HART: Efficient Adaptation via Regularized Autoregressive Parameter Generation

Chen Liang*, **Nikos Karampatziakis***⋆*, **Tuo Zhao***, **Weizhu Chen***⋆*
*Georgia Tech, ⋆Microsoft Azure AI
{cliang73, tourzhao}@gatech.edu, {nikosk, wzchen}@microsoft.com

Abstract

Fine-tuning is an effective approach for adapting a pre-trained language model to downstream tasks, but it incurs a high computational cost. To achieve an extremely efficient task adaptation, [17] have proposed to use an auxiliary hypernetwork to generate task-specific weights without any backpropagation. A hypernetwork can generate weights for parameter-efficient fine-tuning (PEFT) modules, such as prefixes [12] and LoRAs [7], for any unseen task based on a few task-specific demonstration examples, at the cost of a single forward pass. However, hypernetwork training is challenging. Firstly, it is sample inefficient due to the under-exploitation of the dependencies between PEFT weights across layers. Secondly, it exhibits training instability due to the high diversity of few-shot demonstration inputs. To address these limitations, we propose a novel hypernetwork training approach, named HART. It exploits layerwise dependencies by autoregressively generating weights for individual layers, and stabilizes the training by regularizing the consistency between weights generated based on different demonstrations. We train the hypernetwork on a diverse collection of tasks [23, 19] and evaluate its performance on unseen tasks. HART notably outperforms [17] on both T5-Large and T5-XL models.

1 Introduction

Pre-trained large language models (LLMs) have demonstrated remarkable capabilities on various tasks [28, 16, 1, 21, 24]. While fine-tuning is an effective approach for adapting pre-trained models to specific tasks, it incurs significant computational costs, which further escalates with the increasing model size. In contrast, in-context learning (ICL) can quickly generalize a pre-trained model to unseen tasks by conditioning the model's inference on a few demonstration examples [22, 25, 29, 26, 15]. However, without fine-tuning, the model weights lack the adaptability to each task.

To achieve an extremely fast adaptation to unseen tasks, [17] have proposed a hypernetwork approach. A hypernetwork is a text-to-weight generator, which learns a universal mapping across a large collection of tasks from few-shot demonstration examples to parameter-efficient fine-tuning (PEFT) module weights of a pre-trained model, such as the weights of prefixes [12], LoRAs [7] and adaptors [6]. Consequently, when provided with several few-shot demonstration examples from an unseen task, the hypernetwork can generate the PEFT parameters for that task. Compared to iteratively fine-tuning task-specific parameters from scratch, generating these parameters requires only a single forward pass, thereby facilitating an extremely fast adaptation to various unseen tasks.

The hypernetwork adopts an encoder-decoder Transformer architecture, in which the decoder generates parameters conditioned on the encoded demonstration examples. To train such a hypernetwork, the decoder generates a hidden state at each training iteration, which is then projected, by different MLPs, into the weight spaces of different layers in the pre-trained model (referred to as "the main

37th R0-FoMo: Workshop on Robustness of Few-shot and Zero-shot Learning in Foundation Models at NeurIPS 2023(NeurIPS 2023).

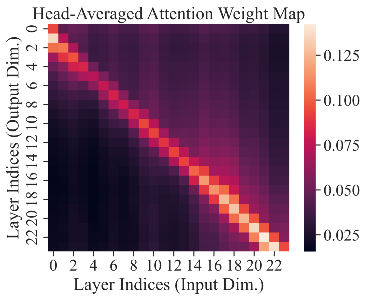

Figure 1: The bi-directional self-attention map averaged by attention heads in the last layer of the hypernetwork decoder. The input key/query is a sequence of hidden states, each responsible for learning PEFT parameters at a corresponding layer. Our experiment is conducted using T5-Large model [18] on P3 [19].

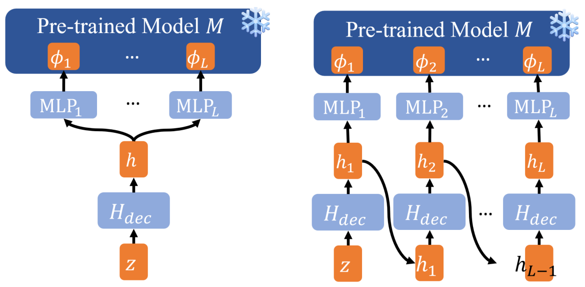

Figure 2: An illustrative comparison between non-autoregressive (left) and autoregressive (right) parameter generation schemes. H_{dec}: the hypernetwork decoder; z: a learnable input state; h_l: the hidden state at the l-th decoding step; ϕ_l: the PEFT parameters for the l-th layer. L: the number of layer in the pre-trained model.

model"). Then the hypernetwork is optimized based on the main model's prediction loss on a training example.

However, such a training scheme faces several limitations. One limitation is its low sample efficiency due to its heavy reliance on MLPs. Firstly, each MLP is assigned to learn parameters for a specific layer, based on a state that is shared among all layers. This implies that the entire responsibility of modeling the specificities of different layers fall upon the MLPs. However, the decoder, which has a greater capacity to capture layer-specific information in high-dimensional spaces, is not utilized to its fullest potential. Secondly, each MLP operates independently, rendering this design incapable of leveraging the potential structured dependencies between weights at different layers, which could serve as useful inductive biases during weight generation. In multi-layer networks, it is plausible to expect structured dependencies between weights across layers due to the structured dependencies between each layer's inputs: each layer only takes inputs from its preceding layer. To validate this hypothesis, we train the decoder to generate a sequence of hidden states using bi-directional attention, with each state responsible for learning the weights for an individual layer. Figure 1 presents the attention map across all states, revealing that each layer primarily attends to its two preceding layers, emphasizing its immediate predecessor. This observation confirms the existence of strong dependencies between weights at adjacent layers, an unexploited inductive bias for weight generation. Both the underutilization of the decoder's capabilities and underexploitation of the layerwise dependency impair the sample efficiency. As a result, the representation power of the generated parameters is compromised, leading to the underfitting of the main model.

To improve the sample efficiency, we propose a novel autoregressive parameter generation scheme as illustrated in Figure 2. Specifically, the decoder generates a sequence of hidden states autoregressively, with each state responsible for learning PEFT parameters of a corresponding layer. This allows each state to be learned through a layer-specific transformation modeled by the decoder. Furthermore, since each state is generated conditioned on its preceding state, the dependencies between adjacent layers are explicitly enforced, thereby introducing a proper inductive bias into the generation process. By capturing the layer-specificity and exploiting the layer-dependencies, the autoregressive scheme improves the expressiveness of the generated parameters.

Another challenge in hypernetwork training is its instability, which arises from the high diversity of inputs. As each input is a randomly selected sequence of demonstrations, considerable variation can occur across iterations, leading to significant variance in generated parameters. This variance destabilizes the main model's prediction loss and the hypernetwork training process (Figure 4). To resolve this issue, we propose a local consistency regularization method. This method discourages significant deviations in parameters generated between consecutive iterations, thereby stabilizing the gradient computation and the hypernetwork training process.

Finally, we propose HART, a novel **H**ypernetwork training scheme that incorporates both **A**uto**R**egressive parameter generation and local consistency regulariza**T**ion. We train the hypernetwork on diverse tasks and evaluate its generalizability on unseen tasks. Specifically, HART outperforms HyperTuning [17] by 1.6 points on the Super-NaturalInstructions task collection (S-NI, [23]) and by 3.6 points on the P3 task collection [19] using the T5-XL model [18].

2 Preliminaries

2.1 In-Context Learning

In-Context learning (ICL) considers inferencing a pre-trained model on an unseen task based on a few demonstration examples [13, 22, 25, 29]. Such an approach significantly outperforms zero-shot inference by leveraging the distribution of few-shot demonstrations as a task-specific prior for model prediction [15, 26]. Specifically, the model takes a concatenated input of the few-shot demonstrations and a query to generate the response. Each input is written in a natural language template, such as a task-specific prompt or instruction, allowing the model to better follow the task intention.

Multi-task In-Context Fine-tuning. Due to the lack of weight adaptation, ICL significantly underperforms fine-tuning. To mitigate this gap, researchers have proposed to in-context fine-tune the pre-trained model on diverse training tasks before ICL [14, 19, 23, 2]. Formally, we consider a pre-trained model denoted as $M(\cdot;\theta)$ parameterized by θ, and a set of training tasks denoted as $\mathcal{T}$. At each training iteration, a task τ is sampled in proportion to the task size from $\mathcal{T}$. Then, a query denoted as x, its corresponding response denoted as y, and $K > 0$ demonstration input-output pairs denoted as $\{(x_k, y_k)\}_{k=1}^{K}$ are sampled from the task distribution $\mathcal{D}^{\tau}$. The model is optimized based on the following objective:

$$\min_{\theta} \mathbb{E}_{\tau\in\mathcal{T},\{(x_k,y_k)\}_{k=1}^{K},(x,y)\sim\mathcal{D}^{\tau}} \ell(M([d:x];\theta),y), \tag{1}$$

where ℓ denotes the task loss and $d := [x_1 : y_1 : ... : x_K : y_K]$ denotes a concatenation of K-shot demonstrations. In prompted fine-tuning, x, y and d will be mapped to the prompted forms [19, 14]. In instruction fine-tuning, d will be appended to a task definition [23, 2].

2.2 Hypernetwork

Parameter Efficient Fine-tuning (PEFT) introduces a minimal set of parameters, termed "PEFT parameters", to each layer of the pre-trained model and fine-tunes only these parameters while keeping the pre-trained model frozen [12, 7, 6]. Although PEFT attains adaptability comparable to standard fine-tuning [4], it incurs significant costs from full-model gradient backpropagation.

Hypernetworks. A hypernetwork is an auxiliary model trained to generate weights for a main model [20, 5]. [17] have proposed to use a hypernetwork to generate PEFT parameters for a pre-trained model. With a few demonstration examples from a new task, the hypernetwork can generate task-specific parameters, enabling the main model to conduct zero-shot inference on task-related queries. In contrast to PEFT, this method requires only one forward pass for task adaptation, thereby cutting the backpropagation costs. When weighted against ICL, this method improves task adaptability through task-specific weight updates.

The hypernetwork is trained in the multi-task in-context fine-tuning setting. Specifically, we denote a hypernetwork parameterized by ξ as $H(\cdot;\xi)$. At each training iteration, the hypernetwork takes the K-shot demonstrations d as input and generates the PEFT parameters. The main model takes in the query x and generates the response y using the PEFT parameters. The hypernetwork is then optimized based on the main model's prediction loss:

$$\min_{\xi} \mathbb{E}_{\tau\in\mathcal{T},\{(x_k,y_k)\}_{k=1}^{K},(x,y)\sim\mathcal{D}^{\tau}} \ell(M(x;\theta,H(d;\xi)),y). \tag{2}$$

Parameter Generation in Hypernetworks. The hypernetwork employs a Transformer encoder to encode the demonstrations and a Transformer decoder to conditionally generate parameters based on the encoded demonstrations. Specifically, we denote the decoder as $H_{\text{dec}}(\cdot;\xi_{\text{dec}})$. At the t-th training iteration, the decoder takes in a learnable token $z^{(t)} \in \mathbb{R}^{1\times d_{\text{m}}}$, where d_{m} is the hypernetwork's hidden dimension. It then generates a hidden state $h^{(t)} \in \mathbb{R}^{1\times d_{\text{m}}}$ through one decoding step:

$$h^{(t)} = H_{\text{dec}}(z^{(t)};\xi_{\text{dec}}^{(t)},e^{(t)}), \tag{3}$$

where $e^{(t)}$ denotes the encoded demonstrations[1]. The generated hidden state is then projected into L sets of PEFT parameters through L learnable MLP layers:

$$\phi_l^{(t)} = \texttt{MLP}_l(h^{(t)}) \quad \forall l \in [L], \tag{4}$$

where $\phi_l^{(t)} \in \mathbb{R}^{d_{\mathrm{P}}}$ denotes the PEFT parameters generated for the l-th layer of the main model, which consists L layers.

3 Method

We introduce HART, which exploits layerwise dependencies through autoregressive parameter generation and stabilizes training through local consistency regularization.

3.1 Autoregressive Parameter Generation

We propose to autoregressively generate the PEFT parameters for different layers in the main model. At the t-th training iteration, the hypernetwork decoder generates L hidden states through L decoding steps. Each state is generated conditioned on its preceding state, and is responsible for learning the weight space of the corresponding layer. At the first decoding step, the decoder takes in a learnable token, $z^{(t)}$, and generates the first hidden state, $h_1^{(t)}$, following Eq. 3:

$$h_1^{(t)} = H_{\mathrm{dec}}(z^{(t)}; \xi_{\mathrm{dec}}^{(t)}).$$

At the second decoding step, the decoder takes in $h_1^{(t)}$ and generates the second hidden state, $h_2^{(t)}$. This procedure is repeated for L decoding steps:

$$h_l^{(t)} = H_{\mathrm{dec}}(h_{l-1}^{(t)}; \xi_{\mathrm{dec}}^{(t)}) \qquad \text{for } l = 2, ..., L.$$

Then we learn L MLP layers to project these L hidden states into L sets of PEFT parameters, respectively, following Eq. 4:

$$\phi_l^{(t)} = \texttt{MLP}_l(h_l^{(t)}) \quad \forall l \in [L],$$

where $\phi_l^{(t)}$ denotes the set of PEFT parameters generated for the l-th layer of the main model. We can then compute the main model's prediction loss as:

$$\mathcal{L}_{\mathrm{pred}}(\xi^{(t)}) = \ell(M(x^{(t)}; \theta, \{\phi_l^{(t)}\}_{l=1}^{L}), y^{(t)}). \tag{5}$$

where ℓ is defined in Eq 2.

The autoregressive generation scheme allows us to leverage a powerful decoder, instead of relying on MLPs, to model layer-specific transformations. Furthermore, this scheme exploits the strong dependencies between weights at adjacent layers as observed in Figure 1, thereby introducing a proper inductive bias into the generation process.

By exploiting the decoder's capabilities and the layerwise dependencies, the autoregressive generation scheme achieves a greater sample efficiency than the original scheme. As a result, the generated parameters are more expressive, leading to a better-fitted main model.

3.2 Local Consistency Regularization

We further propose to encourage the PEFT parameters generated at consecutive training iterations to not deviate significantly from each other. At the t-th iteration, we compute the local consistency loss as:

$$\mathcal{L}_{\mathrm{cst}}(\xi^{(t)}) = \texttt{MSE}([h_1^{(t)} : \ldots : h_L^{(t)}], [h_1^{(t-1)} : \ldots : h_L^{(t-1)}]),$$

where $\ell_{\mathrm{cst}}(\xi^{(1)}) = 0$, $\texttt{MSE}(\cdot, \cdot)$ denotes the mean squared error, and $[h_1^{(\cdot)} : \ldots : h_L^{(\cdot)}] \in \mathbb{R}^{L \times d_m}$ is the concatenation of the sequence of generated hidden states. Finally, we optimize the hypernetwork based on the sum of the prediction loss and the consistency loss using an SGD-type algorithm:

$$\xi^{(t+1)} \leftarrow \xi^{(t)} - \nabla_{\xi^{(t)}}(\mathcal{L}_{\mathrm{pred}}(\xi^{(t)}) + \alpha \mathcal{L}_{\mathrm{cst}}(\xi^{(t)})),$$

[1]We omit $e^{(t)}$ for the rest of the paper to simplify the notations.

where $\mathcal{L}_{\text{pred}}(\xi^{(t)})$ is defined in Eq. 5 and $\alpha > 0$ is a hyperparameter.

By encouraging consistency between states generated at consecutive iterations, we mitigate drastic changes in the generated parameters arising from highly diverse inputs. This improves the smoothness of the main model, stabilizing both gradient computations and hypernetwork training.

4 Experiments

We evaluate HART on several commonly used large-scale multi-task NLP benchmarks. All dataset and implementation details can be found in Appendix 6.1. Additional ablation studies validating our proposed strategies can be found in Appendix 6.2.

4.1 Super-NaturalInstructions (S-NI)

Dataset. Super-NaturalInstructions (S-NI, [23]) consists of 1616 tasks spanning 76 diverse categories, including translation, question answering and sentiment analysis, etc. Each task is associated with an expert-written task definition, a set of input-output pairs as positive and negative demonstrations, and a set of input-output pairs as training queries and responses. For each task, we construct the input to the hypernetwork as the concatenation of the task definition and two fixed positive demonstrations, denoted as "Def+2Pos". This input format has been observed to outperform other formats [23]. Following [17], we select the English tasks for training and evaluation. We evaluate the generation performance of the main model on the held-out test set using the ROUGE-L metric.

Model initialization. All our experiments are conducted based on the LM-adapted T5 models (version 1.1, [11]). T5 models are encoder-decoder Transformer-based models pre-trained using web-scale text-to-text corpus [18][2]. We consider two model scales: T5-Large (770M) and T5-XL (3B). For experiments on the T5-Large/XL, we initialize both the hypernetwork and the main model with the T5-Large/XL unless otherwise stated. We only keep the first 8 decoder layers (out of 24) in the hypernetwork for training efficiency. We randomly initialize the MLP layers.

Training. We freeze the main model and multi-task fine-tune the hypernetwork. The hypernetwork takes in input in the Def+2Pos format and generates the prefixes for the layerwise key and value representations [12]. We further adopt a **fusion-in-decoder** strategy, originally designed for question answering tasks [9, 27]. [8] has validated its effectiveness for hypernetworks, where the main model's decoder attends to the concatenated outputs from both the hypernetwork's encoder and the main model's encoder. We fine-tune the hypernetwork for 10k steps in T5-Large experiments and 20k steps in T5-XL experiments. We use the Adam-8bit optimizer [10, 3] with a learning rate of 5×10^{-5} and a batch size of 256. We select $\alpha \in \{1, 5, 10, 20\}$. Further implementation details are deferred to Appendix 6.1.

Inference. For each task in the held-out test set, the hypernetwork takes in input in the Def+2Pos format and generates a set of prefixes. The main model then predicts all task queries using this single set of prefixes and the fused output from both encoders.

Full Fine-tuning Baselines. We list as references the baselines for fine-tuning the full model:
• **FT-Zero-Shot.** We multi-task fine-tune a model, which predicts the response to a query.
• **FT-Few-Shot.** We multi-task in-context fine-tune a model, which takes the concatenation of the task definition, two fixed positive demonstrations and a query as input, and predicts the response.
• **T*k*-Instruct** [23] mainly differs from **FT-Few-Shot** in that each positive demonstration is further followed by an expert-written explanation.
• **HINT** [8] is a hypernetwork approach where the hypernetwork and the main model share weights and are jointly fine-tuned. It consists of two stages: 1) The hypernetwork is pre-trained on the C4 corpus [18]. Each input string is split into three random-length chunks. The hypernetwork takes the first chunk as input to generate PEFT parameters. The main model takes the second chunk as input to predict the third chunk. 2) The hypernetwork and the main model share weights and are jointly in-context multi-task fine-tuned.

Parameter Efficient Fine-tuning (PEFT) Baselines. We compare with baselines for fine-tuning the PEFT parameters:

[2]LM-adapted T5 models further improve T5 models in activation function, dropout, parameter sharing and data filtration. We will omit "LM-adapted" when referring to the T5 models for the rest of the paper.

• **LoRA.** [7] propose to add a pair of rank-decomposition weight matrices to each attention weight matrix and only fine-tune these matrices. We apply LoRA to **FT-Few-Shot**.
• **Prefix-Tuning.** [12] propose to prepend a prefix to each key and value representation in attention modules and only train the prefixes. We apply Prefix-Tuning to **FT-Few-Shot**.
• **HyperTuning-PT** [17] is a hypernetwork approach where the hypernetwork is fine-tuned while the main model is frozen. Similar to HINT, HyperTuning also consists of two stages: 1) The hypernetwork is pre-trained on the C4 corpus ("PT" stands for pre-training). Each input string is split into four chunks of predefined length. The hypernetwork takes the first and fourth chunk as input to generate PEFT parameters. The main model takes the second chunk as input to predict the third chunk. 2) The hypernetwork is in-context multi-task fine-tuned (Eq. 2).
• **HyperTuning** is our re-implementation of HyperTuning-PT where we remove the hypernetwork pre-training and adopt the fusion-in-decoder strategy.

Comparison of the Hypernetwork Approaches. Table 1 summarizes the differences between HART, HyperTuning-PT and HINT. Compared with HINT, HART unties the weights between the hypernetwork and the main model. While weight-sharing fine-tunes both models jointly and therefore achieves performance close to full fine-tuning, weight-untying can search for better PEFT parameters through leveraging a stronger hypernetwork (as we will see in Table 2). Furthermore, weight-freezing retains the benefits of conventional PEFT methods, e.g., it prevents catastrophic forgetting and saves the storage cost.

Compared with HINT and HyperTuning-PT, HART introduces autoregressive parameters generation and local consistency regularization. HART removes the hypernetwork pre-training to accommodate the computational budget, and adds the fusion-in-decoder approach to alleviate the resulting performance degradation (Table 5). We remark that the hypernetwork pre-training would be complimentary to HART.

Table 1: Comparison of HINT, HyperTuning-PT and HART.

Method	Untie weights	Autoregressive generation	Pre-train hypernetwork	Adopt Fusion -in-decoder
HINT [8]	No	No	Yes	Yes
HyperTuning-PT [17]	Yes	No	Yes	No
HART	Yes	Yes	No	Yes

Main Results. Table 2 shows the evaluation results of the T5-Large and T5-XL main models on the S-NI held-out test set. HART achieves an improvement of 1.6 points over HyperTuning in both the T5-Large and T5-XL experiments, demonstrating the effectiveness of autoregressive decoding and consistency regularization strategies. Compared with Prefix-Tuning and LoRA, HART achieves around 3 points of gain, suggesting that the PEFT parameters learned by HART are more generalizable than those learned by conventional PEFT methods.

Table 2: Evaluation results of the T5-Large/XL main model on the S-NI held-out test set. For all hypernetwork approaches, we use the T5-Large/XL as the initial model to train the hypernetwork.

	Avg. ROUGE-L	
Method	T5-Large	T5-XL
Full Fine-tuning		
FT-Zero-Shot	40.6	46.6
FT-Few-Shot	47.6	54.0
T*k*-Instruct [23]	48.0	54.3
HINT [8]	-	53.2
Parameter-Efficient Fine-tuning (PEFT)		
Prefix-Tuning [12]	42.6	47.1
LoRA [7]	42.9	47.7
HyperTuning-PT [17]	43.5	48.6
HyperTuning	45.2	48.8
HART	46.8	50.4

4.2 Public Pool of Prompts (P3)

Dataset. Public Pool of Prompts (P3, [19]) is a collection of English datasets covering 62 tasks. Each task consists of a set of input-output pairs formatted in manually-written prompt templates. P3 was initially collected for zero-shot settings, so it does not contain a demonstration set. Therefore, at each iteration, we sample 16 prompts from the training set and concatenate them to form the hypernetwork input, denoted as "16-Shots". We evaluate the main model on the held-out validation set using the multiple-choice scoring of accuracy. All model initialization, training and inference configurations follow Section 4.1.

Main Results. Table 3 and Table 4 show the evaluation results of T5-Large and T5-XL main models on the P3 held-out validation set, respectively. HART achieves 2.0 and 3.6 points of improvement over HyperTuning in the T5-Large and T5-XL experiments, respectively. However, as observed, HyperTuning significantly underperforms HyperTuning-PT. We suspect that this is because the P3 collection is difficult to fit. In this case, the fusion-in-decoder approach becomes less effective as the encoder is too weak to extract meaningful representations, making hypernetwork pre-training crucial for facilitating model convergence. Despite a lower baseline, HART still outperforms conventional PEFT methods and achieves performance comparable to the full fine-tuning baselines.

Table 3: Evaluation results of T5-Large main model on the P3 held-out validation set. For all hypernetwork approaches, we use the T5-Large as the initial model to train the hypernetwork.

Method	ANLI	HSwag	CB	COPA	RTE	WiC	WSC	WGD	Avg.
Full Fine-tuning									
FT-Zero-Shot	33.4	28.0	63.0	77.9	71.1	50.8	61.0	53.4	54.8
FT-Few-Shot	35.3	27.5	68.6	70.5	75.2	51.7	62.1	52.2	55.4
Parameter-Efficient Fine-tuning (PEFT)									
Prefix-Tuning [12]	33.1	26.1	53.9	67.8	60.5	49.8	54.7	51.4	49.7
LoRA [7]	31.8	26.3	48.6	61.4	71.3	51.5	63.0	51.1	50.6
HyperTuning-PT[17]	33.4	32.3	60.1	73.9	71.5	51.1	63.0	51.1	54.6
HyperTuning	33.4	28.5	59.4	68.6	67.9	50.6	62.8	52.9	53.0
HART	33.6	28.4	70.2	70.1	72.2	50.3	62.3	53.0	55.0

Table 4: Evaluation results of the T5-XL main model on the P3 held-out validation set. For all hypernetwork approaches, we use the T5-XL as the initial model to train the hypernetwork.

Method	ANLI	HSwag	CB	COPA	RTE	WiC	WSC	WGD	Avg.
Full Fine-tuning									
T0-3B [19]	33.4	27.3	45.4	72.8	64.6	50.6	64.9	50.9	54.9
FT-Zero-Shot	39.9	29.4	64.5	88.0	80.8	51.7	60.7	57.9	59.1
FT-Few-Shot	37.9	30.9	67.6	90.5	76.6	51.2	63.3	61.1	59.9
HINT [8]	41.6	30.3	76.0	88.8	84.2	51.4	59.5	60.1	65.4
Parameter-Efficient Fine-tuning (PEFT)									
Prefix-Tuning [12]	38.3	31.2	61.4	82.4	78.6	52.6	57.0	54.3	57.0
LoRA [7]	38.5	28.8	57.6	80.1	78.2	52.0	59.3	53.3	56.0
HyperTuning-PT [17]	38.7	33.6	69.6	88.4	79.5	53.1	57.6	56.6	59.6
HyperTuning	36.8	26.6	54.6	79.4	76.8	52.3	54.8	50.8	54.0
HART	37.8	28.5	66.7	80.8	79.4	50.5	59.5	57.1	57.6

5 Conclusion

To achieve an efficient task adaptation, we propose a novel hypernetwork training approach, HART. HART incorporates an autoregressive decoding scheme to exploit layerwise dependencies and a consistency regularization technique to improve training stability, allowing the hypernetwork to generate more expressive task-specific PEFT parameters for pre-trained models.

References

[1] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[2] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[3] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

[4] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

[5] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[6] 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 *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[7] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[8] Hamish Ivison, Akshita Bhagia, Yizhong Wang, Hannaneh Hajishirzi, and Matthew Peters. Hint: Hypernetwork instruction tuning for efficient zero-shot generalisation. *arXiv preprint arXiv:2212.10315*, 2022.

[9] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[12] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[13] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[14] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021.

[15] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

[16] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[17] Jason Phang, Yi Mao, Pengcheng He, and Weizhu Chen. Hypertuning: Toward adapting large language models without back-propagation. *arXiv preprint arXiv:2211.12485*, 2022.

[18] 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. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[19] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

[20] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[21] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[22] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

[23] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109, 2022.

[24] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[25] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[26] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

[27] Qinyuan Ye, Iz Beltagy2 Matthew E Peters, Xiang Ren, and Hannaneh Hajishirzi. Investigating fusion methods for in-context learning.

[28] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[29] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

6 Appendix

6.1 Implementation Details

6.1.1 Hypernetwork Architecture

The hypernetwork takes the encoder-decoder LM-adapted T5 Transformer architecture [18, 11]. T5 models are encoder-decoder Transformer-based models pre-trained using web-scale text-to-text corpus. LM-adapted T5 models further improve T5 models in activation function, dropout, parameter sharing and data filtration.

T5-Large and T5-XL models each consists of a 24-layer encoder and a 24-layer decoder. T5-Large has a hidden dimension of 1024 and T5-XL has a hidden dimension of 2048. For training efficiency, we adopt the first 8 layers of the decoder to initialize the hypernetwork's decoder.

6.1.2 Parameter Generation in Details

In Sections 2 and 3, We introduce the parameter generation schemes for HyperTuning [17] and HART with some simplifications for presentation clarity. In this section, we provide the full details in their prefix generation schemes.

Notations. We denote the length of the prefix to be generated as p, which is set to be 32 in both methods. We denote the hidden dimension of both the hypernetwork and the main model as d_{m}, which is 1024 in T5-Large and 2048 in T5-XL. We denote the number of layers in both the main model's encoder and decoder as L, which is 24 in both T5-Large and T5-XL.

Parameter Generation in HyperTuning. In HyperTuning, the input to the hypernetwork's decoder is a learnable embedding with $2p$ as the sequence length, denoted as $z \in \mathbb{R}^{2p \times d_{\mathrm{m}}}$. At each forward pass, the decoder takes in z and generates a hidden state $h \in \mathbb{R}^{2p \times d_{\mathrm{m}}}$ using bi-directional self-attention following Eq. 3. The decoder conditions on the few-shot demonstration examples by crossly attending to the hypernetwork's encoder's output representation.

For each layer of the main model, two layer-specific MLPs would be learned to project the hidden state h to the key and value prefixes, respectively. In other words, for a $2L$-layer main model, there are $4L$ MLPs. Each MLP consists of a layer normalization, 2 linear projections each with a dimension of $\mathbb{R}^{d_{\mathrm{m}} \times d_{\mathrm{m}}}$, and a tanh non-linear activation.

We denotes the MLP that learns the key/value prefix for the l-th layer of the encoder/decoder as $\mathtt{MLP}^{l}_{\mathrm{enc,key}}(\cdot)$, $\mathtt{MLP}^{l}_{\mathrm{enc,value}}(\cdot)$, $\mathtt{MLP}^{l}_{\mathrm{dec,key}}(\cdot)$, and $\mathtt{MLP}^{l}_{\mathrm{dec,value}}(\cdot)$, respectively. $\mathtt{MLP}^{l}_{\mathrm{enc,key}}(\cdot)$ and $\mathtt{MLP}^{l}_{\mathrm{enc,value}}(\cdot)$ would take $h[:p,:] \in \mathbb{R}^{p \times d_{\mathrm{m}}}$ as input and produce $\phi^{l}_{\mathrm{enc,key}}$ and $\phi^{l}_{\mathrm{enc,value}}$, both in $\mathbb{R}^{p \times d_{\mathrm{m}}}$, as the key and value prefixes for the l-th layer of the encoder. Similarly, $\mathtt{MLP}^{l}_{\mathrm{dec,key}}(\cdot)$ and $\mathtt{MLP}^{l}_{\mathrm{dec,value}}(\cdot)$ would take $h[-p:,:] \in \mathbb{R}^{p \times d_{\mathrm{m}}}$ as input and produce $\phi^{l}_{\mathrm{dec,key}}$ and $\phi^{l}_{\mathrm{dec,value}}$ as the key and value prefixes for the l-th layer of the decoder.

For training efficiency, $\{\mathtt{MLP}^{l}_{\mathrm{enc,key}}\}_{l=1}^{L}$ share the weights of their first linear projections. $\{\mathtt{MLP}^{l}_{\mathrm{enc,value}}\}_{l=1}^{L}$, $\{\mathtt{MLP}^{l}_{\mathrm{dec,key}}\}_{l=1}^{L}$ and $\{\mathtt{MLP}^{l}_{\mathrm{dec,value}}\}_{l=1}^{L}$ share their weights in a similar fashion.

Parameter Generation in HART. In HART, the input to the hypernetwork's decoder is a learnable embedding with 2 as the sequence length, denoted as $z \in \mathbb{R}^{2 \times d_{\mathrm{m}}}$. At each forward pass, the decoder takes in z and autoregressively decodes a sequence of hidden states $h_1, ..., h_L$, each with dimension $\mathbb{R}^{2 \times d_{\mathrm{m}}}$. The decoder conditions on the few-shot task-specific demonstration examples by crossly attending to the hypernetwork's encoder's output representation.

For each layer of the main model, two layer-specific MLPs would be learned to project the hidden state h to the key and value prefixes, respectively. Each MLP consists of a layer normalization, 2 linear projections and a tanh non-linear activation. The first linear projection is of dimension $\mathbb{R}^{d_{\mathrm{m}} \times p d_{\mathrm{m}}}$ and the second is of dimension $\mathbb{R}^{d_{\mathrm{m}} \times d_{\mathrm{m}}}$.

Following the same notations from HyperTuning, $\mathtt{MLP}^{l}_{\mathrm{enc,key}}(\cdot)$ and $\mathtt{MLP}^{l}_{\mathrm{enc,value}}(\cdot)$ would take $h[:1,:] \in \mathbb{R}^{d_{\mathrm{m}}}$ as input. After their first linear projections, the intermediate outputs are of dimension $\mathbb{R}^{p d_{\mathrm{m}}}$. We further reshape the intermediate outputs into the dimension $\mathbb{R}^{p \times d_{\mathrm{m}}}$, which is then projected by their second linear projections into $\phi^{l}_{\mathrm{enc,key}}$ and $\phi^{l}_{\mathrm{enc,value}}$, the key and value prefixes for the

l-th layer of the encoder. Both prefixes are of dimension $\mathbb{R}^{p \times d_{\mathrm{m}}}$. Similarly, $\texttt{MLP}^{l}_{\mathrm{dec,key}}(\cdot)$ and $\texttt{MLP}^{l}_{\mathrm{dec,value}}(\cdot)$ would take $h[-1 :, :] \in \mathbb{R}^{d_{\mathrm{m}}}$ as input and produce $\phi^{l}_{\mathrm{dec,key}}$ and $\phi^{l}_{\mathrm{dec,value}}$ as the key and value prefixes for the l-th layer of the decoder.

For training efficiency, $\{\texttt{MLP}^{l}_{\mathrm{enc,key}}\}_{l=1}^{L}$ share the weights of their first linear projections. $\{\texttt{MLP}^{l}_{\mathrm{enc,value}}\}_{l=1}^{L}$, $\{\texttt{MLP}^{l}_{\mathrm{dec,key}}\}_{l=1}^{L}$ and $\{\texttt{MLP}^{l}_{\mathrm{dec,value}}\}_{l=1}^{L}$ share their weights in a similar fashion.

Remark regarding Weight Sharing and Input Sharing in MLPs. We remark that in HyperTuning and HART, the MLPs are not completely independent across layers because they share the same input hidden state and the first linear projection layers. Such input and weight sharing indeed allow MLPs to learn the pattern of layerwise dependency through training, but they need to learn it from scratch. In contrast, autoregressive decoding allows the hypernetwork to directly exploit the layerwise pattern without learning. This pattern is an useful inductive bias that improves the sample efficiency during training.

6.1.3 Training Details

Multi-task Training Data Taxonomy. We adhere to the training and evaluation subsets used in [17] for both P3 and S-NI datasets.

For P3, we use the T0 model's training subset, specifically selecting tasks with an average input length of fewer than 320 tokens following HyperTuning. We also use the T0 model's evaluation subset after excluding StoryCloze, as StoryCloze is not publicly distributed. The task taxonomy for both training and evaluation subsets are illustrated in [19]. The complete list of training tasks are provided in [17].

For S-NI, we utilize the training and evaluation split from the Super-NaturalInstructions V2 dataset, excluding non-English tasks [23].

Multi-task Training Data Sampling. We follow the multi-task in-context fine-tuning setting from MetaICL [14]. At each training iteration, we first randomly sample a task from the training task pool, and then randomly sample K shot demonstration examples and one training example from this task. During inference, for each task, we use a fixed set of demonstration examples for all test queries.

For SN-I, we construct the input to the hypernetwork as "Def+2Pos" because this format has been observed to outperform other formats [23]. Following [17], we select the English tasks for training and evaluation.

For P3, we exclude training tasks with average sequence lengths longer than 320 tokens to fit more prompts into the input following [17].

Fusion-in-decoder. We further adopt a fusion-in-decoder strategy, originally designed for question answering tasks [9, 27]. This strategy requires the decoder to attend to concatenated representations of multiple encoded input contexts. [8] has validated its effectiveness for hypernetworks. Specifically, at each forward pass, we prepend the hypernetwork's encoder output to the main model's encoder output, and require the main model's decoder to attend to such a fused representation in the cross attention module. This approach is adopted in both the training and inference stages.

Hyperparameters. We fine-tune the hypernetwork for 10k steps in T5-Large experiments and 20k steps in T5-XL experiments. For both model experiments, we use the Adam-8bit optimizer [10, 3] with a learning rate of 5×10^{-5} and a batch size of 256. We adopt a linear decay learning rate schedule. We select $\alpha \in \{1, 10, 20\}$. We set the maximum input sequence length for the hypernetwork as 1024 and the prefix length as 32. For the main model, we set the maximum input and target sequence length as 384 and 128. We adopt the same input sequence length and target sequence length during inference.

We use deepspeed library for distributed training and inference. The T5-Large experiments are conducted on 8 Nvidia 32G V100 GPUs and T5-XL experiments are conducted on 8 Nvidia 80G A100 GPUs.

6.2 Analysis

In this section, we justify the design choices for HART. All experiments herein utilize a T5-Large model for initializing both the hypernetwork and the main model.

6.2.1 Ablation Study

Table 5 shows an ablation study of autoregressive parameter generation and local consistency regularization on the S-NI held out test set and the P3 held-out validation set. Starting with HyperTuning, we first remove the hypernetwork pre-training stage, then incorporate the fusion-in-decoder approach, and sequentially introduce the proposed strategies. We observe that autoregressive parameter generation contributes over a one-point gain on both benchmarks, with local consistency adding an additional gain of approximately 0.5 points.

Table 5: Ablation study of the proposed components on the S-NI held out test set and the P3 held-out validation set.

Method	S-NI Avg. ROUGE-L	P3 Avg. Score
HyperTuning-PT	43.5	54.6
- Continual Pre-training	25.3	50.9
+ Fusion-in-Decoder (HyperTuning)	45.2	53.0
+ Autoregressive Parameter Generation	46.4	54.3
+ Local Consistency Regularization (HART)	46.8	55.0

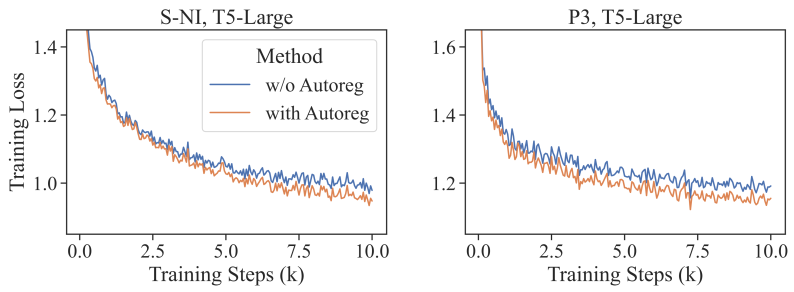

Figure 3: The main model prediction losses with and without using autoregressive parameter generation on the S-NI and the P3 training sets.

6.2.2 Autoregressively Generated Parameters Fit Better

Figure 3 shows the main model's prediction losses with and without autoregressive parameter generation on the S-NI and the P3 training sets. We present the loss curves corresponding to the experiments at the third and the fourth rows in Table 5, without regularizing the local consistency. By using autoregressively generated parameters, the training loss converges faster, suggesting that the parameters better fit the training data.

6.2.3 Local Consistency Regularization Reduces Loss Variance

Figure 4 shows the main model's prediction losses with and without local consistency regularization the P3 training set. We present the loss curves corresponding to the experiments at the fourth and the last rows in Table 5. Local consistency regularization alleviates the loss spikes and reduces the loss variances, suggesting the training is more stable.

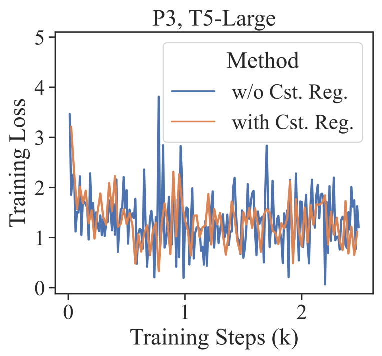

Figure 4: The main model prediction losses with and without using local consistency regularization on the P3 training set.