  - Unstable in low-resource settings
  - Requires storing a separate copy of the model for each task

Fine-tuned Large-scale Pretrained Language models



Downstream Tasks

Contributions

- We propose COMPACTER
  - A parameter-efficient fine-tuning method
  - With a better trade-off between task performance, memory and training time
- Benchmark recent parameter-efficient methods
  - Provide insights on their performance and efficiency

Background: Adapters

- Freeze the model
- Train adapters and layernorms [1]

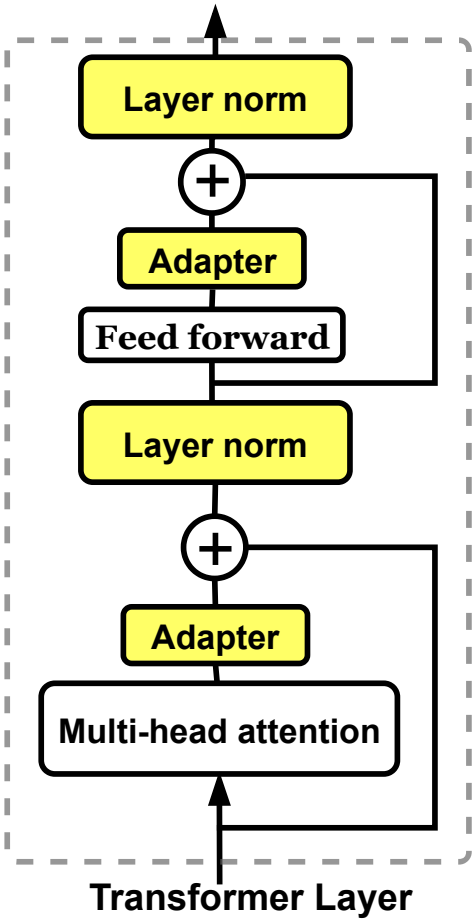


Figure: Adapter integration in a pretrained transformer model.

Background: Adapters

- A bottleneck architecture
- Consisting of a down projection, non-linearity, and up projection

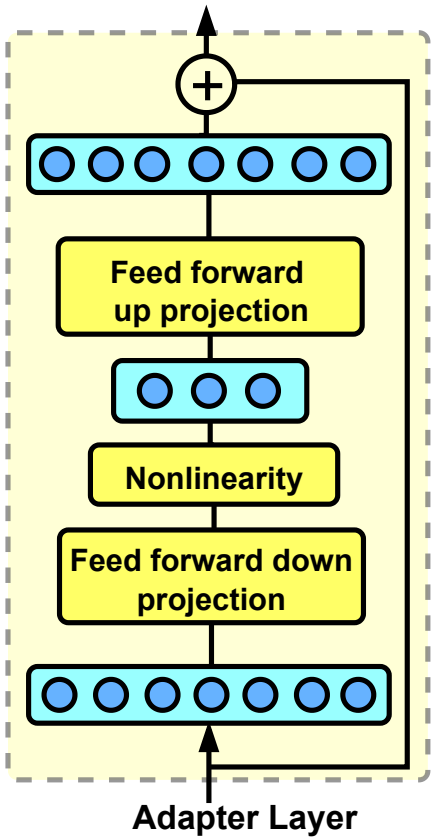


Figure: Adapter architecture.

Compact and Efficient Adapter Layers

- Down and Up projections in adapters ($\boldsymbol{W} \in \mathbb{R}^{k \times d}$) are fully connected layers:

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

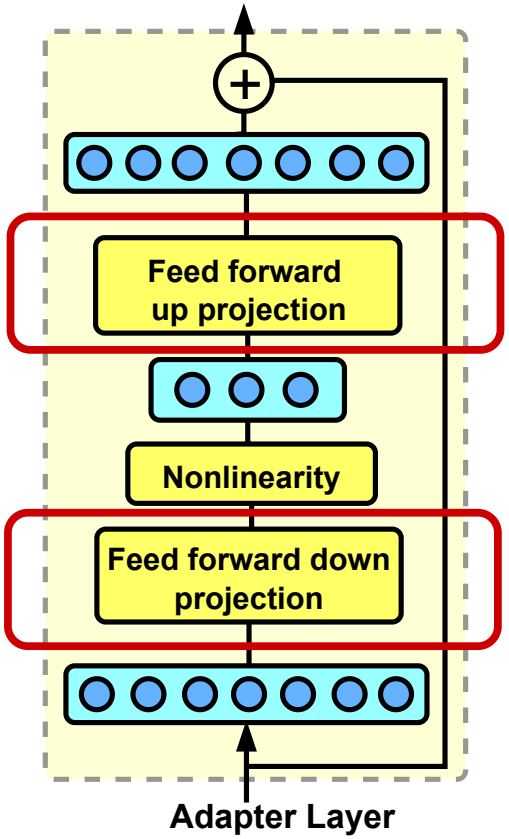


Figure: Adapters' weights.

Compact and Efficient Adapter Layers

- $\boldsymbol{W}$ can be learned via parameterized hypercomplex multiplication (PHM) layers [2].
  - Let $\boldsymbol{W} \in \mathbb{R}^{k \times d}$
  - Assume k and d are divisible by a user-defined hyper-parameter $n \in \mathbb{Z}_{>0}$
  - $\boldsymbol{W}$ is generated by a summation of Kronecker products between $\boldsymbol{A_i} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{B_i} \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}$

$$\boldsymbol{W} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \boldsymbol{B_i},$$

- Reduces trainable parameters by $\frac{1}{n}$

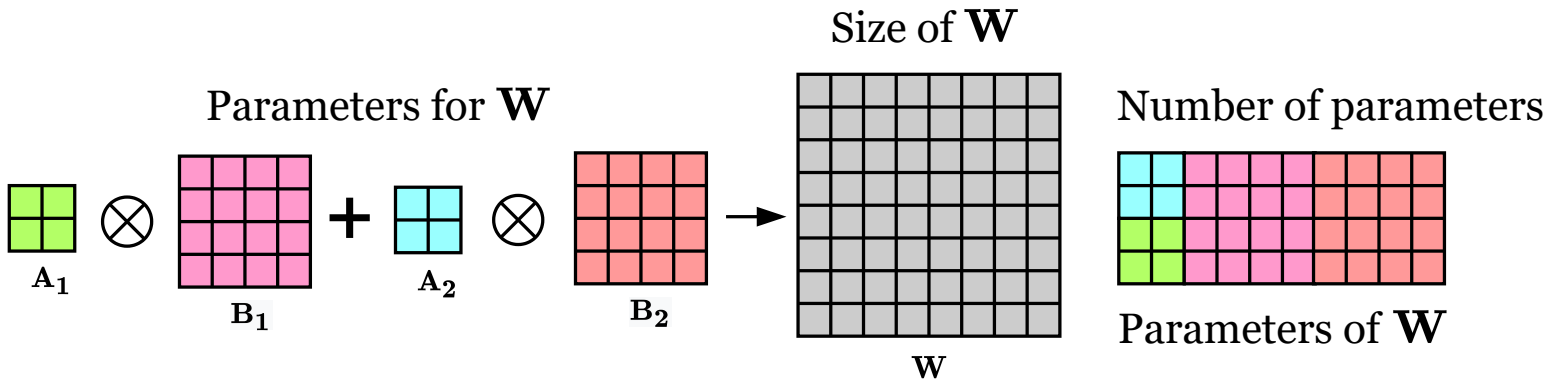


Figure: Parameterized Hypercomplex Multiplication Layers.

Compacter  : Beyond Hypercomplex Adapters

- COMPACTER is motivated by the followings:
  - There are redundancies in information captured by adapters [1].
  - Sharing adapters across layers can cause a small drop in performance [3].

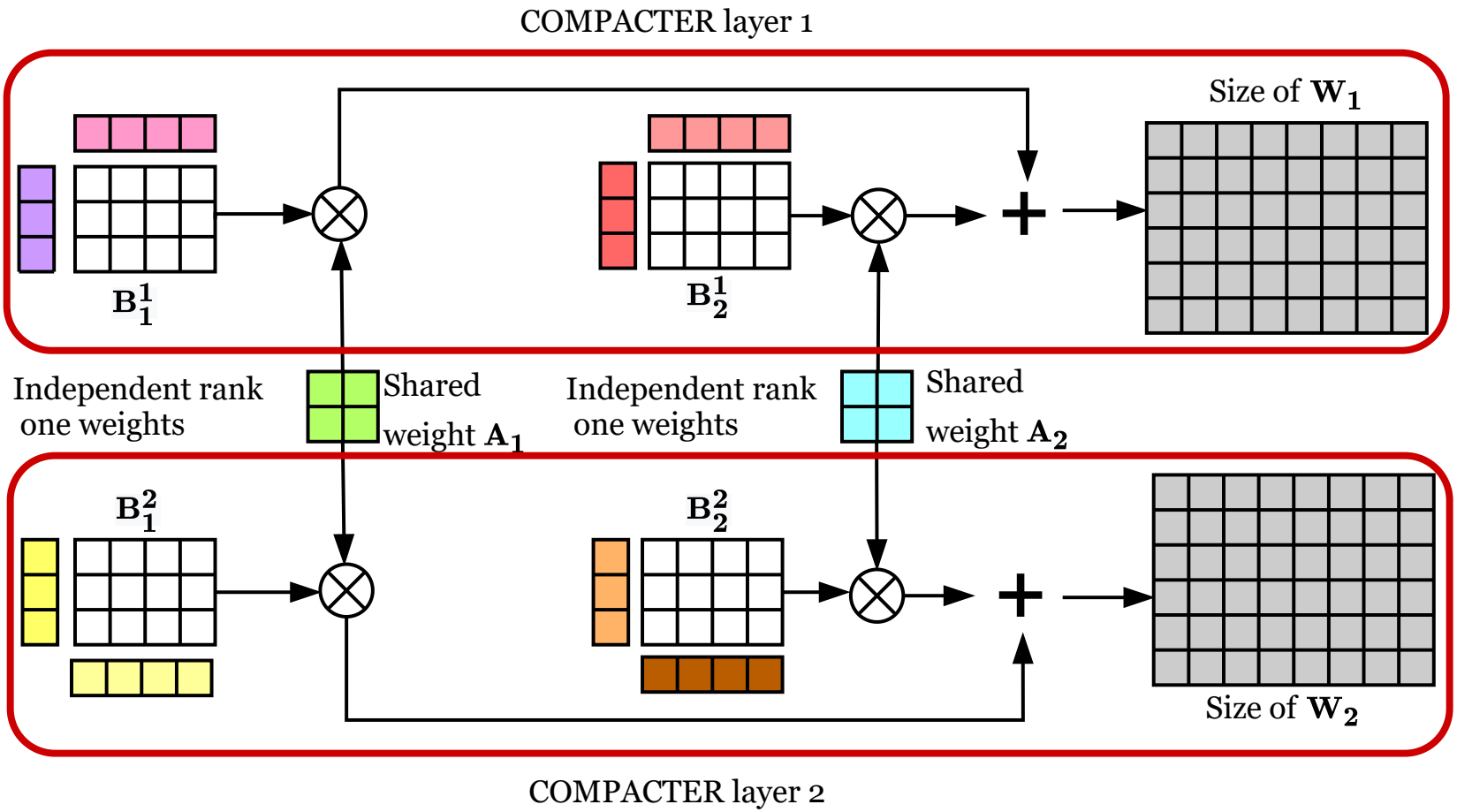
Compacter  : Beyond Hypercomplex Adapters

- Each COMPACTER layer's weight consists of:
  - Shared Weights ($\boldsymbol{A_i}$):
    - Common across all adapter layers
    - Capturing useful information for adapting to the target task
  - Low-rank Weights ($\boldsymbol{B_i}$):
    - *Adapter-specific* parameters
    - Capturing information relevant for adapting each individual layer
- Low-rank parameterized hypercomplex multiplication layers (LPHM):

$$\boldsymbol{W} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \boldsymbol{B_i} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes (\boldsymbol{s_i} \boldsymbol{t_i}^{\top}).$$

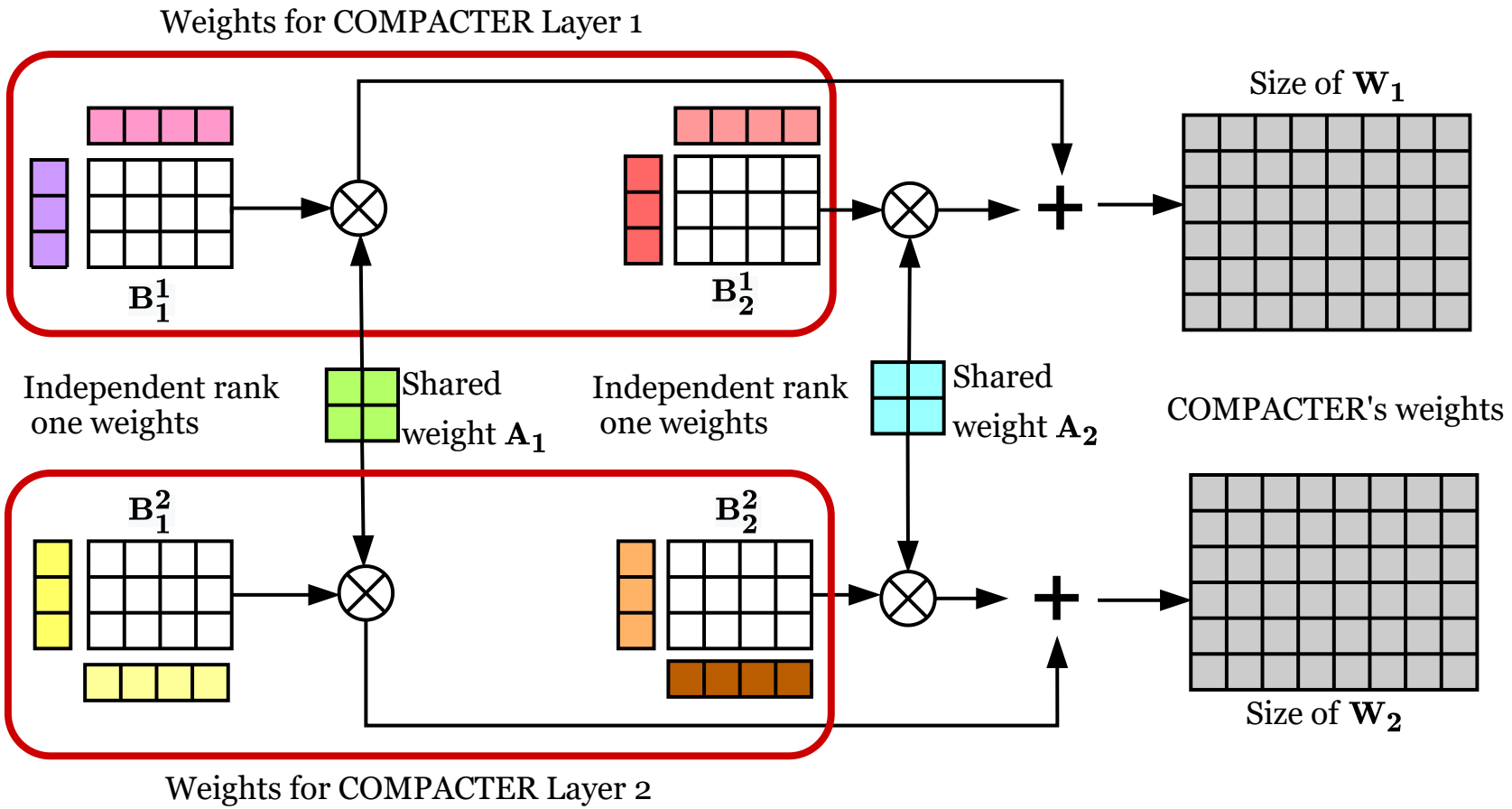
COMPACTER

- We compute sum of Kronecker products of *shared* matrices A_i and *adapter-specific* matrices B_i^j



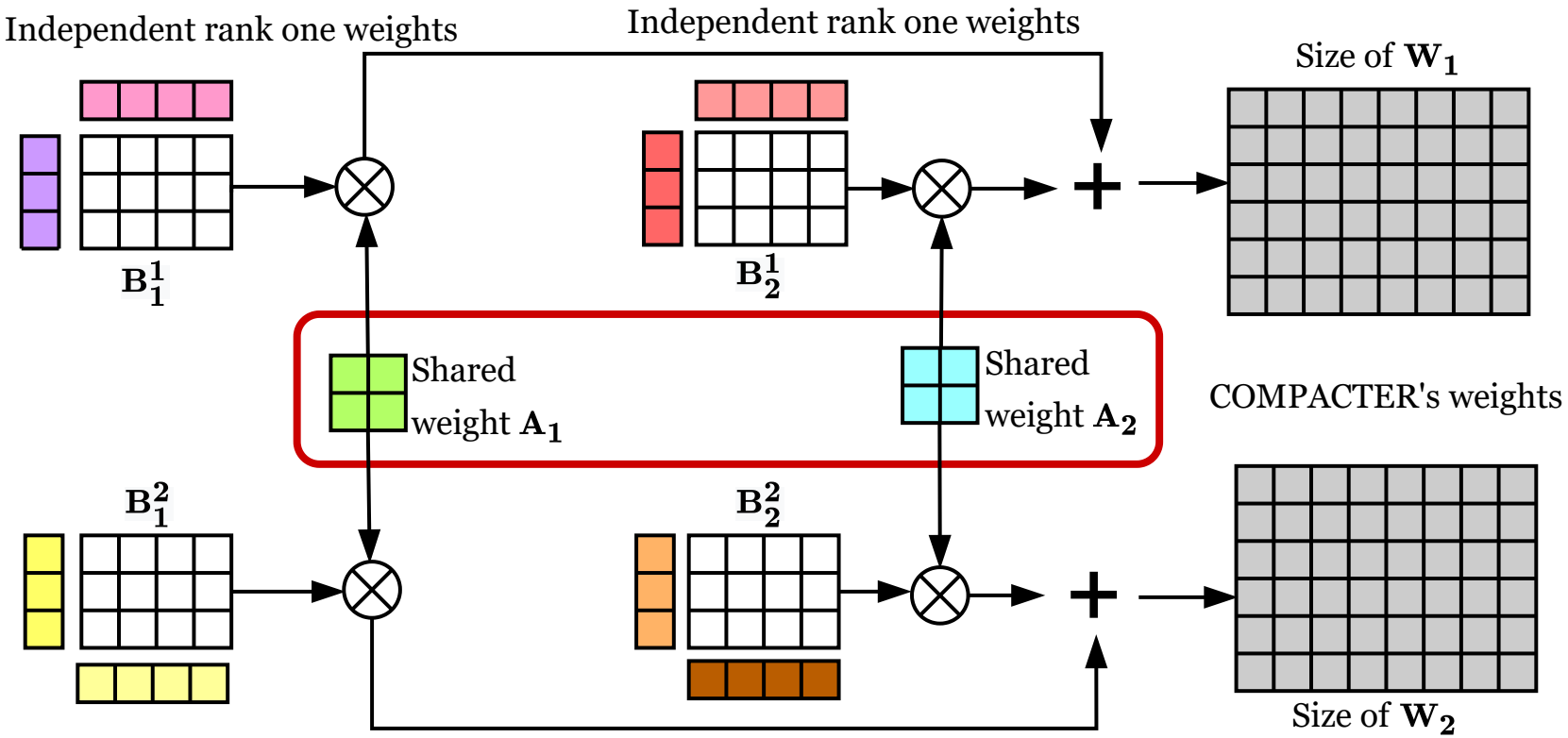
COMPACTER

- "Fast" weights B_i:
  - Independent rank-one weights
  - Learns adapter-layer specific information



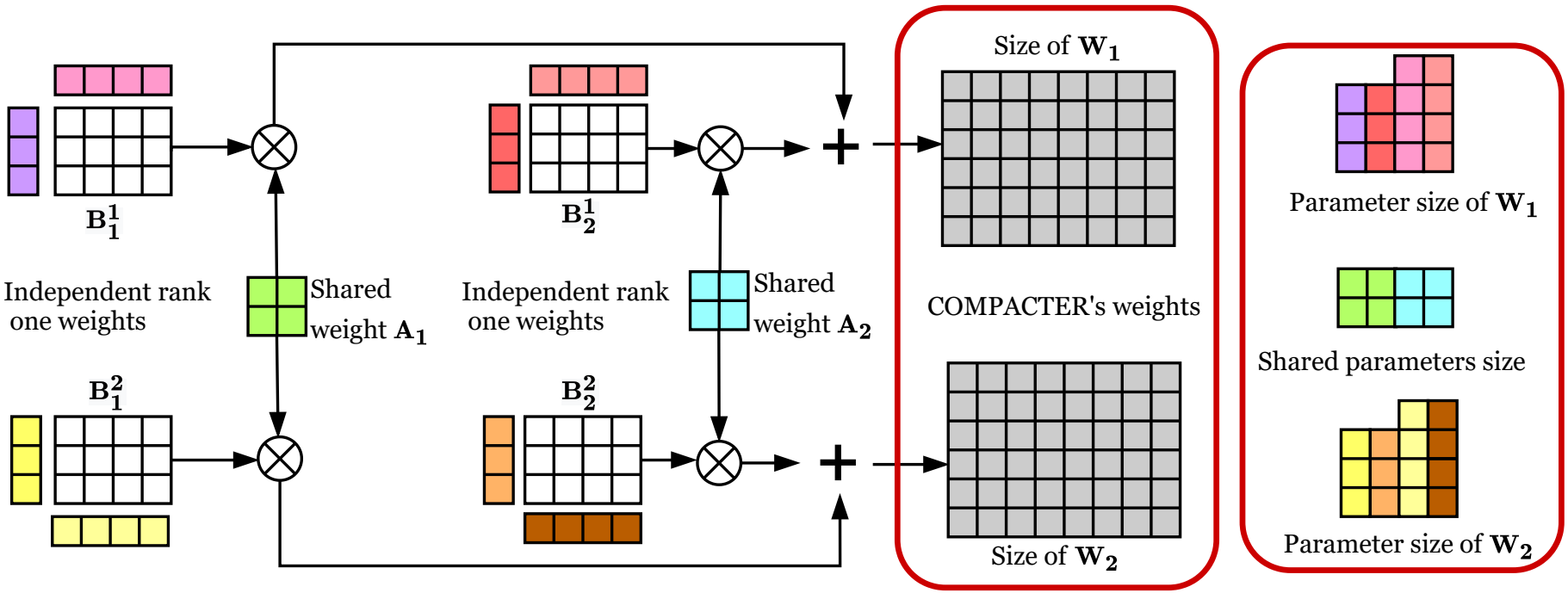
COMPACTER

- "Slow" shared weights A_i:
  - Shared across all COMPACTER layers
  - Capture general information useful for adapting to the target task



COMPACTER

- Parameter size of COMPACTER weights is much smaller than the size of the weights.



Parameter Efficiency

For a transformer of L layers and adapters of size $k \times d$:

- ADAPTER parameters:
  - $2kd$ parameters for down and up projections (encoder/decoder): $4kd$
  - Total parameters' complexity: $\mathcal{O}(\boldsymbol{Lkd})$
- PHM-ADAPTER
  - $\boldsymbol{A_i} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{B_i} \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}$ define the degree of freedom for $\boldsymbol{W}$
  - Total adapters' parameters: $4L \times (\frac{kd}{n} + n^3)$
  - With a mild assumption $kd > n^4$: $\mathcal{O}(\frac{1}{\boldsymbol{n}}\boldsymbol{Lkd})$
- COMPACTER 
  - $A_i \in \mathbb{R}^{n \times n}$ for all layers: n^3
  - Two rank-one weights for each adapter: $4L(k+d)$
  - Total parameters: $4L(k+d)+n^3$
  - With a mild assumption $4L(k+d) > n^3$: $\mathcal{O}(\boldsymbol{L}(\boldsymbol{k}+\boldsymbol{d}))$

Benchmarking Parameter-efficient Methods

Our Proposed Methods:

- **Compacter:** We learn adapter weights using LPHM layers.
- **Compacter++:** Removing COMPACTER layers after the self-attention layer.
- **PHM-Adapter:** We learn adapters' weight using PHM layers [2].

Baselines:

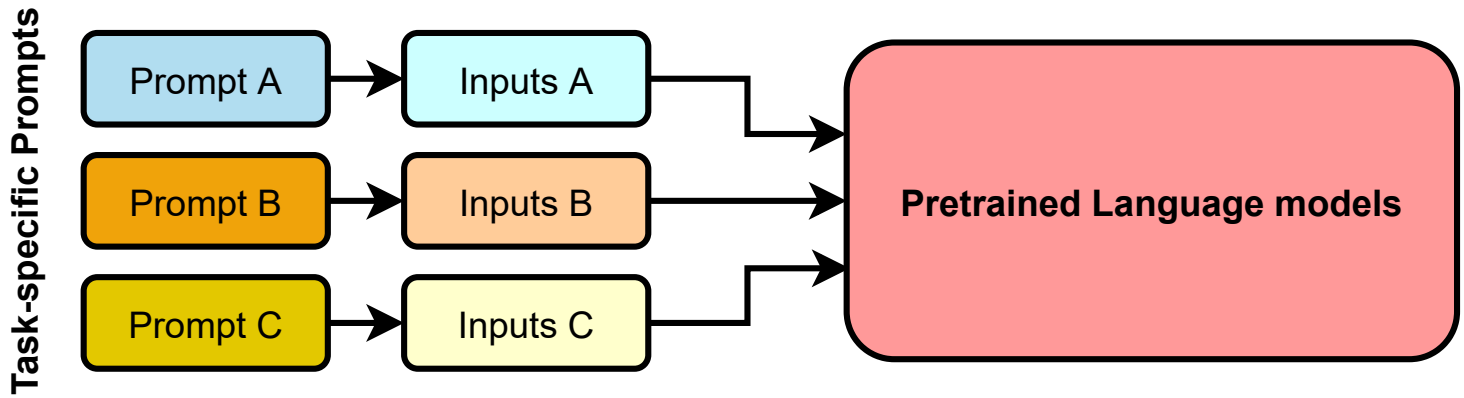
- **T5_{BASE}:** Fine-tuning all parameters of T5_{BASE} [4]
- **Adapter:** Including adapters after feedforward and self-attention [1]
- **Pfeiffer-Adapter:** Including adapters only after self-attention [5]
- **AdapterDrop:** Dropping adapters from lower transformer layers (first 5 layers) [3]
- **Adapter-LowRank:** Adapter's weights parameterized as a product of two rank-one weights.
- **BitFit:** Fine-tuning only biases [6, 7].

Benchmarking Parameter-efficient Methods

- **Intrinsic-SAID**: reparameterize in a low-dimensional subspace $\boldsymbol{\theta}^{\boldsymbol{d'}}$ $(d' \ll D)$ [8]:

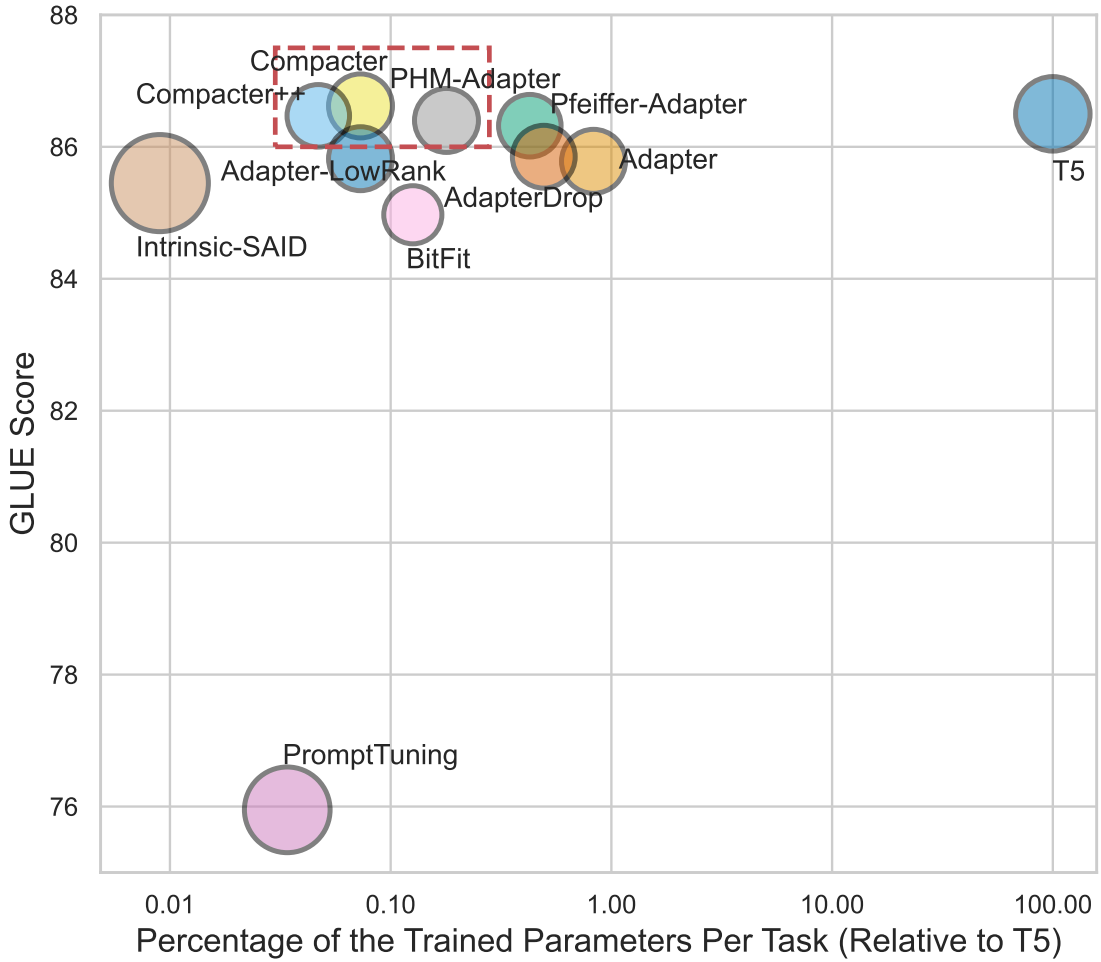
$$\boldsymbol{\theta}_{\boldsymbol{i}}^{\boldsymbol{D}} = \boldsymbol{\theta}_{\boldsymbol{i},0}^{\boldsymbol{D}} + \lambda_i \boldsymbol{P} \boldsymbol{\theta}_{\boldsymbol{i}}^{\boldsymbol{d'-m}},$$

  - Parameter $\boldsymbol{\theta}_{\boldsymbol{i},0}^{\boldsymbol{D}}$ are the pretrained model's parameters
  - $\boldsymbol{P} \in \mathbb{R}^{d'-m} \rightarrow \mathbb{R}^{D}$ is a random linear projection via the Fastfood transform
  - The total trainable parameters are $\boldsymbol{\theta}^{\boldsymbol{d'-m}} \in \mathbb{R}^{d'-m}$ and $\boldsymbol{\lambda} \in \mathbb{R}^{m}$
- **Prompt Tuning:** Prepends a randomly initialized continuous prompt to the input [9].
  - Initializing prompts from pretrained language model's vocabulary



Trade-off Between Parameter Efficient Fine-tuning Methods

- Trade-off between quantitative performance (score on GLUE (y axis))
- Percentage of trained parameters (x axis, in log scale)
- Memory footprint (size of the circles).



Performance Evaluation: COMPACTER (++)

- Performs on par with full fine-tuning.
- Outperforms all previous parameter-efficient methods.
- Only trains 0.07% (0.047%) of parameters.
- Reduces memory usage and speeds up the training.

Model	Trained params/ per task	Avg	Memory (MB)	Δ%	Time/ Epoch (min)	Δ%
$T5_{BASE}$	100%	86.5	167.99	—	42.13	—
ADAPTER	0.832%	85.78	124.02	-35.45%	31.81	-24.50%
PFEIFFER-ADAPTER	0.427%	86.32	118.4	-41.88%	28.19	-33.09%
ADAPTERDROP	0.494%	85.85	119.41	-40.68%	28.08	-33.35%
ADAPTER-LOWRANK	0.073%	85.82	123.8	-35.69%	32.71	-22.36%
PROMPT TUNING	0.034%	75.95	222.27	24.42%	44.54	5.72%
INTRINSIC-SAID	0.009%	85.45	285.40	41.14%	144.01	241.82%
BITFIT	0.126%	84.97	102.31	-64.20%	27.36	-35.06%
PHM-ADAPTER	0.179%	86.40	123.93	-35.55%	35.55	-15.62%
COMPACTER	0.073%	86.62	123.91	-35.57%	36.48	-13.41%
COMPACTER++	0.047%	86.47	118.35	-41.94%	30.96	-26.51%

Performance Evaluation: Prompt Tuning

- Low number of parameters but high memory overhead and slow to train
  - Computation of self-attention scales quadratically with the sequence length
- Its performance substantially lags behind full fine-tuning
  - High sensitivity to initialization and learning rate
  - Limited interaction with the model
  - Less suitable to deal with large contexts

Model	Trained params/ per task	Avg	Memory (MB)	Δ%	Time/ Epoch (min)	Δ%
$T5_{BASE}$	100%	86.5	167.99	—	42.13	—
Adapter	0.832%	85.78	124.02	-35.45%	31.81	-24.50%
Pfeiffer-Adapter	0.427%	86.32	118.4	-41.88%	28.19	-33.09%
AdapterDrop	0.494%	85.85	119.41	-40.68%	28.08	-33.35%
Adapter-LowRank	0.073%	85.82	123.8	-35.69%	32.71	-22.36%
Prompt Tuning	0.034%	75.95	222.27	24.42%	44.54	5.72%
Intrinsic-SAID	0.009%	85.45	285.40	41.14%	144.01	241.82%
BitFit	0.126%	84.97	102.31	-64.20%	27.36	-35.06%
PHM-Adapter	0.179%	86.40	123.93	-35.55%	35.55	-15.62%
Compacter	0.073%	86.62	123.91	-35.57%	36.48	-13.41%
Compacter++	0.047%	86.47	118.35	-41.94%	30.96	-26.51%

Performance Evaluation: Intrinsic-SAID

- Tunes only 0.009% of parameters
- Performs worse than fine-tuning
- High memory overhead and slow to train
  - Requires storing large random projection matrices.
  - Computing projections via FastFood transform [10] is slow in practice
  - Not suitable for large-scale pretrained language models

Model	Trained params/ per task	Avg	Memory (MB)	Δ%	Time/ Epoch (min)	Δ%
$T5_{BASE}$	100%	86.5	167.99	—	42.13	—
Adapter	0.832%	85.78	124.02	-35.45%	31.81	-24.50%
Pfeiffer-Adapter	0.427%	86.32	118.4	-41.88%	28.19	-33.09%
AdapterDrop	0.494%	85.85	119.41	-40.68%	28.08	-33.35%
Adapter-LowRank	0.073%	85.82	123.8	-35.69%	32.71	-22.36%
Prompt Tuning	0.034%	75.95	222.27	24.42%	44.54	5.72%
Intrinsic-SAID	0.009%	85.45	285.40	41.14%	144.01	241.82%
BitFit	0.126%	84.97	102.31	-64.20%	27.36	-35.06%
PHM-Adapter	0.179%	86.40	123.93	-35.55%	35.55	-15.62%
Compacter	0.073%	86.62	123.91	-35.57%	36.48	-13.41%
Compacter++	0.047%	86.47	118.35	-41.94%	30.96	-26.51%

Performance Evaluation: BitFit

- Performs worse than fine-tuning (-1.53 points).
  - Tuning only biases is not sufficient
- Lowest memory overhead and the fastest to train
  - Does not store intermediate activations.

Model	Trained params/ per task	Avg	Memory (MB)	Δ%	Time/ Epoch (min)	Δ%
$T5_{BASE}$	100%	86.5	167.99	—	42.13	—
Adapter	0.832%	85.78	124.02	-35.45%	31.81	-24.50%
Pfeiffer-Adapter	0.427%	86.32	118.4	-41.88%	28.19	-33.09%
AdapterDrop	0.494%	85.85	119.41	-40.68%	28.08	-33.35%
Adapter-LowRank	0.073%	85.82	123.8	-35.69%	32.71	-22.36%
Prompt Tuning	0.034%	75.95	222.27	24.42%	44.54	5.72%
Intrinsic-SAID	0.009%	85.45	285.40	41.14%	144.01	241.82%
BitFit	0.126%	84.97	102.31	-64.20%	27.36	-35.06%
PHM-Adapter	0.179%	86.40	123.93	-35.55%	35.55	-15.62%
Compacter	0.073%	86.62	123.91	-35.57%	36.48	-13.41%
Compacter++	0.047%	86.47	118.35	-41.94%	30.96	-26.51%

Performance Evaluation: ADAPTER-based methods

- Low memory-overhead and fast to train
- Generally perform worse than finetuning (exception:PFEIFFER-ADAPTER)
  - ADAPTERDROP: Adapting lower layer of T5 is important.
  - ADAPTER-LOWRANK is not expressive enough.
- Order of magnitude more trainable parameters cf. COMPACTER++

Model	Trained params/ per task	Avg	Memory (MB)	Δ%	Time/ Epoch (min)	Δ%
$T5_{BASE}$	100%	86.5	167.99	—	42.13	—
ADAPTER	0.832%	85.78	124.02	-35.45%	31.81	-24.50%
PFEIFFER-ADAPTER	0.427%	86.32	118.4	-41.88%	28.19	-33.09%
ADAPTERDROP	0.494%	85.85	119.41	-40.68%	28.08	-33.35%
ADAPTER-LOWRANK	0.073%	85.82	123.8	-35.69%	32.71	-22.36%
PROMPT TUNING	0.034%	75.95	222.27	24.42%	44.54	5.72%
INTRINSIC-SAID	0.009%	85.45	285.40	41.14%	144.01	241.82%
BITFIT	0.126%	84.97	102.31	-64.20%	27.36	-35.06%
PHM-ADAPTER	0.179%	86.40	123.93	-35.55%	35.55	-15.62%
COMPACTER	0.073%	86.62	123.91	-35.57%	36.48	-13.41%
COMPACTER++	0.047%	86.47	118.35	-41.94%	30.96	-26.51%

Low-resource Fine-tuning

- Subsampling GLUE for varying sizes (100,500,1000,2000,4000).
- COMPACTER++:
  - Generalizes substantially better in resource-limited settings.
  - Offers a more effective fine-tuning in this regime.

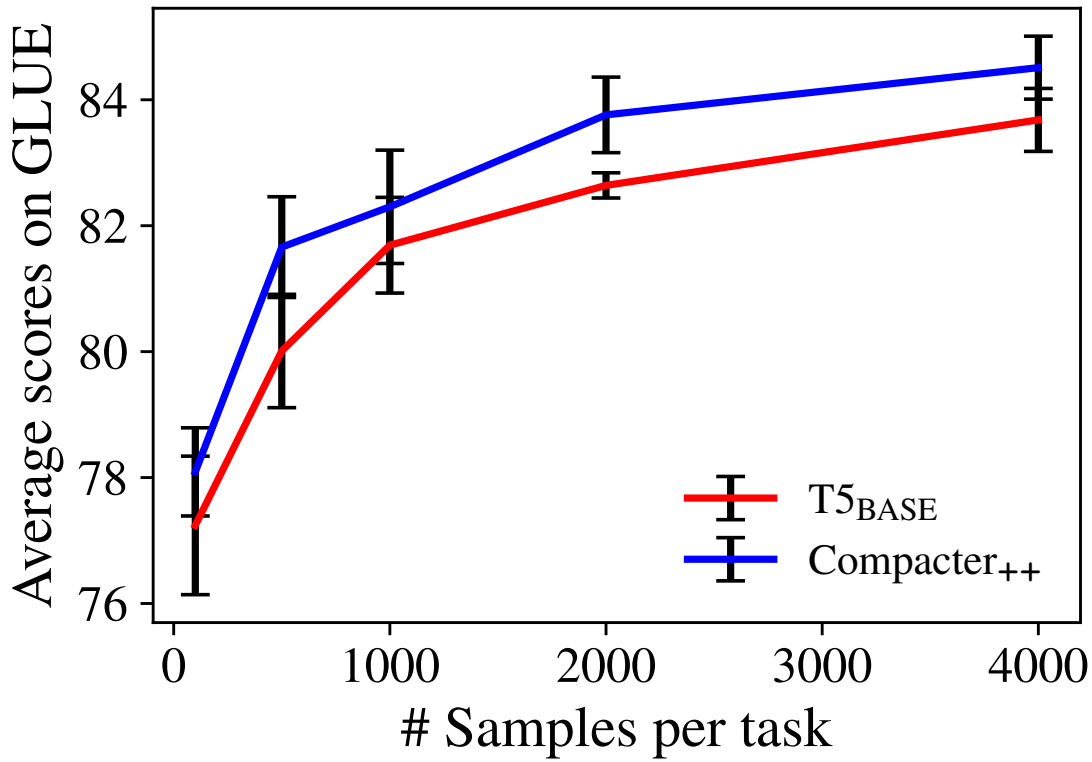


Figure: Results on GLUE for low-resource setting.

Takeaways

COMPACTER (++)

- Is a light-weight fine-tuning method for large-scale language models.
- Generates adapter's weights by summing Kronecker products between:
  - shared "slow" weights
  - "fast" rank-one matrices, specific to each adapter layer.
- Reduces the number of parameters substantially from $\mathcal{O}(kd)$ to $\mathcal{O}(k+d)$.
- Learns only 0.073% (0.047%) parameters, still:
  - Obtains comparable performance in a full-data setting.
  - Outperforms fine-tuning in data-limited scenarios.

Questions?

Please join our poster presentation during NeurIPS, 2021.

References I

[1] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.

[2] Aston Zhang, Yi Tay, SHUAI Zhang, Alvin Chan, Anh Tuan Luu, Siu Hui, and Jie Fu. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with 1/n parameters. In *ICLR*, 2021.

[3] Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. AdapterDrop: On the Efficiency of Adapters in Transformers. *arXiv preprint arXiv:2010.11918*, 2020.

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

[5] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Cho Kyunghyun, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *EACL*, 2021.

[6] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. *NeurIPS*, 2020.

[7] Shauli Ravfogel, Elad Ben-Zaken, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked languagemodels. 2021.

References II

[8] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

[9] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

[10] Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood-approximating kernel expansions in loglinear time. In *ICML*, 2013.

[11] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. Revisiting Few-sample BERT Fine-tuning. In *ICLR*, 2021.

[12] Hyung Won Chung, Thibault Févry, Henry Tsai, Melvin Johnson, and Sebastian Ruder. Rethinking Embedding Coupling in Pre-trained Language Models. In *ICLR*, 2021.