8 APPENDIX

8.1 IMPLEMENTATION DETAILS

8.1.1 HYPERNETWORK ARCHITECTURE

The hypernetwork takes the encoder-decoder T5 Transformer architecture (Raffel et al., 2020). 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.

8.1.2 PARAMETER GENERATION IN DETAILS

In Sections 2 and 3, We introduce the parameter generation schemes for HyperTuning (Phang et al., 2022) 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 $\texttt{MLP}^l_{\mathrm{enc,key}}(\cdot)$, $\texttt{MLP}^l_{\mathrm{enc,value}}(\cdot)$, $\texttt{MLP}^l_{\mathrm{dec,key}}(\cdot)$, and $\texttt{MLP}^l_{\mathrm{dec,value}}(\cdot)$, respectively. $\texttt{MLP}^l_{\mathrm{enc,key}}(\cdot)$ and $\texttt{MLP}^l_{\mathrm{enc,value}}(\cdot)$ 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, $\texttt{MLP}^l_{\mathrm{dec,key}}(\cdot)$ and $\texttt{MLP}^l_{\mathrm{dec,value}}(\cdot)$ 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, $\{\texttt{MLP}^l_{\mathrm{enc,key}}\}_{l=1}^L$ 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.

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, $\texttt{MLP}^l_{\mathrm{enc,key}}(\cdot)$ and $\texttt{MLP}^l_{\mathrm{enc,value}}(\cdot)$ 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, $\{\texttt{MLP}^l_{\text{enc,key}}\}_{l=1}^L$ share the weights of their first linear projections. $\{\texttt{MLP}^l_{\text{enc,value}}\}_{l=1}^L$, $\{\texttt{MLP}^l_{\text{dec,key}}\}_{l=1}^L$ and $\{\texttt{MLP}^l_{\text{dec,value}}\}_{l=1}^L$ 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.

8.1.3 Training Details

Multi-task Training Data Sampling. We follow the multi-task in-context fine-tuning setting from MetaICL (Min et al., 2021). 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 P3 training data, we exclude tasks with average sequence lengths longer than 320 tokens to fit more prompts into the input following Phang et al. (2022).

Fusion-in-decoder. We further adopt a fusion-in-decoder strategy, originally designed for question answering tasks (Izacard & Grave, 2020; Ye et al.). This strategy requires the decoder to attend to concatenated representations of multiple encoded input contexts. Ivison et al. (2022) 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 (Kingma & Ba, 2014; Dettmers et al., 2021) 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.

8.2 Adjacent Layers Exhibit Stronger Dependency

Table 7 shows the evaluation results under different decoding schemes: 1) generate a single layer-shared state (HyperTuning); 2) generate layer-specific states autoregressively for a randomly-chosen, fixed order of layers; 3) generate layer-specific states autoregressively from the top layer to the bottom layer; 4) generate layer-specific states autoregressively from the bottom layer to the top layer (HART). All experiments are conducted without regularizing the local consistency. We can observe that 2), 3) and 4) achieve noticeable improvements upon 1), demonstrating the benefit of utilizing the capabilities of the decoder. Furthermore, 5) outperforms 3), suggesting that exploiting the underlying problem structure improves the sample efficiency.

8.3 Visualizing Consistency Regularized Parameters

Figure 5 (Left) showcases the value of $\mathcal{L}_{\text{cst}}$ with and without local consistency regularization on the S-NI training set. We can observe that the generated parameters no longer change drastically across iterations after applying local consistency regularization. Figure 5 (Right) shows the t-SNE plot of the hidden states generated for different layers with and without applying local consistency regularization on four P3 held-out test tasks. One observation is that, with or without regularization, the states generated for different tasks are clustered, while the states generated for different layers are scattered. This suggests that layer-specific weight structure maybe more distinct than task-specific weight structure. Another observation is that, after applying consistency regularization, the states generated for different layers become more diverse.

Table 7: Evaluation results of the T5-Large model on S-NI test set with PEFT parameters generated under different layerwise dependencies. *The result is obtained by averaging over three different randomly-chosen orders.

Generate Multiple Layer-specific States?	Layerwise Dependency	S-NI Avg. ROUGE-L
No	N/A (HyperTuning)	45.2
Yes, autoregressively	Depend on one random, fixed layer*	45.5
Yes, autoregressively	Depend on the next layer	45.8
Yes, autoregressively	Depend on the previous layer (HART)	46.4

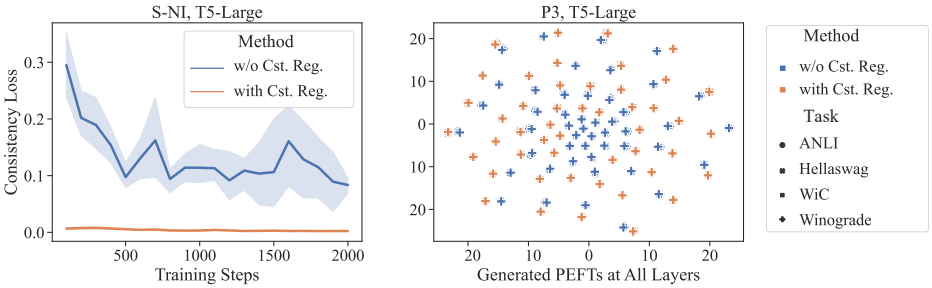

Figure 5: *Left*: The value of $\mathcal{L}_{cst}$ with and without local consistency regularization on the S-NI training set. *Right*: The t-SNE plot of the hidden states generated for different layers with and without applying local consistency regularization on four P3 held-out test tasks.