Meta-Learning the Difference: Preparing Large Language Models for Efficient Adaptation

Zejiang Hou
Princeton University*
zejiangh@princeton.edu

Julian Salazar
Amazon AWS AI
julsal@amazon.com

George Polovets
Amazon AWS AI
polovg@amazon.com

Abstract

Large pretrained language models (PLMs) are often domain- or task-adapted via finetuning or prompting. Finetuning requires modifying all of the parameters and having enough data to avoid overfitting while prompting requires no training and few examples but limits performance. Instead, we prepare PLMs for data- and parameter-efficient adaptation by *learning to learn the difference* between general and adapted PLMs. This difference is expressed in terms of model weights and sublayer structure through our proposed dynamic low-rank reparameterization and learned architecture controller. Experiments on few-shot dialogue completion, low-resource abstractive summarization, and multi-domain language modeling show improvements in adaptation time and performance over direct finetuning or preparation via domain-adaptive pretraining. Ablations show our task-adaptive reparameterization (TARP) and model search (TAMS) components individually improve on other parameter-efficient transfer like adapters and structure-learning methods like learned sparsification.

1 Introduction

Finetuning large pretrained language models (PLMs) on task-specific supervised data has become the default strategy to produce performant models for various NLP tasks (Dai and Le, 2015; Howard and Ruder, 2018; Radford et al., 2019, *inter alia*), provided a task has enough training data to be adapted to without overfitting. For few-shot tasks, very large PLMs like the 175B-parameter GPT-3 (Brown et al., 2020) do surprisingly well without training using *prompts*, where task-specific examples (x_j, y_j) are presented as text to condition the PLM before a test input x_{test} is given. Our work considers an important middle ground: minimizing the computational cost of finetuning while improving on its performance in low-resource and few-shot settings.

In general, self-supervised objectives used for PLMs assume little about the nature of downstream tasks. Earlier works suggested that task-awareness is unnecessary for PLMs of sufficient scale; e.g., Raffel et al. (2020) found that multi-task learning underperformed pretrain-finetune for the largest T5 models on multi-format question answering. However, Gururangan et al. (2020) showed that further pretraining on unlabeled text from the downstream task (task-adaptive pretraining, or TAPT) or a related domain (DAPT) consistently improved adaptation performance. Aghajanyan et al. (2021a) revisited Raffel et al. and found that by greatly improving the number and balance of tasks, one can utilize a multitask objective after pretraining and achieve gains in proportion to the number of tasks. As for even larger models, Brown et al. (2020) argue that the impressive few-shot prompting ability of GPT-3 comes from "implicit" meta-learning (Schmidhuber, 1987; Bengio et al., 1990) which they term *in-context learning*, where the outer loop is performed by self-supervised pretraining, and the inner loop is performed by forward passes on implicit examples in unlabeled texts.

These works motivate that exposure to broad information about downstream tasks remains useful in preparing a large PLM for adaptation. Hence, **we propose explicit meta-learning for preparing large PLMs for data-efficient adaptation**; a visual comparison is in Figure 1. To also achieve parameter efficiency and performance, **we adapt meta-transfer learning** (Sun et al., 2019) **to large PLMs in two proposed ways**: an inner loop optimizing a low-rank **task-adaptive reparameterization (TARP)** of weights, and an outer loop learning an architecture controller for searching **task-adaptive model structures (TAMS)**. These

*Work done during an internship at Amazon AWS AI.

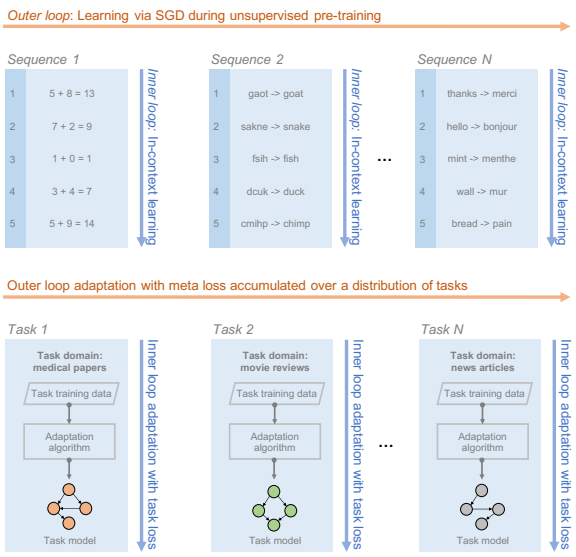

Figure 1: Comparison between (**top**) *implicit* meta-learning from text corpora that incidentally contain task "prefixes", as in GPT-3 (Brown et al., 2020; Fig. 1.1), and (**bottom**) *explicit* meta-learning the transformation of a PLM's weights and sublayers for a distribution of tasks.

improve over general finetuning and even DAPT-prepared LMs on generative and unconditional few-shot and low-resource settings, such as multi-domain abstractive summarization (AdaptSum; Yu et al., 2021) and language modeling.

Furthermore, our analysis shows that **each component of our task distribution-aware strategy independently improves over prior work**: (1) meta-transfer learning improves over model-agnostic meta learning (Finn et al., 2017) even after multitask learning on the same data, setting a new state-of-the-art on few-shot Persona-Chat dialog personalization (Zhang et al., 2018); (2) our proposed dynamic low-rank TARP outperforms recent methods such as MAM adapters (He et al., 2022) and alternate reparameterizations like Kronecker products (Zhang et al., 2021); (3) our lightweight controller for generating task-aware architectures in TAMS extends improvements into higher resource tasks and rediscovers task-specific modifications like 1D convolutions for Transformers.

Our proposal is summarized in Figure 2, with pseudocode in Algorithm 1 at the end of the next section. We publicly release the code for our experiments and our reference library online.[1]

2 Methodology

Our goal is to explicitly optimize a PLM for efficient adaptation to any task $\mathcal{T}_i$ sampled from a distribution of low-resource NLP tasks $p(\mathcal{T})$. Each task consists of a training set $\mathcal{D}_i^{\text{train}}$, a test set $\mathcal{D}_i^{\text{test}}$, and a loss function $\mathcal{L}_i$.

The prevailing approach for efficiently optimizing a base model f_Θ on a (relatively) small task-specific dataset is to use model-agnostic meta-learning (MAML; Finn et al., 2017). This is a bi-level optimization process that uses a stochastic gradient-based strategy to sample a batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B}$ from the task distribution $p(\mathcal{T})$ in each meta-iteration. In the inner loop, each task finetunes a copy of the model's weights Θ for a small number of steps T_{in}, producing task-specific weights Θ_i. In the outer loop, each task model f_{Θ_i} is evaluated on its corresponding task's test set $\mathcal{D}_i^{\text{test}}$ and these losses are summed to produce the overall meta-loss. The meta-loss $\sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_i^{\text{test}}(f_{\Theta_i})$ is then used to optimize and update Θ; see Weng (2018)[2] for a more detailed overview.

MAML, however, is not generally used in NLP as a competitive alternative to pretrain-then-finetune methods for low-resource and few-shot settings. To rectify MAML's limitations, we propose a meta-learning the difference (MLtD) framework to optimize PLMs for fast and data-efficient adaptations with the following contributions:

MAML after pretraining. Earlier works performed MAML using random initializations, or at best with pretrained token embeddings (Madotto et al., 2019), which was shown to underperform the pretrain-finetune paradigm. With the increased prevalence of large-scale pretraining, recent works have begun to initialize MAML with PLMs (Dou et al., 2019). We continue this approach, but further show that pretraining + MAML, even when labeled (i.e., multitask) and performed only on the meta-training data (i.e., no external text), improves performance and mitigates overfitting versus pretraining alone or MAML alone (Section 4), suggesting that pretraining produces a better initialization that promotes generalization in later meta-learning.

Parameter-efficient transfer. Adaptation data is typically limited, making it easy for large models to overfit. Previous works use very shallow CNNs

[1] https://github.com/amazon-research/meta-learning-the-difference

[2] https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html

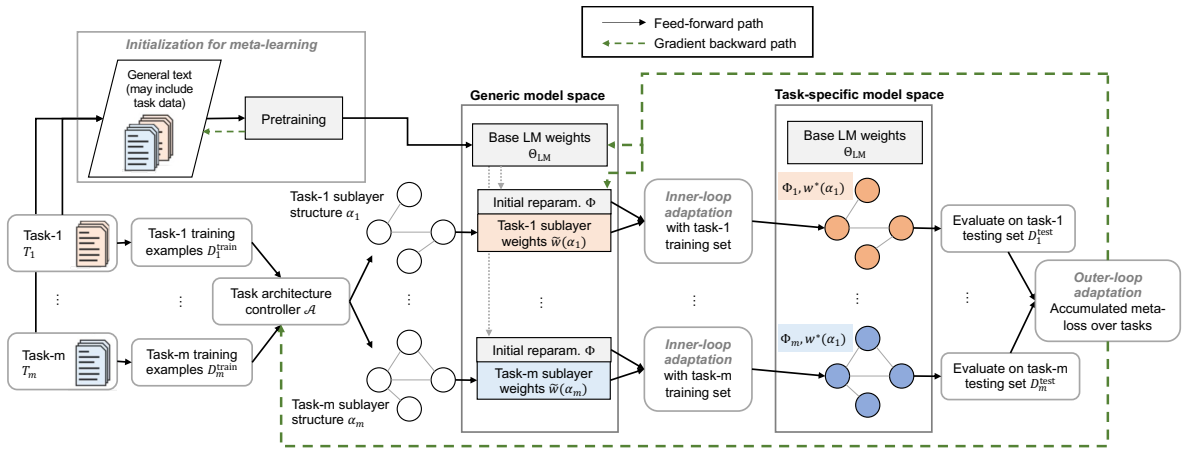

Figure 2: Overview of our proposed method, which learns to transform a small set of weights Φ_i (TARP learning) and modify sublayer modules α_i (TAMS learning) in a task-specific, data-efficient, and parameter-efficient manner. First, we initialize with a base PLM (**top left**). In each meta-iteration, we sample a batch of tasks from a task distribution (**left**). In the inner loop (**middle**), independent sets of dynamic low-rank reparameterizations are initialized, and an architecture controller generates independent task-specific sublayer modules, all of whose weights are adapted to the task's training set. Each task model is evaluated on the corresponding task's test set. In the outer loop (**right**), these task losses are summed up to produce the overall meta-loss, and the backward path optimizes the base model, the initial reparameterization, and the architecture controller.

(Finn et al., 2017), only adapt scale-and-shift parameters atop the original model (Sun et al., 2019), or apply various general regularization techniques such as weight decay, label smoothing, dropout, early stopping, and ℓ_1 regularization (Madotto et al., 2019; Song et al., 2020). In contrast, we propose learning dynamic low-rank reparameterizations g_{Φ_i} (Section 2.1) of the base model such that $\Theta_i(x) = g_{\Phi_i}(\Theta_{\text{LM}}, x)$ for task $\mathcal{T}_i$. Here, Φ is a small set of new parameters that are adapted into task-specific Φ_i when finetuning. Notably, we modify MAML to incorporate these parameter-efficient modules, so that during task adaptation in both meta-training (the inner loop) and meta-testing (novel tasks) $\mathcal{T}_i$, we only adapt $\Phi \rightarrow \Phi_i$ instead of $\Theta \rightarrow \Theta_i$, speeding up both phases and improving overall performance. Though some works explore the benefits of joint training or fusion of parameter-efficient modules (Stickland and Murray, 2019; Lin et al., 2020; Pfeiffer et al., 2021), prior work has not explored meta-learning to learn these adaptations in a task distribution-aware setting.

Architecture adaptation. While the Transformer has proven to be a robust general-purpose architecture, recent work has shown that the optimal attention-then-FFN sublayer structure can vary across tasks (e.g., Sandwich Transformers; Press et al., 2020). However, previous data-driven sublayer searches are often task-agnostic (e.g., So et al., 2021), where the sublayer search is implemented before pretraining. Meta-learning enables learning data-driven sublayers *after* pretraining, in a differentiable, task-adaptive manner (Section 2.2). Instead of a separate search per task as in previous methods (DARTS; Liu et al., 2019a), we propose meta-learning a task-aware architecture controller to help it generalize to new tasks when searching neural architectures, by learning to directly generate task-specific sublayer structures from the dataset. By exploiting architectural knowledge learned over the task distribution, our task-adaptive model structure approach improves test-time performance. A related work in customizing model structure is CMAML (Song et al., 2020), which applies a sparse pruning algorithm to obtain task-specific weight masks. Our method differs in that we consider generalization over a distribution of tasks (instead of a single task), and has a richer search space with different operations, numbers of layers, and widths of layers, so that our method provides architecture diversity to accommodate to the different task data.

In all, we employ meta-learning to improve upon initializing from a pretrained Θ_{LM}, allowing better downstream finetuning on tasks. By learning only the transformation weights Φ_i and (optionally) the task-specific architecture α_i for new tasks $\mathcal{T}_i$, our

method "learns to learn the difference" between a PLM and a task-specific LM in a training-efficient way.

2.1 Efficient parameter adaptation

We categorize recent works in parameter-efficient adaptation of large PLMs into three types:

Adding parameter-efficient layers. Low-dimensional adapters (Rebuffi et al., 2018) have been injected into a frozen pretrained BERT either serially after each sublayer (Houlsby et al., 2019), or in parallel to the self-attention layers (PALs; Stickland and Murray, 2019). Following works (Bapna and Firat, 2019; Lin et al., 2020) applied adapters to other NLP models, e.g. GPT-2. Compacters (Mahabadi et al., 2021) reduce the adapter parameter count via hypercomplex multiplications (Zhang et al., 2021).

Adding parameter-efficient prefixes. Inspired by prompting, the learning of automated prompts or task-specific continuous variants has been applied for encoder-only PLMs like BERT (Shin et al., 2020; Hambardzumyan et al., 2021) and generative PLMs (Li and Liang, 2021; Liu et al., 2021; Lester et al., 2021) where one learns task-specific vectors prepended to inputs or hidden representations.

Transformations only. The adapter and prefix-tuning strategies insert layers or introduce prefixes, increasing inference time or in-memory size. Instead, Zhao et al. (2020) learn binary masks, and diff pruning (Guo et al., 2021) learns sparse additive vectors. Both methods use unstructured sparsity to achieve parameter efficiency. Later works like BitFit (Zaken et al., 2021) and LoRA (Hu et al., 2022) introduce parameter-efficient modifications targeting the Transformer architecture: BitFit only tunes the bias parameters, while LoRA adds low-rank decomposition weights to the self-attention weights. Recently, He et al. (2022) proposed parallel mix-and-match (MAM) adapters which leverage benefits of the preceding types.

Hence, to minimize overhead we focus on a "transformations only" approach. Inspired by the scale-and-shift parameters of Sun et al. (2019), we propose learning *affine* transformations to reparameterize the pretrained model weights towards a task. For a pretrained weight matrix $W_0^l \in \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$ (can be any dense layer in the self-attention module or the FFN module in a transformer based architecture), we first reparameterize the task-specific weights as:

$$W^l = \Phi_1^l \odot W_0^l + \Phi_2^l \tag{1}$$

where $\Phi_1^l, \Phi_2^l \in \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$ and $\odot$ denotes the elementwise (Hadamard) product.

At adaptation time, we apply low-rank constraints while optimizing the reparameterization weights only, giving the training objective

$$\min_{\substack{\{\Phi_1^l, \Phi_2^l\}_{l=1}^L \\ \text{rank}(\Phi_i^l) < r}} \sum_{t=1}^{T} \log p(y_t | x, y_{<t}; \{W^l\}_{l=1}^L). \tag{2}$$

A straightforward approach to solve the rank-constrained problem is to apply a low-rank decomposition to the transformation weights Φ_i^l. We term this approach of learning parameter-efficient affine transformations **task-adaptive reparameterization (TARP)**. We consider two standard static decomposition methods:

- **Bilinear**, which takes $\Phi_j^l = U_j^l V_j^{l^T}$ where $U_j^l \in \mathbb{R}^{C_{\text{in}} \times r}$ and $V_j^l \in \mathbb{R}^{C_{\text{out}} \times r}$, as done in the additive-only setting ($\Phi_1^l = I$) by LoRA.
- **Kronecker product**, which takes $\Phi_j^l = \sum_{k=1}^n H_k^l \otimes (U_k^l V_k^{l^T})$ where $H_k \in \mathbb{R}^{n \times n}$, $U_k^l \in \mathbb{R}^{(C_{\text{in}}/n) \times r}$, $V_k^l \in \mathbb{R}^{(C_{\text{out}}/n) \times r}$, and n is a hyperparameter, as used in the "added-layer" Compacter approach.

In addition, we propose a novel decomposition inspired by the self-attention mechanism, which aggregate features using input-dependent attention weights and can be regarded as a function using input-dependent parameters. Similarly, the optimal reparameterization of the base model may vary with different input values. To account for this, our TARP parameters are modeled by a **dynamic low-rank decomposition** (Figure 3):

$$\Phi_j^l(x) = U_j^l \Sigma_j^l(x) V_j^{l^T}, \tag{3}$$

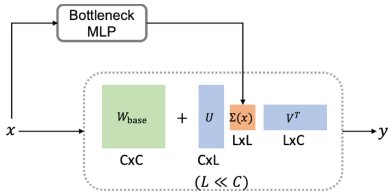

Figure 3: TARP with dynamic decomposition (only the additive Φ_2^l is depicted for simplicity).

The square matrices $\Sigma_j^l(x) \in \mathbb{R}^{r\times r}$ are generated by a lightweight multi-layer perceptron (MLP) for different input vectors and U_j^l, V_j^l are the learnable weight matrices with $r \ll \min(C_{\text{in}}, C_{\text{out}})$.

We compare popular parameter-efficient transfer schemes and these three decompositions in Section 4.

2.2 Efficient architecture adaptation

We also propose adapting the model structure for each task in a data-driven manner. The weights of the *task-specific architectures* are learned in the inner loop, while the *task-aware architecture generator* which produces architecture candidates is learned in the outer loop. We term this approach **task-adaptive model structure (TAMS)** learning.

We first represent each task $\mathcal{T}_i$ with an embedding vector z_i based on the task training set D_i^{train}. An embedding module $\mathcal{E}$ computes the task representation by aggregating features of all training data:

$$z_i = \mathcal{E}(D_i^{\text{train}}) = \frac{\sum_{(x,y)\in D_i^{\text{train}}} \text{Embed}(x)}{|D_i^{\text{train}}|}, \quad (4)$$

where $\text{Embed}(x)$ are intermediate representations produced by the PLM. For encoder-decoder models (Transformer), we take Embed to be the encoder; for encoder-only or decoder-only PLMs (BERT, GPT-2), we use the token embedding layer.

Inspired by DARTS (Liu et al., 2019a), we define the possible sublayer structures by a search space expressed as a directed acyclic graph (DAG), where each directed edge corresponds to a set of candidate operations $\mathcal{O}$. The task architecture is represented by a set of parameters α_i that encode the structure, where $\alpha_i \in \mathbb{R}^{E\times|\mathcal{O}|}$ (E is the number of edges, and $|\mathcal{O}|$ is the number of operations). In our proposed TAMS approach we also introduce a controller $\mathcal{A}$ to generate these task-specific architecture parameters, as a function of the task embedding vector $\alpha_i = \mathcal{A}(z_i)$. The probability of choosing operation m in edge n is given by $P_n(m) = \text{softmax}_{m\in\mathcal{O}}(\alpha_i[n,m])$. In meta-testing, the discrete architecture is obtained by taking the argmax. Since argmax is non-differentiable, we use the straight-through Gumbel-Softmax estimator to backpropagate gradients for optimizing the architecture controller during meta-training.

In TAMS, all possible architectures are initialized as part of the meta-parameters $\tilde{w}$ based on weight-sharing (Pham et al., 2018), i.e., architecture α_i's weights $\tilde{w}(\alpha_i)$ are selected from the meta-parameters. After the reparameterization steps in TARP and the architecture generation steps in TAMS, our inner loop optimization takes parameters $(\Phi, \tilde{w}(\alpha_i))$ and performs a small number of gradient steps T_{in} on the task training set to give $(\Phi_i, \tilde{w}_i)$. In the outer loop optimization, we thus have to simultaneously optimize the architecture controller to perform architecture search, as well as the parameter initialization. This is in contrast to MAML which just optimizes the parameter initialization in the outer loop. The meta-loss becomes: $\min_{\mathcal{W}} \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{D_i^{\text{test}}}(f_{\Theta_{\text{LM}}\cup\Phi_i\cup\tilde{w}_i})$, where the tuple $\mathcal{W}$ contains the base PLM's weights Θ_{LM}, the low-rank reparameterization weights Φ, the architecture controller $\mathcal{A}$, and the weight-sharing meta-parameters $\tilde{w}$.

In summary, our contributions with the TAMS framework are that (1) it meta-learns a task-aware controller by training on the task distribution and then generalizes to new tasks by automatically generating an optimized architecture α_i from the task training data, and (2) it optimizes the controller and parameter initialization (shared by all tasks) simultaneously under a unified meta-learning objective. This is in contrast to DARTS, which performs a separate search and architecture parameter optimization for each task independently.

We summarize our net method with TARP and TAMS as pseudocode in Algorithm 1.

3 Main results

To demonstrate the overall benefit of our method, we compare our results to other approaches on generative adaptation tasks in the few-shot (dialogue personalization), low-resource (abstractive summarization), and medium-resource (multi-domain language modeling) regimes. In Section 4 we perform some analyses and also compare TARP by itself to previous parameter-efficient works.

3.1 Implementation

All of our experiments ran using PyTorch on single machines with 32GB NVIDIA V100 GPUs. See per-task hyperparameters in Appendix A.1 and our code release.

TARP decomposition. We apply task-adaptive reparameterization (TARP) to the pretrained self-attention and feed-forward network (FFN) blocks. In Section 4.2 we conclude that TARP with dynamic decomposition outperforms other parameter-efficient transfer methods; TARP will always be

Algorithm 1: Meta-Learning the Difference (MLtD) with TARP and TAMS.

Require: pretraining dataset D^{pre}; meta-training dataset of tasks D^{meta}; base model (LM) f_Θ; TARP weights Φ; embedding module $\mathcal{E}$ in TAMS; architecture controller $\mathcal{A}$ in TAMS; meta-parameters $\tilde{w}$ in TAMS; inner-/outer-loop learning rate $\eta_{\text{in}}/\eta_{\text{out}}$; meta-training iterations T_{meta}; inner-loop iterations T_{in}; meta-batch size B;

/* Pretraining phase in MLtD */

Pretrain the base LM's weights $\Theta \rightarrow \Theta_{\text{LM}}$ on D^{pre};

// In contrast, MAML runs on random initialization.

/* Meta-training phase in MLtD */

for *each meta-iteration* $t \in [T_{meta}]$ **do**

  Sample a batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B}$ from D^{meta};

  $\mathcal{L}_{\text{meta\_loss}} = 0$;

  for $\mathcal{T}_i \in \{\mathcal{T}_i\}_{i=1}^{B}$ **do**

    Initialize $\Phi_i = \Phi$;

    Reparameterize Θ_{LM} with Φ_i (Eq.1,3);

    Expand task-specific sublayers by TAMS-generated architecture $\alpha_i = \mathcal{A}(\mathcal{E}(D_i^{\text{train}}))$;

    Initialize sublayer's weights: $\tilde{w}_i = \tilde{w}(\alpha_i)$;

    // In contrast, MAML does not adapt model architecture to a task.

    for T_{in} *iterations* **do**

      $\mathcal{L}_{\text{inner}} = \mathcal{L}_{D_i^{\text{train}}}\left(f_{\Theta_{\text{LM}} \cup \Phi_i \cup \tilde{w}_i}\right)$;

      $\left(\Phi_i, \tilde{w}_i\right)$ -= $\eta_{\text{in}} \nabla_{(\Phi_i, \tilde{w}_i)} \mathcal{L}_{\text{inner}}$;

      // In contrast, MAML updates all parameters, but we only update a small number in the inner loop.

    Evaluate on D_i^{test}:

    $\mathcal{L}_{\text{meta\_loss}}$ += $\mathcal{L}_{D_i^{\text{test}}}\left(f_{\Theta_{\text{LM}} \cup \Phi_i \cup \tilde{w}_i}\right)$;

  Perform outer-loop optimization:

  $\left(\Theta_{\text{LM}}, \Phi, \mathcal{A}, \tilde{w}\right)$ -= $\eta_{out} \nabla_{(\Theta_{\text{LM}}, \Phi, \mathcal{A}, \tilde{w})} \mathcal{L}_{\text{meta\_loss}}$;

Return: meta-trained PLM with learned $(\Theta_{\text{LM}}, \Phi, \mathcal{A}, \tilde{w})$;

of this form for our main experiments, with rank $r \leq 32$.

TAMS details. We apply TAMS to expand the FFN block, so the shared (in structure) sublayers capture the commonalities among tasks while new searched sublayers capture task-specific structure. Our search DAG contains two input nodes that project the inputs to a low-dimensional space, one output node that projects the intermediate representation back to the original dimension, and three intermediate nodes. Candidate operations for each edge are {linear, conv-3×1, conv-5×1, gated linear unit (GLU), zeroize, and skip connection}; see code for definitions. All the candidates operate on a reduced feature dimension to ensure the parameter efficiency of the search cell. Our controller $\mathcal{A}$ is a two-layer MLP. The first fully-connected layer has 128 output neurons, and the second layer has $E \times |\mathcal{O}|$ neurons (see Section 2.2 for notation). We apply ReLU after the first layer and softmax the final output.

Method	PPL	BLEU	C-score
Pretrain (multitask)*	36.75	0.64	-0.03
Pretrain+Finetune*	33.14	0.90	0.00
MAML+Finetune*	40.34	0.74	0.20
CMAML+Finetune*	36.30	0.89	0.18
*Pretrain+Persona**	*30.42*	*1.00*	*0.07*
Pretrain+MAML+Finetune	32.54	0.97	0.23
MLtD (TARP only)	32.15	0.99	0.25
MLtD	**28.14**	**1.20**	**0.30**

Table 1: Comparison of test perplexity (PPL; lower is better), BLEU (higher is better), and C-score (higher is better) for few-shot dialogue generation on Persona-Chat dataset. *: published results from Madotto et al. (2019); Song et al. (2020); the rest are ours.

3.2 Few-shot dialogue personalization

Persona-Chat (Zhang et al., 2018) is a dialogue generation benchmark with 1137/99/100 personas for training/validation/testing. We follow recent work (Madotto et al., 2019; Song et al., 2020) and regard learning a dialogue model for each persona as a few-shot meta-learning task. On average, each persona has 8.3 unique dialogues, 6-8 turns per dialogue, and 15 words per turn. Following these works, we use a standard Transformer model with pretrained GLoVe embeddings and separate the dialogues by their persona description into meta-training/-validation/-testing using Madotto et al. (2019)'s splits and code[3].

Baselines. The following are from previous works. **Pretrain** denotes a multitask dialogue model trained on labeled data from all meta-training tasks. **MAML** meta-trains the Transformer model from scratch (Madotto et al., 2019), and **CMAML** (Song et al., 2020) additionally applies a pruning algorithm to customize the model structures for different tasks. **+Finetune** corresponds to finetuning on each testing task. Finally, **Pretrain+Persona** is a partial oracle for reference only, where the persona description is available.

Results (Table 1). We include the same evaluation metrics from previous works, including perplexity, BLEU score, and C-score, where C-score is a domain-specific metric that uses a pretrained natural language inference model to evaluate whether the hypothesis matches the persona or not. Training MAML from scratch yields worse

[3] https://github.com/HLTCHKUST/PAML

Method	Dialog	Email	Movie	Debate	Social	Science	Avg.
Baseline (full finetuning)*	39.95	24.71	25.13	24.48	21.76	72.76	34.80
DAPT (Domain-Adaptive Pre-Training)*	41.22	26.50	24.25	26.71	22.95	71.88	35.59
TAPT (Task-Adaptive Pre-Training)*	40.15	25.30	25.27	24.59	22.81	73.08	35.20
SDPT (Supervised Domain Pre-Training)*	42.84	25.16	25.45	25.61	22.43	73.09	35.76
MLtD	**44.81**	25.30	**26.83**	**26.88**	**24.40**	**74.03**	**37.04**
(TARP only)	42.88	**26.92**	25.98	25.95	23.34	73.69	36.46
(TARP only, no meta-learning)	40.39	23.20	25.81	26.67	21.46	73.20	35.12
(TARP only, multitask pretraining instead)	41.82	25.41	26.17	25.70	22.54	73.50	35.85

Table 2: ROUGE F1s from multi-domain adaptation for abstractive summarization on AdaptSum (higher is better). All methods are initialized with pretrained BART and finetuned on the labeled task training set of each domain at the end. *: published results from Yu et al. (2021), using DAPT and TAPT methods from Gururangan et al. (2020); the rest are ours.

results than the Pretrain model. However, when MAML is initialized from the multitask model (**Pretrain+MAML+Finetune**), the result already outperforms previous work. Note that the *same labeled data* is used for both Pretrain and MAML, suggesting that meta-learning benefits from the more robust initialization that pretraining provides to improve task-specific few-shot adaptation (also see analysis in Section 4.1).

Moreover, we see further improvements by "meta-learning the difference" (MLtD). By using TARP for MAML's inner loop adaptation (**MLtD, TARP only**), we attain equivalent or better results and faster training time while only updating a small amount of task-specific parameters (Section 4.2). This indicates that our method helps mitigate overfitting to low-resource tasks. Finally, by incorporating TAMS (**MLtD**), we use the full framework and achieve the best performance, suggesting the task-adapted model structure gives better architectures for personas. In this regard, **CMAML** lags behind MLtD as well. We conjecture this is because it uses a pruning algorithm to "customize" the model with different weight masks, which may not generate enough model diversity for diverse tasks as the architectural inductive bias remains the same.

3.3 Low-resource abstractive summarization

AdaptSum (Yu et al., 2021) is a new multi-domain dataset used to evaluate domain adaptation schemes for abstractive summarization. It consists of six diverse target domains ranging from movie reviews to scientific abstracts. Each domain has a low-resource task corpus and a larger unlabeled text corpus as well (list and statistics in Table 4) that is used to evaluate domain- and task-adaptive pretraining (DAPT/TAPT; Gururangan et al., 2020). We use pretrained BART (Lewis et al., 2020) and finetune to each low-resource task corpus as in Yu et al. (2021), whose code[4] we extend.

Baselines. **DAPT** continues pretraining with BART's self-supervised objective using the unlabeled domain corpus. **TAPT** continues pretraining with the set of unlabeled documents found in the target summarization task. **SDPT** uses the XSum dataset in the News domain to further pretrain BART with a supervised training objective using document-summary pairs before finetuning.

Results (Table 2). We find that **MLtD**, even without architecture search (**TARP only**), outperforms DAPT, TAPT, and SDPT. These methods use in-domain/-task knowledge and the standard pretraining objective to help adaptation to the target task, while our method considers cross-domain knowledge via the meta-learning objective, sampling meta-training tasks from multiple domain corpora to train the model. Moreover, the use of meta-learning as preparation outperforms multitask pretraining (**TARP only, multitask pretraining instead**), signifying that mere exposure to the cross-domain data may not be enough and using a meta-learning objective to explicitly optimize for the lightweight adaptation is beneficial. Finally, we see that without meta-learning or multitasking (**TARP only, no meta-learning**) our performance is also better than the baseline. This demonstrates the effectiveness of the lightweight TARP adaptation, which matches the performance of full finetuning while only updating less than 5% of parameters.

[4] https://github.com/TysonYu/AdaptSum

Method	Dialog	Email	Movie	Debate	Social	Science	Avg.
Baseline (full finetuning)	31.95	31.57	42.25	34.38	33.02	28.82	33.67
Zero-shot (no finetuning)	37.26	38.45	49.46	41.38	37.13	34.20	39.65
DAPT†	35.15	**16.04**	43.12	33.83	27.15	18.96	29.04
MLtD	29.66	16.93	**35.38**	**30.61**	**19.78**	**17.06**	**24.90**
(TARP only)	**28.63**	18.67	39.73	32.70	26.93	20.39	27.84
(TARP only, no meta-learning)	31.66	31.59	41.78	33.18	32.78	28.20	33.19

Table 3: Test perplexities from multi-domain language modeling adaptation on AdaptSum (lower is better). All methods are initialized with pretrained GPT-2 medium and finetuned on the labeled domain set at the end. †: our re-implementation of Gururangan et al. (2020).

Domain	# of tokens			
	Text only	Task corpus		
		train	val	test
Dialog	44.96M	27K	74K	75K
Email	117.54M	37K	243K	237K
Movie review	11.36M	633K	1056K	6193K
Debate	122.99M	59K	188K	197K
Social media	153.30M	68K	229K	229K
Science	41.73M	63K	221K	314K

Table 4: Data sizes for AdaptSum (Yu et al., 2021) across the six domains, for both the text-only domain-related corpus and the low-resource task corpus.

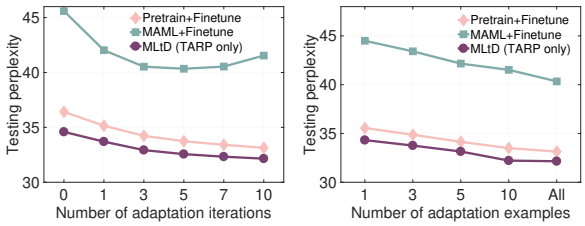

Figure 4: Perplexities on Persona-Chat testing tasks with MLtD (TARP only) versus Pretrain+Finetune and MAML+Finetune. **Left**: Influence of number of adaptation iterations. **Right**: Influence of the number of adaptation dialogues.

3.4 Multi-domain language modeling

Though the text corpora in AdaptSum were originally included to evaluate DAPT, we also use them to evaluate our methods on multi-domain language modeling. As this is a novel benchmark, to demonstrate fast adaptation we take $T_{\text{in}} = 1$.

Baselines. We start with pretrained GPT-2 medium (345M) (Radford et al., 2019) with input sequence length 512 using the Transformers library (Wolf et al., 2019). Finetuning is performed on the training documents of the task corpus, and we evaluate perplexity on the test documents of the task corpus. The only exception to finetuning is **Zero-shot** which evaluates the pretrained GPT-2 model directly. **DAPT** continues pretraining of GPT-2 with the language modeling objective on the unlabeled domain corpus before finetuning.

Results (Table 3). Our findings in summarization also hold for the unconditional causal language modeling task. Namely, we see equal or better performance of TARP vs. full finetuning and that meta-learning plays a significant role in the task adaptation quality. In contrast to summarization with BART (Section 3.3) but similar to Persona-Chat with Transformer (Section 3.2), we see that TAMS leads to noticeable improvements. We explain why this may be the case and present the TAMS-learnt sublayer modules in Section 4.4.

4 Analysis

4.1 Pretraining improves meta-learning

We analyze the performance of MLtD on Persona-Chat at meta-testing time (i.e., finetuning then testing on unseen personas) with respect to the number of inner loop steps and training dialogues. In Figure 4 (left), we see that original MAML (no pretraining) overfits, while finetuning the multitask-pretrained model keeps improving. Moreover, MLtD atop the multitask-pretrained model followed by finetuning continues to improve test perplexity. In Figure 4 (right), we fix the finetuning steps and vary the number of training dialogues used in finetuning. Using more dialogues improves perplexity for all three methods, with MLtD still leading over full MAML and direct finetuning after pretraining. The takeaway from these results is that applying MAML on a pretrained model prevents overfitting and promotes better generalizability from meta-learning.

Method	Params. per task	CoLA Matt. corr	MRPC Acc.	STS-B Pear. corr	RTE Acc.	SST-2 Acc.	MNLI Acc.	QNLI Acc.	QQP Acc.	Avg.
Finetuning (full)*	100%	63.6	90.2	**91.2**	78.7	**94.8**	**87.6**	92.8	**91.9**	86.4
Masking*	3%	60.3	88.5	-	69.2	94.5	-	92.4	-	-
AdapterFusion*	1%	-	89.7	-	78.8	93.7	86.2	-	90.3	-
MAM Adapter*	0.5%	59.2	88.5	90.6	74.3	94.2	87.4	92.6	90.2	84.6
BitFit*	0.1%	61.8	**92.0**	90.8	77.8	93.7	84.8	91.3	84.5	84.6
MAM Adapter†	1%	59.7	90.2	90.6	77.3	94.6	**87.6**	92.9	90.9	85.5
LoRA (orig.)†,[5]	1%	63.9	89.7	90.7	76.2	94.5	87.5	92.7	90.8	85.8
Dynamic TARP	1%	**65.3**$_{\pm.8}$	90.9$_{\pm.4}$	91.0$_{\pm.2}$	**80.9**$_{\pm.7}$	**94.8**$_{\pm.2}$	**87.6**$_{\pm.2}$	**93.0**$_{\pm.2}$	91.3$_{\pm.1}$	**86.8**

Table 5: Comparison with other adaptation methods on the GLUE benchmark. We report single-task results (including our dynamic TARP) of adapting RoBERTa-base to each task only. *: published results from Liu et al. (2019b); Zhao et al. (2020); Zaken et al. (2021); Pfeiffer et al. (2021); He et al. (2022); †: recreated using He et al. (2022)'s implementation. For dynamic TARP, we provide the 95% confidence interval over five runs.

Method	Params. per task	E2E BLEU	E2E NIST	E2E METEOR	E2E ROUGE-L	E2E CIDEr	DART BLEU	WebNLG BLEU
Finetuning (full)*	100%	68.2	8.62	46.2	71.0	2.47	46.0	47.6
FT-Top2*	7.1%	68.1	8.59	46.0	70.8	2.41	38.1	33.5
BitFit*	0.1%	67.2	8.63	45.1	69.3	2.32	43.3	50.5
Adapter*	3.2%	68.9	8.71	46.1	71.3	2.47	45.4	54.0
Prefix*	1.0%	69.7	8.81	46.1	71.4	2.49	45.7$_{\pm.2}$	54.4$_{\pm.1}$
LoRA*	1.0%	**70.4**$_{\pm.1}$	**8.85**$_{\pm.02}$	46.8$_{\pm.2}$	71.8$_{\pm.1}$	**2.53**$_{\pm.02}$	47.1$_{\pm.2}$	**55.3**$_{\pm.2}$
Bilinear TARP	2.4%	68.8	8.75	46.1	70.8	2.43	46.7	54.0
Kronecker TARP	2.4%	68.2	8.73	45.2	69.4	2.36	45.6	53.1
Dynamic TARP	1.0%	69.7$_{\pm.1}$	8.78$_{\pm.02}$	**46.9**$_{\pm.2}$	**72.1**$_{\pm.1}$	2.51$_{\pm.01}$	**47.9**$_{\pm.2}$	**55.3**$_{\pm.1}$
w/ matrix mult.	1.0%	68.3	8.64	46.4	71.1	2.47	46.5	53.2

Table 6: Comparison with other adaptation methods for natural language generation (E2E, DART, WebNLG) on GPT-2 medium. *: published results from Houlsby et al. (2019); Zaken et al. (2021); Li and Liang (2021); Hu et al. (2022); the rest are ours. For dynamic TARP, we provide the 95% confidence interval over five runs.

4.2 Dynamic TARP versus alternatives

We benchmark our dynamic low-rank reparameterization on a variety of NLP models and tasks. To show that dynamic TARP individually improves on full finetuning and other parameter-efficient adaptation methods, we report single-task results here. For classification, we use pretrained RoBERTa (Liu et al., 2019b) on the GLUE benchmark tasks (Wang et al., 2019), which were evaluated on by many recent parameter efficient adaptation methods (Houlsby et al., 2019; Zhao et al., 2020; Zaken et al., 2021; Pfeiffer et al., 2021; He et al., 2022). For generative tasks, we use pretrained GPT-2 medium on natural language generation datasets: we specifically evaluate on E2E (Novikova et al., 2017), which was used for adapters (Lin et al., 2020); WebNLG (Gardent et al., 2017); and DART (Nan et al., 2021), which was used by LoRA (Hu et al., 2022). Further dataset and experimental setup details are in Appendix B. In particular, we chose rank r to give similar parameter counts to other approaches; $r = 4$ in Table 6, $r = 8$ in Table 5.

For classification tasks, we compare with **finetuning** all layers; weight **Masking** (Zhao et al., 2020); **BitFit** (Zaken et al., 2021), which only finetunes the biases; **AdapterFusion** (Pfeiffer et al., 2021), which composes learned adapters (Houlsby et al., 2019); as well as He et al. (2022), which proposed a unified framework connecting several state-of-the-art adaptation methods like LoRA[5] (Hu et al., 2022) and Adapter (Houlsby et al., 2019), and derived an improved method (**MAM Adapter**). Our dynamic TARP can only partly be viewed in this unified framework as we explore a novel design dimension, i.e., making the modification to the base model dynamic w.r.t. input tokens. For fair comparisons, we follow past works (Liu et al.,

[5](orig.) denotes Hu et al. (2022)'s v1 preprint, which was the one available during our's and He et al. (2022)'s work and thus used in their implementation. Published LoRA adds a per-dataset tunable scaling α that we do not explore for TARP.

2019b; Zhao et al., 2020; He et al., 2022) and set the maximum finetuning epochs to 10 on each task. In Table 5, dynamic TARP introduces and trains only 1% versus the number of original parameters, while achieving comparable results to full finetuning and outperforming the previous best results from MAM adapters.

For generative tasks, we ablate the design of TARP and we compare with available numbers, including **finetuning** all layers; **FT-Top2**, which only finetunes the last two layers of the model; **Adapter** (Houlsby et al., 2019), which only finetunes the adapter layers inserted after each feed-forward and self-attention sublayer; **Prefix-tuning** (Li and Liang, 2021) and **LoRA** (Hu et al., 2022). As shown in Table 6, TARP methods match or outperform other parameter-efficient methods, while learning task-specific parameters that are $<3\%$ of the number of base parameters and keep the base model unchanged. Among the three TARP variants, we find that **Dynamic** $>$ **Bilinear** $>$ **Kronecker** in terms of performance across generative metrics. This suggests that the optimal adaptation to the underlying model weights may vary per token, which dynamic low-rank accounts for. Moreover, dynamic TARP performs better than an alternative where the $O(n^2)$ Hadamard product in Eq.(1) is replaced by $O(n^3)$ matrix multiplication (**w/ matrix mult.**).

4.3 Dynamic TARP outperforms finetuning

Tables 5 and 6 also show that dynamic low-rank reparameterization outperforms finetuning on corresponding evaluating metrics, while being faster as it only adapts a small set of weights. The training time further improves through utilizing the training data more efficiently. In Figure 5 (left) we compare perplexities of our method against finetuning on subsets of WikiText-2 and see that finetuning increasingly underperforms as the number of examples decrease. To explain this behavior, in Figure 5 (right) we fix the number of training examples to 100 and ablate the rank. Our method performs best with a very small rank value, suggesting that the difference between the pretrained and finetuned weight matrices lies in a lower-dimensional subspace. This complements Aghajanyan et al. (2021b)'s observation that direct adaptation in lower-dimensional spaces can be equally as effective as in the original space. Moreover, we find that the larger the model (GPT-2 medium vs. the GPT-2 small), the lower the rank value required for the best adaptation performance.

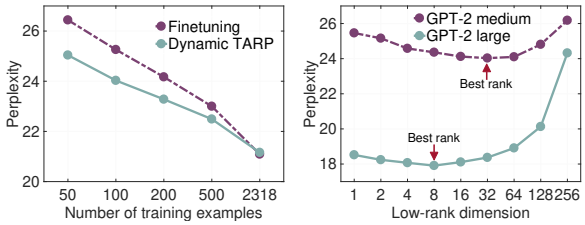

Figure 5: Testing perplexities on WikiText-2 with full finetuning and/or low-rank adaptation with dynamic TARP. **Left**: Low-rank adaptation is extremely helpful on low-resource tasks. **Right**: Holding the number of training examples fixed, the model adaptation space is optimized by fewer dimensions.

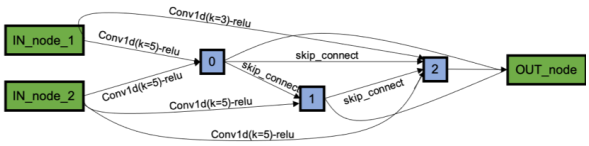

Figure 6: Dominant structure of the TAMS-learned sublayers for AdaptSum language modeling.

4.4 TAMS discovers better architectures

Recent studies have shown that simple modifications to the transformer architecture, such as reorganizing the MHSA and FFN modules (Zhao et al., 2021) or adding 1D convolutions to self-attention (So et al., 2021), improve the task performance. Similarly, from our results in Table 3, adapting the model structure through sub-layer modifications in our meta-learning framework further reduces the testing perplexity compared to MLtD with fixed model structure. Applying task-aware architecture search (TAMS) on the FFN module incurs less than 5% additional model parameters compared to the original GPT-2 model, but reduces the perplexity by 3 points on average.

A limitation we observe is that the TAMS method tends to produce a dominant architecture (cf. Figure 6) as opposed to one different architecture for each task. We conjecture this may be because our initial task representation strategy has low variance due to averaging across the entire task training data. This may explain why TAMS did not uniformly improve MLtD in all settings. Nevertheless, the perplexity reduction implies that there is still room to optimize the architecture of current LMs without significantly increasing total model size. Thus, we believe task-aware architec-

Method	Prep. (hrs.)	Finetuning (mins.)
Baseline (direct finetuning)	–	26
SDPT	64	16
DAPT	208	23
TAPT	8	18
MLtD	39	9
(TARP only)	22	7
(TARP only, no meta-learning)	–	20

Table 7: Wall-clock time comparison on AdaptSum during preparation on the meta-training data (Prep.) and during meta-testing (Finetuning) to convergence (early stopping), summed over all domains. Times were measured on one GPU.

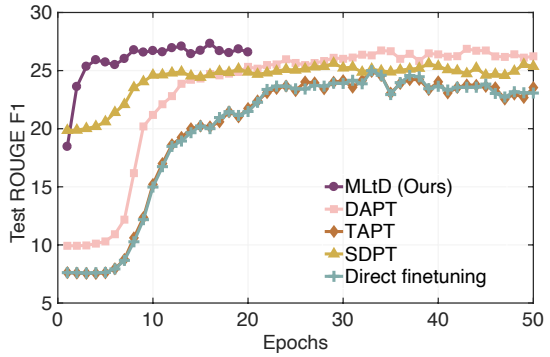

Figure 7: Convergence trajectories when finetuning BART upon applying each method on AdaptSum, using the Debate domain as an example.

ture search is a promising direction to continue to invest in the future.

4.5 Training efficiency of MLtD

We study training efficiency by comparing the training and finetuning wall-clock time for multi-domain abstractive summarization on AdaptSum. The results are shown in Table 7.

We have the following observations: (1) since meta-learning explicitly optimizes the model for fast adaptation. Compared with previous methods, MLtD takes fewer epochs to reach convergence (e.g., Figure 7) and takes the least time to adapt the model to each the task; (2) since our lightweight adaptation method (TARP) only updates a small set of task-specific weights, our model variant (TARP only, no meta-learning) reduces the adaptation time by 20% over direct BART finetuning.

On the other hand, the proposed TARP and TAMS components introduce some inference overhead. Due to limitations of current DL libraries in implementing parallel computation branches, the dynamic low-rank decomposition and the task-aware architecture generation increases the inference time by 10% and 6%, respectively, measured with a batch size of 4 and a sequence length of 1024 on one GPU.

5 Conclusion

We have shown that explicit meta-learning is a useful preparation step on top of PLMs to improve later finetuning. Specifically, our MLtD framework incorporating dynamic task-adaptive reparameterization (TARP) and task-adaptive model search (TAMS) enable data- and parameter-efficient adaptation to a family of low-resource tasks. Future avenues include applying our method in other modalities like vision and speech, as well as exploring better model formulations for TARP and TAMS.

Acknowledgements

We thank our colleagues on the Speech Science team at Amazon AWS AI for supporting this research. We also thank our TACL action editor Shay Cohen and the reviewers for their helpful feedback.

References

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021a. Muppet: Massive multi-task representations with pre-finetuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5799–5811, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021b. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. 1990. *Learning a synaptic learning rule*. Citeseer.

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 *NeurIPS*.

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. In *NIPS*, pages 3079–3087.

Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. 2019. Investigating meta-learning algorithms for low-resource natural language understanding tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1192–1197, Hong Kong, China. Association for Computational Linguistics.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics.

Demi Guo, Alexander Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4884–4896, Online. Association for Computational Linguistics.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level Adversarial ReProgramming. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4921–4933, Online. Association for Computational Linguistics.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

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*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics:*

EMNLP 2020, pages 441–459, Online. Association for Computational Linguistics.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. DARTS: Differentiable architecture search. In *ICLR (Poster)*. OpenReview.net.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *CoRR*, abs/2103.10385.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Andrea Madotto, Zhaojiang Lin, Chien-Sheng Wu, and Pascale Fung. 2019. Personalizing dialogue agents via meta-learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5454–5459, Florence, Italy. Association for Computational Linguistics.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2021. DART: Open-domain structured data record to text generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 432–447, Online. Association for Computational Linguistics.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, Saarbrücken, Germany. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 4092–4101. PMLR.

Ofir Press, Noah A. Smith, and Omer Levy. 2020. Improving transformer models by reordering their sublayers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2996–3005, Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

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.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2018. Efficient parametrization of multi-domain deep neural networks. In *CVPR*, pages 8119–8127. Computer Vision Foundation / IEEE Computer Society.

Jürgen Schmidhuber. 1987. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Ph.D. thesis, Technische Universität München.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

David So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. 2021. Primer: Searching for efficient transformers for language modeling. *Advances in Neural Information Processing Systems*, 34:6010–6022.

Yiping Song, Zequn Liu, Wei Bi, Rui Yan, and Ming Zhang. 2020. Learning to customize model structures for few-shot dialogue generation tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5832–5841, Online. Association for Computational Linguistics.

Asa Cooper Stickland and Iain Murray. 2019. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 5986–5995. PMLR.

Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. 2019. Meta-transfer learning for few-shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach,*

CA, USA, June 16-20, 2019, pages 403–412. Computer Vision Foundation / IEEE.

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 *ICLR (Poster)*. OpenReview.net.

Lilian Weng. 2018. Meta-learning: Learning to learn fast. https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Tiezheng Yu, Zihan Liu, and Pascale Fung. 2021. AdaptSum: Towards low-resource domain adaptation for abstractive summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5892–5904, Online. Association for Computational Linguistics.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *CoRR*, abs/2106.10199.

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

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2204–2213, Melbourne, Australia. Association for Computational Linguistics.

Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. 2020. Masking as an efficient alternative to finetuning for pretrained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2226–2241, Online. Association for Computational Linguistics.

Yuekai Zhao, Li Dong, Yelong Shen, Zhihua Zhang, Furu Wei, and Weizhu Chen. 2021. Memory-efficient differentiable transformer architecture search. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4254–4264, Online. Association for Computational Linguistics.

A Further details for main experiments

A.1 Hyperparameters

Most of the experimental setups, e.g., model type, max sequence, optimizer, batch size, beam search size, are taken from previous methods for fair comparison. We tuned the inner-loop and outer-loop learning rates in meta-training on the meta-validation set, and adjust the learning rate schedule accordingly. We chose the rank values r in our dynamic low-rank reparameterization to give similar parameter counts to other parameter-efficient methods. We adapted the search space in our task-aware model structure from DARTS. η_{in} denotes inner-loop and finetuning learning rate, η_{out} denotes outer-loop learning rate, B_{in} denotes inner-loop and finetuning batch size, B_{out} denotes meta-batch size, bsz denotes decoding beam size, and T_{in} denotes inner loop steps.

Few-shot dialogue personalization. We take $r = 4$, $B_{\text{out}} = 16$ (as in previous works), bsz $= 5$, $T_{\text{in}} = 10$. For meta-training we use SGD ($\eta_{\text{in}} = 0.01$) in the inner loop and Adam for the outer loop ($\eta_{\text{in}} = 0.0003$).

Low-resource abstractive summarization. We take $r = 16$, $B_{\text{in}} = 40$ (via gradient accumulation), $T_{\text{in}} = 20$, bsz $= 4$. We truncated the input documents into 1024 tokens due to the limit of max input length of BART model. We used Adam with momentum ($\beta_1 = 0.9, \beta_2 = 0.998$) and the Noam schedule (linear warmup of 1000 steps, then inverse square-root decay). Since the low-resource training set of science domain only has 100 samples, we used 3 times more training epochs than other domains.

Multi-domain language modeling. We take $r = 32$, $B_{\text{in}} = 4$, and $T_{\text{in}} = 1$. We used Adam with $\eta_{\text{in}} = 5 \times 10^{-4}$, $\eta_{\text{out}} = 5 \times 10^{-5}$. In meta-testing we linear decay η_{in}.

A.2 Training costs

Table 8 provides information about the amount of training that MLtD has taken for the main experiments. The reported training times are for the single run of each experiment. For hyperparameter tuning, we search the inner-loop learning rate and outer-loop learning rate over five runs respectively.

#GPUs	GPU type	Training time	Meta-iterations	Cluster	Costs
Low-resource abstractive summarization (Table 1)					
1	32GB V100	39hrs	100	AWS p3.2xlarge	$120
Few-shot dialogue personalization (Table 2)					
1	32GB V100	1.5hrs	100	AWS p3.2xlarge	$5
Multi-domain language modeling (Table 4)					
1	32GB V100	22hrs	100	AWS p3.2xlarge	$68

Table 8: Details on the training cost of MLtD per run. Cost ($) estimated from on-demand instance prices.

B Further details for TARP experiments

Datasets. **E2E** (Novikova et al., 2017) is commonly used for data-to-text evaluation of NLG systems. It consists of approximately 50K examples in total from the restaurant domain. Each input consists of a sequence of slot-value pairs and can have multiple references. The average output length is 22.9. We use the official evaluation script, which reports BLEU, NIST, METEOR, ROUGE-L, and CIDEr. **WebNLG** (Gardent et al., 2017) is a multi-domain dataset for data-to-text evaluation. It contains 22K examples in total from 14 distinct domains, and the average output length is 22.5. Nine domains are used for training, and the remaining five domains are used for testing. Each input is represented by a sequence of SUBJECT | PROPERTY | OBJECT triples. The evaluation metric is BLEU. **DART** (Nan et al., 2021) is an open-domain data-to-text dataset. The inputs are structured as sequences of ENTITY | RELATION | ENTITY triples. It contains 82K examples in total and the average output length is 21.6. The evaluation metric is BLEU. **GLUE** We report Matthew's correlation for CoLA, Pearson correlation for STSB, and accuracy for the other tasks in Table 5. The dev set performance is presented by following Zhao et al. (2020) and He et al. (2022).

Setup. For the natural language generation tasks, we build upon Hu et al. (2022)'s code.[6] We used the GPT2-medium as the underlying LM. In training, we used the AdamW optimizer with weight decay 0.01. The batch size is set to be 8 and we trained for 5 epochs in total. We used linear decay learning rate scheduler with the first 500 iterations for warmup. The initial learning rate is set to be 0.0002. In decoding, we used beam search with beam size 10.

For GLUE tasks, we built upon He et al. (2022)'s code[7]. Our experiments were performed on RoBERTa$_{\text{base}}$ model. We limited maximum length of a sentence (pair) to be 512 after wordpiece tokenization. We used the Adam optimizer with batch size 32, and trained for 10 epochs on each task. The learning rate is a hyperparameter to tune for different tasks over $\{1, 2, 3, 4, 5\} \times 10^{-4}$, with a linear warmup for the first 6% of steps followed by a linear decay to zero.

[6] https://github.com/microsoft/LoRA/tree/snapshot-9-15-2021; they have greatly refactored their code since our experiments.

[7] https://github.com/jxhe/unify-parameter-efficient-tuning