Instant Transformer Adaption via HyperLoRA

Rujikorn Charakorn
Sakana.AI, VISTEC
rujikorn.ch@gmail.com

Edoardo Cetin
Sakana.AI
edo@sakana.ai

Yujin Tang
Sakana.AI
yujintang@sakana.ai

Robert Lange
Sakana.AI
robert@sakana.ai

Abstract

While Foundation Models provide a general tool for rapid content creation, they regularly require task-specific adaptation. Traditionally, this exercise involves careful curation of datasets and repeated fine-tuning of the underlying model. Fine-tuning techniques enable practitioners to adapt foundation models for many new applications but require expensive and lengthy training while being notably sensitive to hyper-parameter choices. To overcome these limitations, we introduce **HyperLoRA**, a model capable of adapting Large Language Models *on the fly*—solely based on a natural language description of the target task. **HyperLoRA** is a hypernetwork trained to construct LoRAs in a single inexpensive forward pass. After training **HyperLoRA** on a suite of 9 pre-trained LoRA adapters (GSM8K, Arc, etc.), we show that the ad-hoc reconstructed LoRA instances match the performance of task-specific adapters across the corresponding test sets. Furthermore, **HyperLoRA** can compress hundreds of LoRA instances and zero-shot generalize to entirely unseen tasks. This approach provides a significant step towards democratizing the specialization of foundation models and enables language-based adaptation with minimal compute requirements. Our code and pre-trained checkpoints will be available through GitHub and HuggingFace upon publication.

1 Introduction

Biological systems are capable of rapid adaptation, given limited sensory cues. For example, the human visual system can tune its light sensitivity and focus through neuromodulation of the fovea and rod cells (Wurtz et al., 2011; Digre & Brennan, 2012). While recent Large language models (LLMs) exhibit a wide variety of capabilities and knowledge, they remain rigid when adding task-specific capabilities. In such cases, practitioners often resort to re-training parts of the model (Gururangan et al., 2020; Wei et al., 2021; Dettmers et al., 2022; Tay et al., 2021) using techniques such as parameter-efficient fine-tuning, e.g. via Low-Rank Adaptation (LoRA, Hu et al., 2022). Typically, a LoRA adapter has to be optimized for each individual downstream task and requires a task-specific dataset and hyperparameter setting. This fine-tuning scheme for adaptation significantly limits the possibility of transferring knowledge between tasks and induces engineering overhead.

Recently, it has been observed that by inducing structural constraints, the low-rank matrices learned by LoRA adapters can be further compressed. For example, one can train *lossy* versions of the original adapter while maintaining downstream performance (Brüel-Gabrielsson et al., 2024; Kim et al., 2024; Kopiczko et al., 2024). Furthermore, multiple LoRAs can be combined for new tasks at inference time (Ostapenko et al., 2024). At the core of these approaches lies the explicit use of decomposition or dimensionality reduction techniques (e.g., SVD or routing) for better compression and online composition of existing LoRAs. This raises the following questions:

1. Can we end-to-end train a neural network to compress many pre-trained LoRAs?
2. Can we decode new task-specific LoRA adapters from this network solely based on natural-language instructions for an unseen task at test time?

We hypothesize that different LoRA adapters share the same underlying adaptation mechanism and can be optimized simultaneously without any explicit structure or recipe for combining them. To explicitly test this hypothesis, we propose **HyperLoRA** (see Section 1), a hypernetwork (Ha et al.,

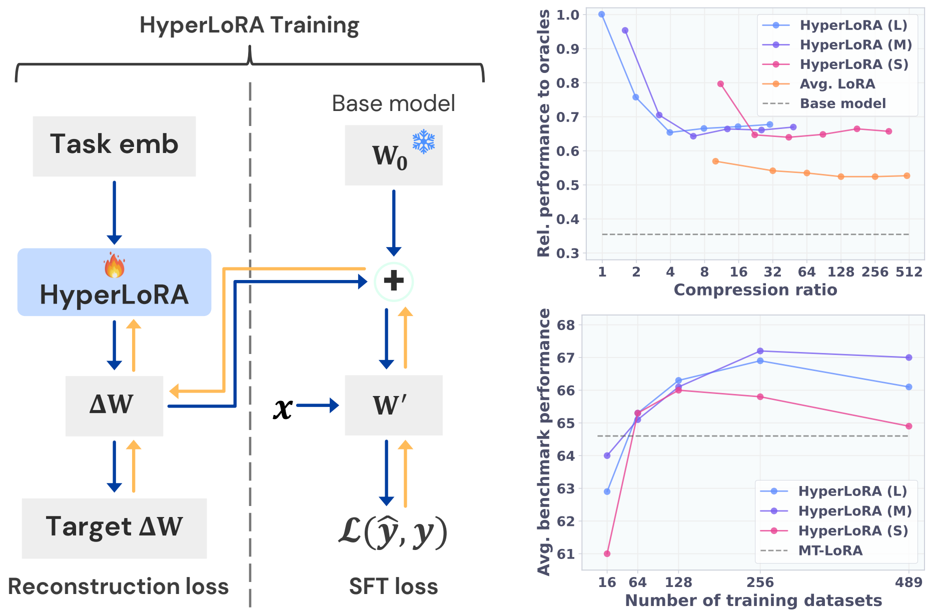


Figure 1: **Left:** Conceptual overview of the HyperLoRA training routine. Given task description embeddings, we train a hypernetwork to generate LoRA adaptation matrices (ΔW) for various tasks. The weights of the HyperLoRA are either optimized to distill pre-trained LoRA weights or via multi-task supervised fine-tuning on downstream tasks. **Right, Top:** Relative performance to the oracles on training SNI tasks with varying compression ratios. **Right, Bottom:** Zero-shot LoRA generation performance on 10 benchmark tasks. As we increase the number of pre-training datasets, the performance of HyperLoRA increases for three different HyperLoRA architectures.

2016) that compresses task-specific LoRAs and generates a new LoRA adapter *zero-shot* at inference time. HyperLoRA is trained to compress LoRAs on a diverse task distribution from the Super Natural Instruction (SNI) dataset (Wang et al., 2022). Importantly, HyperLoRA takes a natural language description of the target task as an input, allowing zero-shot LoRA generation to unseen tasks. Empirically, we show that HyperLoRA can effectively be trained either to reconstruct pre-trained adapters or via supervised fine-tuning on a distribution of downstream tasks (see Fig. 1, top right). After training HyperLoRA outperforms a multi-task LoRA baseline and Arrow Routing (Ostapenko et al., 2024), a state-of-the-art zero-shot LoRA routing method, on various benchmark tasks. Furthermore, we show that HyperLoRA can generate LoRA adapters for previously unseen tasks solely using the language-based task description. This highlights the generalization capabilities and applicability of our proposed indirect adaptation encoding. Our contributions are summarized as follows:

1. We introduce a hypernetwork-based architecture for producing LoRA adapters with a single forward pass (Section 3) and based on text instructions. The HyperLoRA architecture can be trained using both distillation of pre-trained adapters and supervised multi-task fine-tuning.
2. We show that HyperLoRA can efficiently encode hundreds of LoRA adapters (Section 4). While the compression is lossy, HyperLoRA maintains the performance of task-specifically tuned LoRA adapters. Furthermore, HyperLoRA can generalize to unseen tasks given suitable natural language instructions.
3. We provide rigorous ablations (Appendix A) including HyperLoRA scaling with datasets (see Fig. 1, bottom right), the impact of different task description embeddings, the training routine and text-based task descriptions.
4. Finally, we provide various efforts to analyze the nature of the HyperLoRA generations (Section 5). We study the relationship between LoRA adapters and find compelling evidence why reconstruction-trained HyperLoRA cannot generalize. Furthermore, we find semantically meaningful LoRA clusters when visualizing the generated LoRAs in a dimensionality-reduced space.

2 Preliminaries

We utilize multiple fine-tuning datasets $\mathcal{D} = \{\mathcal{D}^1, \dots, \mathcal{D}^T\}$, which correspond to different tasks $\mathcal{T} = \{t^1, \dots, t^T\}$. For the purpose of training `HyperLoRA`, we assume that each fine-tuning dataset has a set of natural language *task descriptions* ($Z^i = \{z_1^i, \dots, z_m^i\}$): $\mathcal{D}^i = \{X^i, Y^i, Z^i\}$. The task descriptions do not need to be specific to each sample but rather a general description of the dataset. For a single task t^i, the fine-tuning objective of an LLM with pre-trained weights (Ψ) is given by

$$\Delta \boldsymbol{W}^i = \underset{\Delta \boldsymbol{W}^i}{\arg\min}\, \mathcal{L}_{\text{SFT}}(\mathcal{D}^i, \Psi, \Delta \boldsymbol{W}^i), \tag{1}$$

where $\mathcal{L}_{\text{SFT}}$ gives the supervised fine-tuning loss and $\Delta \boldsymbol{W}^i$ is the fine-tuning adaption for task t^i to the base weights. For the *multi-task* setting, we train a single adapter $\Delta \boldsymbol{W}$ to minimize the expected loss over the union of all datasets $\mathcal{D}$:

$$\Delta \boldsymbol{W} = \underset{\Delta \boldsymbol{W}}{\arg\min}\, \mathbb{E}_{\mathcal{D}^i \sim \mathcal{D}}\ \mathcal{L}_{\text{SFT}}(\mathcal{D}^i, \Psi, \Delta \boldsymbol{W}). \tag{2}$$

Low-Rank Adaptation (LoRA, Hu et al., 2022): LoRA is a parameter-efficient fine-tuning method that freezes the pre-trained weights of a base model and only learns low-rank weight matrices, which serve as an adapter to the base model. For each selected linear transformation $h = \boldsymbol{W}_0 \boldsymbol{x}$, the fine-tuned transformation is given by $h = \boldsymbol{W}_0 \boldsymbol{x} + \Delta \boldsymbol{W} \boldsymbol{x} = \boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{B}^T \boldsymbol{A} \boldsymbol{x}$, where $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{r \times d}$ are weight matrices of rank $r < d$. We drop the layer index and module type of the LoRA weights when referring to all LoRA weights simultaneously. Otherwise, we use subscripts to represent the layer index and module type, e.g., $\Delta \boldsymbol{W}_{m,l}$, where m is the module type (e.g., query projection) and l is the layer index.

Hypernetworks: A hypernetwork is a neural network that generates parameters for another 'base' network (Ha et al., 2016). It serves as an indirect encoding (Schmidhuber, 1997; Stanley & Miikkulainen, 2003) of the base network, given that the parameter count of the hypernetwork is much smaller. This compression is achieved by learning to share parameters indirectly. More specifically, given a layer-specific descriptor vector ϕ_l, a hypernetwork with parameters θ generates the parameters of the base model at layer $l \in \{1, \dots L\}$ as follows: $\boldsymbol{W}_l = h_\theta(\phi_l)$. Traditionally, the layer descriptors are either one-hot or learned vectors. The weights θ are trained via end-to-end optimization on a downstream task. Hypernetworks have been applied to encode weights of various architectures (Ha et al., 2016; Zhang et al., 2018; Schug et al., 2024).

3 HyperLoRA: Learning to Compress and Generate LoRAs

In this work, we utilize a hypernetwork to generate LoRA adapters for specific fine-tuning tasks. For each target module (m) and layer index (l), a hypernetwork generates the two low-rank matrices $\boldsymbol{A}, \boldsymbol{B}$ based on a task description $z^i \in Z^i$ of a task t^i as follows:

$$\Delta \boldsymbol{W}_{m,l}^i = h_\theta(\phi_{m,l}^i), \text{ with } \phi_{m,l}^i = \texttt{concat}\left[f(z^i), E[m], E[l]\right], \tag{3}$$

where f gives a vector representation of a text description, typically represented by a `CLS` token of a bidirectional transformer model or last token activation of an LLM. E is a learnable embedding dictionary indexed by either a module type m or a layer index l. For legibility, we introduce a shorthand notation for `HyperLoRA`'s output $\Delta \boldsymbol{W}^i := h_\theta(\phi^i) := h_\theta(\{\phi_{m,l}^i\})$. Then, a supervised fine-tuning training objective for `HyperLoRA` is

$$\theta = \underset{\theta}{\arg\min}\ \mathbb{E}_{\mathcal{D}^i \sim \mathcal{D}, z^i \sim Z^i}\ \mathcal{L}_{\text{SFT}}(\mathcal{D}^i, \Psi, h_\theta(\phi^i)), \tag{4}$$

Note that values of m and l can be batched, which allows `HyperLoRA` to generate ΔW for all the modules and layer indices efficiently within a single forward-pass.

3.1 HyperLoRA Architectures

Most of a hypernetwork's parameters come from the output layer, which scales linearly with the size of the target weights (Von Oswald et al., 2019). To explore the complexity-performance trade-off, we propose three variants of `HyperLoRA`: **L**, **M**, and **S**. We impose different output spaces on the hypernetwork that represent different inductive biases and parameter counts (see Fig. 2). We

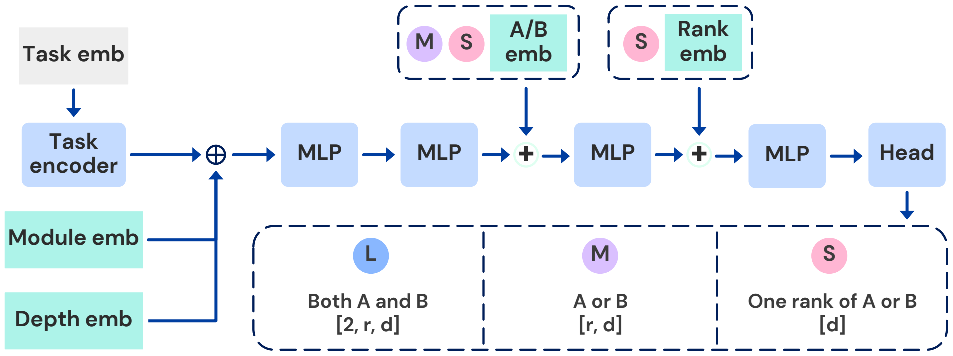


Figure 2: Overview of **HyperLoRA** architectural variations. The dashed box at the bottom shows the output size of a single forward pass of **HyperLoRA**. Blue boxes are trainable modules. Cyan Boxes are trainable embedding layers. Components in dashed boxes are only used with their corresponding architectures. r is the rank of a LoRA adapter and d is the size of the input and the output dimension.

note that all variants use the same backbone architecture and only differ in their output heads and learnable embeddings: The **L architecture** is the largest variant. Its final linear layer outputs low-rank $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices simultaneously with the number of weight connections to the output head $|\theta_{\text{head}}| = d_{\text{out}} \times 2 \times r \times d$, where d_{out} is the output size of the last MLP block. **M architecture** is the medium-size model with a shared output layer between the low-rank A and B matrices. That is, the head outputs a low-rank matrix, either A or B, depending on the learnable embedding. The size of the output head is $|\theta_{\text{head}}| = d_{\text{out}} \times r \times d$. Finally, **S architecture** is the most parameter-efficient model with the strongest inductive biases, where the hypernetwork outputs only one rank of a low-rank matrix at a time. This output space makes the size of the head much smaller: $|\theta_{\text{head}}| = d_{emb} \times d$. For reference, a LoRA adapter has $r \times d \times 2 \times L \times |M|$ trainable parameters, where L is the number of layers and $|M|$ is the number of target modules. We note that every architecture can generate the entirety of low-rank matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ in a single forward pass by batching all the input embeddings. We provide more details of the architectures in Appendix C.

3.2 Training HyperLoRA via LoRA Reconstruction (Distillation)

The most straightforward way to train **HyperLoRA** is to reconstruct pre-trained task-specific LoRAs. This setup allows us to utilize publicly available libraries of LoRAs (Brüel-Gabrielsson et al., 2024; Zhao et al., 2024). Alternatively, one can also use a two-stage procedure, in which a library of LoRAs is manually pre-trained first, and then one trains **HyperLoRA** to reconstruct them. For the sole purpose of compressing LoRAs, we can train **HyperLoRA** using one-hot or learnable vectors as task embeddings. However, these embeddings do not allow zero-shot LoRA generation for unseen tasks. To enable zero-shot LoRA generation, we additionally condition **HyperLoRA** with embeddings of natural language task descriptions, which allows **HyperLoRA** to generate LoRA adapters for various tasks—including unseen ones—given corresponding task descriptions. Given a suitable library of LoRA adapters Ω, the reconstruction loss for **HyperLoRA** can be written as

$$\mathcal{L}(\Omega, \theta) = \mathbb{E}_{\Delta \boldsymbol{W}^i \sim \Omega} \, |\Delta \boldsymbol{W}^i - h_\theta(\phi^i)|. \tag{5}$$

3.3 Training HyperLoRA via Supervised Fine-Tuning

Alternatively, **HyperLoRA** can be directly optimized on fine-tuning datasets. Training **HyperLoRA** with SFT sidesteps the need for intermediate target LoRA adapters and allows for end-to-end training of the hypernetwork. This training scheme is preferred if existing trained LoRAs are not naturally clustered by their functionalities or downstream tasks. For instance, t^1 and t^2 could be two related tasks requiring a similar LLM capability, but $\Delta \boldsymbol{W}^1$ and $\Delta \boldsymbol{W}^2$ could be in different minima. Thus, **HyperLoRA** trained via reconstruction training would have to compress numerically different $\Delta \boldsymbol{W}^1$ and $\Delta \boldsymbol{W}^2$, making it less likely to generalize. In fact, we empirically find that a **HyperLoRA** trained via reconstruction fails to generalize to unseen tasks (Appendix A.5). In contrast, an SFT-trained **HyperLoRA** can implicitly learn to cluster tasks, which has been shown to improve zero-shot LoRA routing performance (Ostapenko et al., 2024). The SFT loss for **HyperLoRA** is given by Eq. (4).

4 Experiments

We investigate the effectiveness of the different **HyperLoRA** architectures and training schemes in terms of compression of adapters and zero-shot LoRA generation for unseen tasks. We include task-specific LoRAs, element-wise average LoRA, and multi-task LoRA—a LoRA adapter trained on all training tasks—as baselines. Additionally, we include results of Arrow Routing zero-shot performance from Ostapenko et al. (2024). Note that the performance should not be compared directly as it uses a different set of LoRA adapters and training tasks, and there are likely differences in the benchmark evaluation prompts. Furthermore, **HyperLoRA** uses a multi-task (MT) LoRA as a prediction offset in all experiments because of its strong performance on the benchmarks. That is, **HyperLoRA** has to learn to generate task-specific deltas, which will be added to the offset.

In all experiments, we use `Mistral-7B-Instruct` (Jiang et al., 2023) as the base LLM model and `gte-large-en-v1.5` (Li et al., 2023; Zhang et al., 2024) for extracting the task embedding from a natural language task description. All LoRA adapters are of rank 8 and only target the query and the value projection modules in every attention block of the base LLM. With this LoRA configuration, **L**, **M**, and **S** have 55M, 34M, and 5M trainable parameters respectively.

We utilize the SNI dataset (Wang et al., 2022) for training LoRA adapters. We use a subset of 500 tasks following Brüel-Gabrielsson et al. (2024), 10 of which are manually chosen while the rest are randomly sampled. We use 11 tasks for held-out evaluation, leaving 489 datasets for training. Finally, we limit the size of each task to 500 samples to ensure compute feasibility. All samples are in English.

For evaluation, we choose 10 widely used benchmarks that collectively cover a variety of LLM capability assessments, e.g., reasoning, math, science, coding, and world knowledge. Specifically, we include the following benchmarks: Arc-challenge (ArcC) and Arc-easy (ArcE) (Clark et al., 2018), BoolQ (Clark et al., 2019), GSM8K (Cobbe et al., 2021), Hellaswag (HS) (Zellers et al., 2019), OpenBookQA (OQA) (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), Winogrande (WG) (Keisuke et al., 2019), HumanEval (HE) (Chen et al., 2021), and MBPP (Austin et al., 2021). Task descriptions for the training datasets and the benchmarks are fully generated, as described in Appendix F. When we use language task embedding as a part of the input, we average **HyperLoRA** performance using three descriptions for each benchmark.

4.1 LoRA Compression

Table 1: Benchmark performance of **HyperLoRA** trained via reconstruction loss on 9 benchmark tasks. **Green highlight** indicates that **HyperLoRA** outperform the benchmark-specific LoRA adapters.

	ArcC (acc)	**ArcE (acc)**	**BQ (acc)**	**GSM8K (acc)**	**HS (acc)**	**OQA (acc)**	**PIQA (acc)**	**WG (acc)**	**MBPP (pass@1)**	**Avg. (9 tasks)**
Base model	65.4	77.8	71.6	40.9	49.7	54.2	72.8	45.0	43.1	55.8
One-Hot Task Embeddings										
HyperLoRA (Recon) **L**	76.4	89.9	89.4	**53.8**	92.6	85.0	69.7	**51.2**	**52.6**	**73.4**
HyperLoRA (Recon) **M**	**76.7**	89.9	89.4	53.2	92.6	85.0	69.9	**51.4**	**52.9**	**73.4**
HyperLoRA (Recon) **S**	75.2	88.8	87.4	50.9	89.1	75.6	**83.9**	**58.1**	48.1	73.0
Task Description Embeddings										
HyperLoRA (Recon) **L**	76.6	89.8	89.4	**53.9**	92.6	85.0	69.6	**51.2**	51.8	73.3
HyperLoRA (Recon) **M**	76.5	89.9	89.4	**53.9**	92.5	84.9	**70.4**	**51.6**	**52.8**	**73.5**
HyperLoRA (Recon) **S**	75.4	88.8	87.8	49.1	89.7	76.7	**84.2**	**56.9**	48.0	73.0
Task-specific LoRAs	76.6	89.9	89.4	53.5	92.6	85.0	69.9	51.1	52.1	73.3

In this experiment, we aim to investigate whether **HyperLoRA** can recover the performance of trained LoRAs via reconstruction training. For quality control and consistent evaluation, we train a task-specific LoRA (oracle) on the training split of each training and benchmark task, collectively forming a library of LoRAs. Table 1 shows the benchmark performance of **HyperLoRA** trained by distilling 9 benchmark-specific LoRAs using either one-hot or natural language task embeddings from `gte-large-en-v1.5`. We can see that **HyperLoRA** fully recovers the performance of the oracle adapters with both task embedding types. Notably, **HyperLoRA** outperforms task-specific LoRAs on some benchmarks (highlighted in green). We hypothesize that the gain comes from the lossy compression of the target LoRAs, which acts as a regularization on the already trained LoRA weights. This effect is most apparent on PIQA and WG benchmarks, where the oracle LoRA overfits and performs worse than the base model.

Next, we explore whether one-hot **`HyperLoRA`** can maintain the oracles' performance when using increasing number of training tasks. Fig. 3 shows the performance of one-hot **`HyperLoRA`** on the test splits of a subset of 10 SNI training tasks with varying degree of final training L1 reconstruction error. For each architecture, we train various **`HyperLoRA`** instances using $\{16, 32, 64, 128, 256, 489\}$ training tasks, effectively increasing the training reconstruction error by training on more tasks. Although **`HyperLoRA`** fully recovers the oracles' performance when the reconstruction loss is less than 10^{-4}, we can see that the performance drops as the training error increases. This result suggests that **`HyperLoRA`** learns lossy compression of the target LoRAs. Still, we find that all **`HyperLoRA`** architectures can maintain around 65% of oracles' performance and the performance does not drop further even at $> 8 \times 10^{-4}$ per element L1 error. Despite the performance drop, we show that increasing the number of training tasks are beneficial, increasing generalization of **`HyperLoRA`** when generating LoRAs for unseen tasks in Appendix A.1.

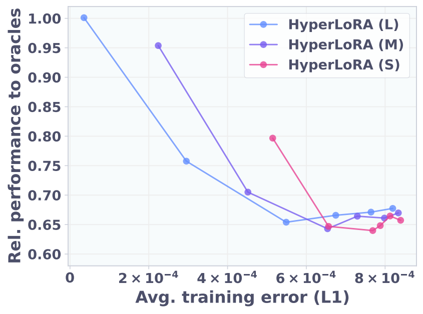


Figure 3: Relative performance (y-axis) and training reconstruction error (x-axis) of **`HyperLoRA`** instances trained with increasing number of tasks.

4.2 Zero-Shot LoRA Generation

Table 2: Zero-shot performance on unseen benchmark tasks. **`HyperLoRA`** generates LoRAs based on unseen task descriptions. Its performance is an average of three generated LoRAs, each with a different instance of descriptions. Arrow Routing results are taken from (Ostapenko et al., 2024). **Green highlight** indicates high performance than that of the benchmark-specific LoRA adapters. **Bold numbers** are used when the performance is higher than the multi-task LoRA.

	ArcC (acc)	ArcE (acc)	BQ (acc)	HS (acc)	OQA (acc)	PIQA (acc)	WG (acc)	MBPP (pass@1)	Avg. (8 tasks)	GSM8K (acc)	HE (pass@1)	Avg.
No Test-Time Adaptation												
Base model	65.4	77.8	71.6	49.7	54.2	72.8	45.0	43.1	60.0	40.9	37.2	55.8
Average LoRA	70.7	84.4	75.4	59.9	59.0	78.0	54.3	47.1	66.1	42.4	37.8	60.9
Multi-task LoRA	73.7	86.9	84.6	64.3	66.6	80.7	59.0	50.4	70.8	44.2	36.0	64.6
Zero-Shot Adaptation												
Arrow Routing	60.9	86.2	**87.6**	**80.8**	48.6	**83.0**	**68.5**	50.2	70.7	N/A	28.7	N/A
`HyperLoRA` (SFT) L	**74.0**	**87.9**	**85.1**	**65.7**	**73.8**	**82.0**	**61.7**	**53.2**	**72.9**	**46.2**	**39.8**	**66.9**
`HyperLoRA` (SFT) M	**74.9**	**88.0**	**84.9**	**65.6**	**73.9**	**82.0**	**62.1**	**52.7**	**73.0**	**47.7**	**40.2**	**67.2**
`HyperLoRA` (SFT) S	**74.1**	86.9	84.3	**64.9**	**69.9**	**80.8**	**60.5**	**52.1**	**71.7**	**45.5**	**38.6**	**65.8**
Oracle												
Task-specific LoRAs	76.6	89.9	89.4	92.6	85.0	69.9	51.1	52.1	75.8	53.5	N/A	N/A

Here, we explore whether **`HyperLoRA`** can generate useful LoRA adapters for unseen tasks. We train **`HyperLoRA`** with SFT on 256 SNI tasks, each with 128 task descriptions. For each sample in a training minibatch, we sample a description from the corresponding dataset in an online fashion. Table 2 shows the zero-shot performance on 10 benchmarks. We observe that a multi-task LoRA adapter performs considerably well on the benchmarks despite no adaptation. Still, there is a performance gap between task-specific LoRAs and MT LoRA. We can see that SFT-trained **`HyperLoRA`** indeed generates useful LoRAs, thus improving over the multi-task LoRA adapter consistently across benchmarks (indicated by bold numbers). Notably, even though **`HyperLoRA`** cannot fully bridge the performance gap, it outperforms the oracles on some tasks (highlighted in green).

We also observe that the best-performing variant in both experiments (Sections 4.1 and 4.2) is the **M** variation, which has a shared output layer for the low-rank matrices (see Fig. 2). This result corroborates with a general knowledge in the machine learning literature that certain inductive biases improve models' robustness and generalization.

5 Analysis

5.1 LoRAs of Similar Tasks

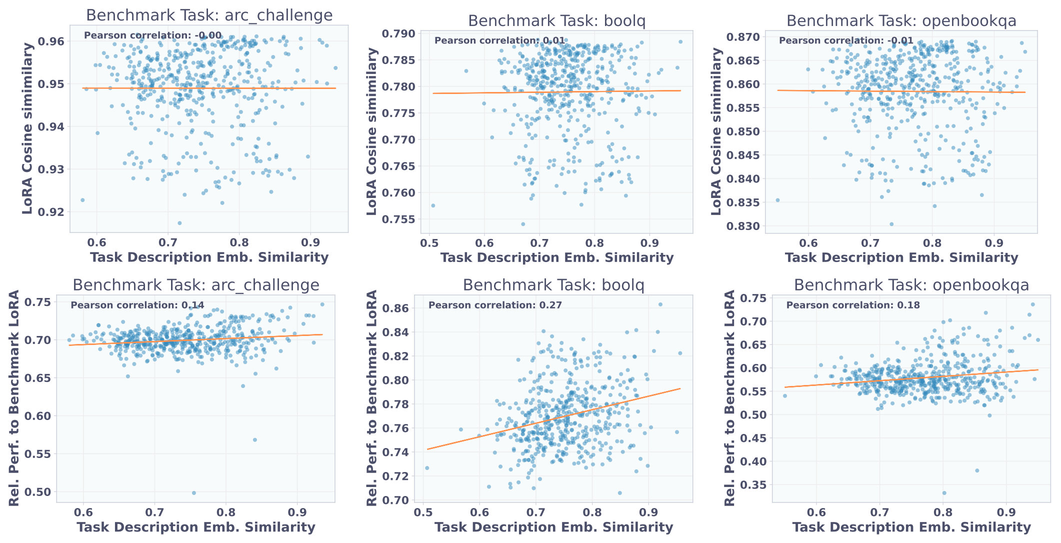


Figure 4: **Top row:** Each plot shows the similarity between a benchmark LoRA adapter and 489 SNI-trained adapters in the weight space (y-axis) against their similarity in the task embedding space (x-axis). **Bottom row:** Each plot shows SNI-trained adapters' performance relative to a benchmark adapter (y-axis) with the same x-axis. We can see that LoRAs with similar description embeddings to the benchmarks' perform better in those benchmarks, suggesting their shared functionalities. However, LoRAs with similar functionalities are not nearby in the parameter space.

Here, we investigate the relationship between LoRA adapters by inspecting their similarity in the parameter space, performance on the benchmarks, and similarity of their description embeddings. To measure adapter similarity, we compute the cosine similarity of the concatenation of flattened low-rank $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices of all layers. We plot the adapters' similarity against task description similarity (using the mean embedding of each task) in the top row of Fig. 4. We find no correlation between the cosine similarity of the adapters' weights (y-axis) and the task embedding similarity (x-axis) indicated by near-zero Pearson correlation coefficients.

In the bottom row of Fig. 4, we change the y-axis to adapters' relative benchmark performance to benchmark-specific adapters. We find a positive correlation between the relative benchmark performance of SNI-trained adapters and the task embedding similarity. That is, adapters perform better on a benchmark if their task descriptions are similar to those of the benchmark. However, despite their similar functionalities, adapters with similar descriptions are not similar in the parameter space. We believe that this relationship has a significant impact on the limited generalization of reconstruction-trained **HyperLoRA**. We further discuss this topic in Appendix E.

5.2 Visualization of **HyperLoRA** Activations

Next, we aim to understand **HyperLoRA** further and see whether it generates task-specific LoRA adapters for unseen tasks with unseen descriptions. We probe our best performing model in the zero-shot evaluation, SFT **HyperLoRA** **M** trained on 256 training tasks. We probe the model on all the benchmark tasks, each with three unseen descriptions. Fig. 5 shows the 2D t-SNE projection of **HyperLoRA**'s task encoder activation and the output of the last MLP block. We can see clear clustering in both projection plots based on the benchmark tasks (colors). This means that **HyperLoRA** generates different adapters for different tasks, confirming that **HyperLoRA** indeed generates task-specific adapters. Moreover, similar tasks, e.g., MBPP and HumanEval, are clustered together in both plots, suggesting that SFT-trained **HyperLoRA** produces similar adapters for semantically similar tasks.

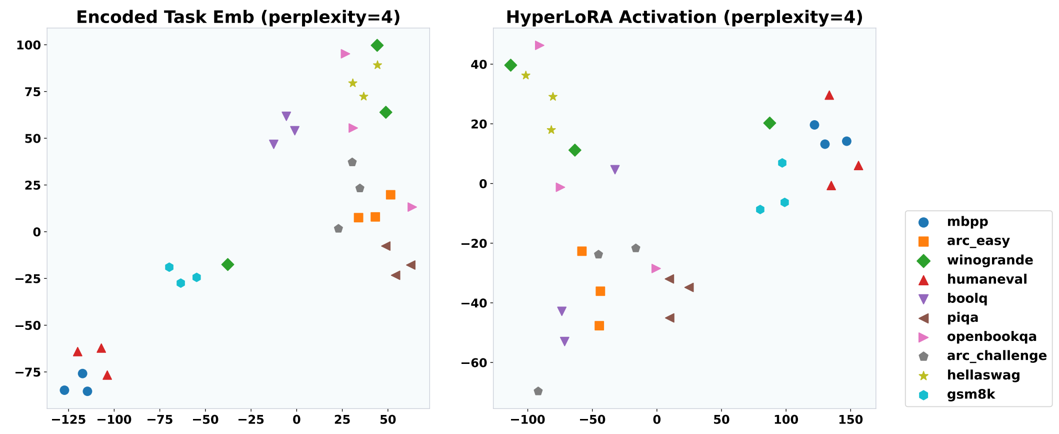


Figure 5: 2D t-SNE projection of activations of **HyperLoRA**'s task encoder (left) and activations of the last MLP block (right) grouped by benchmark tasks (represented by colors). We probe **HyperLoRA** with unseen three task descriptions per benchmark. We can see activations clustering in both plots, indicating that **HyperLoRA** indeed learns to generate LoRAs tailored to specific tasks.

6 Related Work

Hypernetworks for Adaptation: Hypernetworks (Ha et al., 2016) provide a general indirect encoding method for neural network weights. They have been applied to different architectures (e.g., in attention, Schug et al., 2024) and training paradigms (e.g., in continual learning, Von Oswald et al., 2019). Here, we focus on generating low-rank adapters using natural language instruction. Previous work (Mahabadi et al., 2021; He et al., 2022; Ortiz-Barajas et al., 2024) considers hypernetworks for LLM adaptation in a multi-task context but uses learned task identifiers instead of natural language for adaptation. Thus, these approaches do not enable task-wise zero-shot generalization.

Hypernetworks for Zero-Shot LLM Adaptation: Xiao et al. (2023) explore the use of HyperLoRA on a limited set of English dialects; they only consider five dialects, one of which is unseen. Plus, the hypernetwork relies on expert-based transformation of the dialects, limiting the possibility of generalization. Closely related to our work is Hyperdecoders (Ivison & Peters, 2022): a hypernetwork that generates adapters on the fly based on the input sequence. While per-sequence adaptation is desirable for benchmark evaluation—where the LLM should always output the correct answer—we argue that description-based adaptation gives more control to users since they can steer the LLM in creative ways based on user-generated descriptions. Our work improves upon Xiao et al. (2023) and Ivison & Peters (2022) in many ways, including achieving task-wise zero-shot generalization, simpler and more general hypernetwork input requirement, and more comprehensive experiments, ablations and analyses. Concurrent to our work, Lv et al. (2024) propose a similar approach that utilizes a hypernetwork to generate LoRA adapter at inference time. However, their hypernetwork assumes that the context vector provided to the hypernetwork contains few-shot examples. In contrast, **HyperLoRA** only assumes a task description, which users can produce by themselves within minutes.

7 Limitations and Future Work

Limitations. Our results have primarily focused on modulating the Mistral architecture, with an emphasis on LLM adaptation. However, no inherent constraints are preventing us from applying **HyperLoRA** to other LLMs or to adapt vision language models. Additionally, we believe the compression achieved by **HyperLoRA** can be further optimized. Finally, the potential for **HyperLoRA** trained on a smaller base model to transfer to effectively to larger models within the same architecture class remains an open area for exploration.

Future Work. We plan to expand **HyperLoRA** across multiple architectures and explore the potential for transfer between diverse systems. In addition, we intend to rigorously study the scaling laws of LoRA compression as the number of **HyperLoRA** parameters increases. Finally, our goal is to provide an openly accessible service for various **HyperLoRA** configurations. We envision a use-friendly platform where individuals could generate and download fine-tuned adapters by simply prompting a model with a minimal chat interface.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Rickard Brüel-Gabrielsson, Jiacheng Zhu, Onkar Bhardwaj, Leshem Choshen, Kristjan Greenewald, Mikhail Yurochkin, and Justin Solomon. Compress then serve: Serving thousands of lora adapters with little overhead. *arXiv preprint arXiv:2407.00066*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

Kathleen B Digre and KC Brennan. Shedding light on photophobia. *Journal of Neuro-ophthalmology*, 32(1):68–81, 2012.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

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

Yun He, Steven Zheng, Yi Tay, Jai Gupta, Yu Du, Vamsi Aribandi, Zhe Zhao, YaGuang Li, Zhao Chen, Donald Metzler, et al. Hyperprompt: Prompt-based task-conditioning of transformers. In *International conference on machine learning*, pp. 8678–8690. PMLR, 2022.

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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Hamish Ivison and Matthew E Peters. Hyperdecoders: Instance-specific decoders for multi-task nlp. *arXiv preprint arXiv:2203.08304*, 2022.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Sakaguchi Keisuke, Le Bras Ronan, Bhagavatula Chandra, and Choi Yejin. Winogrande: An adversarial winograd schema challenge at scale. 2019.

Hwichan Kim, Shota Sasaki, Sho Hoshino, and Ukyo Honda. A single linear layer yields task-adapted low-rank matrices. *arXiv preprint arXiv:2403.14946*, 2024.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

Chuancheng Lv, Lei Li, Shitou Zhang, Gang Chen, Fanchao Qi, Ningyu Zhang, and Hai-Tao Zheng. HyperloRA: Efficient cross-task generalization via constrained low-rank adapters generation. In *ACL Findings 2024*, 2024. URL https://openreview.net/forum?id=xa4GYUSvhW.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Jesus-German Ortiz-Barajas, Helena Gomez-Adorno, and Thamar Solorio. Hyperloader: Integrating hypernetwork-based lora and adapter layers into multi-task transformers for sequence labelling. *arXiv preprint arXiv:2407.01411*, 2024.

Oleksiy Ostapenko, Zhan Su, Edoardo Maria Ponti, Laurent Charlin, Nicolas Le Roux, Matheus Pereira, Lucas Caccia, and Alessandro Sordoni. Towards modular llms by building and reusing a library of loras. *arXiv preprint arXiv:2405.11157*, 2024.

Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

Simon Schug, Seijin Kobayashi, Yassir Akram, João Sacramento, and Razvan Pascanu. Attention as a hypernetwork. *arXiv preprint arXiv:2406.05816*, 2024.

Kenneth O Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial life*, 9 (2):93–130, 2003.

Yi Tay, Mostafa Dehghani, Jinfeng Rao, William Fedus, Samira Abnar, Hyung Won Chung, Sharan Narang, Dani Yogatama, Ashish Vaswani, and Donald Metzler. Scale efficiently: Insights from pre-training and fine-tuning transformers. *arXiv preprint arXiv:2109.10686*, 2021.

Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

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*, pp. 5085–5109, 2022.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Robert H Wurtz, Wilsaan M Joiner, and Rebecca A Berman. Neuronal mechanisms for visual stability: progress and problems. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 366(1564):492–503, 2011.

Zedian Xiao, William Held, Yanchen Liu, and Diyi Yang. Task-agnostic low-rank adapters for unseen English dialects. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7857–7870, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.487. URL https://aclanthology.org/2023.emnlp-main.487.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, et al. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. *arXiv preprint arXiv:2407.19669*, 2024.

Justin Zhao, Timothy Wang, Wael Abid, Geoffrey Angus, Arnav Garg, Jeffery Kinnison, Alex Sherstinsky, Piero Molino, Travis Addair, and Devvret Rishi. Lora land: 310 fine-tuned llms that rival gpt-4, a technical report. *arXiv preprint arXiv:2405.00732*, 2024.

A Ablations

A.1 Scaling Number of Training Tasks

Table 3: Benchmark performance of SFT-trained **HyperLoRA** with varying numbers of training tasks. Due to the space constraint, we show results with $\{128, 256, 489\}$ tasks. ▲ (▼) indicates increased (decreased) performance compared to the previous increment in the number of training tasks.

	Number of tasks	ArcC (acc)	ArcE (acc)	BQ (acc)	GSM8K (acc)	HS (acc)	OQA (acc)	PIQA (acc)	WG (acc)	HE (pass@1)	MBPP (pass@1)	Avg.
	128	75.4	88.0	85.0	43.6	66.7	74.7	82.2	57.7	38.0	51.4	66.3 ▲
HyperLoRA (SFT) L	256	74.0	87.9	85.1	46.2	65.7	73.8	82.0	61.7	39.8	53.2	66.9 ▲
	489	74.1	87.6	84.9	44.9	65.6	69.9	81.4	59.1	40.7	52.5	66.1 ▼
	128	74.2	87.7	83.6	46.9	65.4	71.1	81.4	56.3	40.7	53.3	66.1 ▲
HyperLoRA (SFT) M	256	74.9	88.0	84.9	47.7	65.6	73.9	82.0	62.1	40.2	52.7	67.2 ▲
	489	74.7	87.8	85.4	46.9	65.7	72.5	81.2	61.5	40.9	53.6	67.0 ▼
	128	73.2	86.5	84.6	45.8	65.6	69.5	81.4	59.2	42.9	51.4	66.0 ▲
HyperLoRA (SFT) S	256	74.1	86.9	84.3	45.5	64.9	69.9	80.8	60.5	38.6	52.1	65.8 ▼
	489	73.2	86.7	85.1	45.0	64.4	66.1	80.1	59.4	37.4	52.0	64.9 ▼

We study the impact of the number of training tasks on the zero-shot benchmark performance of **HyperLoRA** in the SFT setting, where all **HyperLoRA** instances are trained for the same number of gradient steps (see details in Appendix D). We find that increasing the number of training tasks improves the average zero-shot benchmark performance of the hypernetwork (Fig. 1 and Table 3). However, after 256 tasks (128 tasks for S), we see a consistent performance drop on all variants. We hypothesize that hypernetworks have a certain capacity for storing task knowledge. Thus, training hypernetworks on more tasks beyond their capacities causes performance degradation. This also explains why the smallest architecture S experiences a performance drop before other variants.

A.2 Task Embedding Models

Table 4 shows the zero-shot benchmark performance with two different embedding models: `gte-large-en-v1.5` and `Mistral-7B-Instruct`. For the `gte` model, we represent a task description with the activation of the `CLS` token in the last layer (1024D) as the model is a bidirectional model. For `Mistral`, we use the activation of the last token in the sequence (4096D) to represent a given description. Table 4 shows the results with the two embedding models used for **HyperLoRA** SFT training on 489 tasks. We find that both embedding models yield **HyperLoRA** instances with comparable generalization capability (66.0 vs 66.3), suggesting **HyperLoRA** robustness to specific text embedding methods as long as the embeddings contain some semantic information of the tasks.

Table 4: Zero-shot benchmark performance of **HyperLoRA** trained via SFT on 489 tasks.

	`gte`			`Mistral`		
	S	M	L	S	M	L
Avg. Benchmark performance	64.9	67.0	66.1	65.4	66.9	66.5
Avg.		**66.0**			**66.3**	

A.3 Prediction Offset

We ablate the inclusion of the MT LoRA prediction offset in **HyperLoRA**. We train **HyperLoRA** L with 256 SNI training datasets using three different offset values: MT LoRA, Avg. LoRA, and not using offset. Fig. 6 shows the performance differences. We find that using either MT LoRA or Avg. LoRA improves for the zero-shot performance equally, despite the worse benchmark performance of Avg. LoRA (see Table 2). Because we apply the offset to low-rank matrices as opposed to the full adapter matrix in each layer, we hypothesize that the offset serves as an adapter "basis", which eases the learning process. We discuss this topic further in Appendix E.

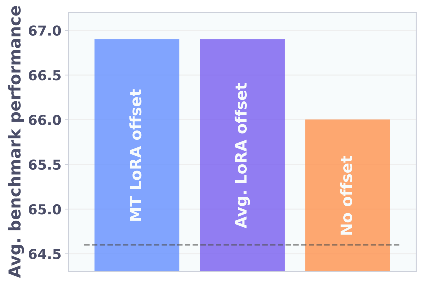


Figure 6: Impact of prediction offsets.

A.4 Varying Task Descriptions

We investigate the impact of input task descriptions on the performance of generated LoRAs. We use four types of task descriptions:

- **Train:** Training descriptions
- **Eval:** Unseen descriptions
- **Random strings:** Random literal strings
- **Train (random):** Training descriptions randomly sampled from other benchmarks

Table 5: Benchmark performance of **HyperLoRA** trained via reconstruction on 9 benchmark tasks.

	Aligned		Unaligned	
	Train	Eval	Train (random)	Random strings
HyperLoRA **L**	73.3	73.6	49.1	68.2
HyperLoRA **M**	73.5	70.2	49.5	68.5
HyperLoRA **S**	73.0	72.9	55.7	53.9
Avg.	**73.3**	**72.2**	**51.4**	**63.5**

For each description type, we report the average performance using three descriptions. The four description types can be grouped into two categories based on the alignment between the descriptions and the tasks: aligned (**Train**, **Eval**) and unaligned (**Train (random)** and **Random strings**). We can see a clear performance gap between the two description categories. Specifically, training and evaluation descriptions generate the best performing LoRAs, matching the performance of oracle LoRAs, despite the evaluation descriptions being unseen. These results suggest that **HyperLoRA** are robust to changes in the task description as long as the descriptions are aligned with the task. On the other hand, if the descriptions are not aligned with the task at hand, the generated LoRAs will not perform as well, as indicated by the performance of the unaligned group.

A.5 Training Schemes

In this section, we investigate the zero-shot performance of SFT-trained and reconstruction-trained **HyperLoRA**. All model instances are trained with roughly equal wall-clock time of 10 hours (see Appendix D for details). From Table 6, we can see that there is a clear performance gap between reconstruction and SFT training schemes. Specifically, SFT produces **HyperLoRA** instances that perform significantly better than that of reconstruction training (66.63 vs 61.83 benchmark performance averaged over model architectures). We attribute the performance difference to the library of LoRAs needed for reconstruction training. For reconstruction-trained **HyperLoRA** to generalize, the target LoRA adapters of similar tasks should be clustered in some latent manifold. In contrast, SFT training does not need pre-trained task-specific LoRA adapters, thus sidestepping this problem by learning end-to-end. In the next section, we show that adapters of similar tasks do not live nearby in the weight space (Section 5.1), supporting our claim of a potential problem when reconstructing pre-trained LoRA adapters.

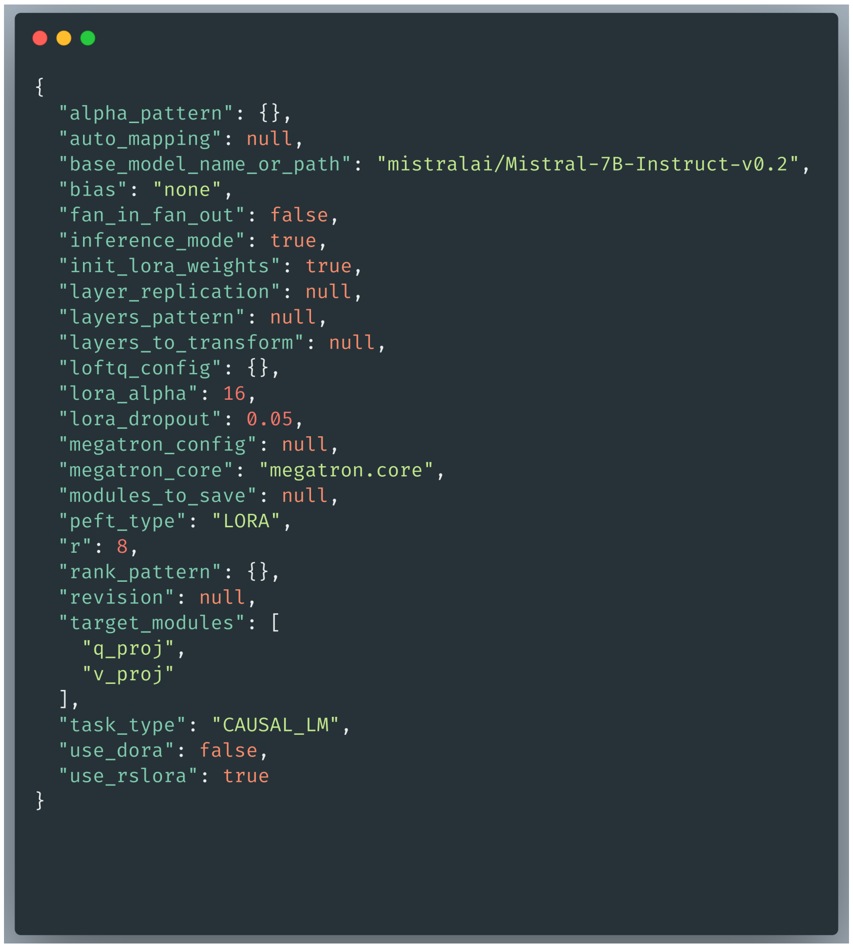
Table 6: Zero-shot benchmark performance of **HyperLoRA** trained via reconstruction and SFT.

	Recon			SFT		
	S	**M**	**L**	**S**	**M**	**L**
Benchmark performance	61.8	61.7	62.0	65.8	67.2	66.9
Avg.		**61.8**			**66.6**	

B Hyperparameter Settings

Table 7: Hyperparameters for training a task-specific LoRA adapter.

Hyperparameters	Task-specific LoRA	HyperLoRA (SFT)	HyperLoRA (recon)
Batch size	4	4	Number of the target LoRAs
Epochs	10	80 for 16 tasks 40 for 32 tasks 20 for 64 tasks 10 for 128 tasks 5 for 256 tasks 3 for 489 tasks	100000
Gradient accumulation steps	2	16	1
Max learning rate	8×10^{-5}	10^{-4}	10^{-3}
Max gradient norm	1.0	1.0	1.0
NEFTune noise alpha	5.0	5.0	5.0
Warmup fraction	0.1	0.1	0.1
Learning rate scheduler	Linear with warm up	Linear with warm up	Linear with warm up



```
{
  "alpha_pattern": {},
  "auto_mapping": null,
  "base_model_name_or_path": "mistralai/Mistral-7B-Instruct-v0.2",
  "bias": "none",
  "fan_in_fan_out": false,
  "inference_mode": true,
  "init_lora_weights": true,
  "layer_replication": null,
  "layers_pattern": null,
  "layers_to_transform": null,
  "loftq_config": {},
  "lora_alpha": 16,
  "lora_dropout": 0.05,
  "megatron_config": null,
  "megatron_core": "megatron.core",
  "modules_to_save": null,
  "peft_type": "LORA",
  "r": 8,
  "rank_pattern": {},
  "revision": null,
  "target_modules": [
    "q_proj",
    "v_proj"
  ],
  "task_type": "CAUSAL_LM",
  "use_dora": false,
  "use_rslora": true
}
```

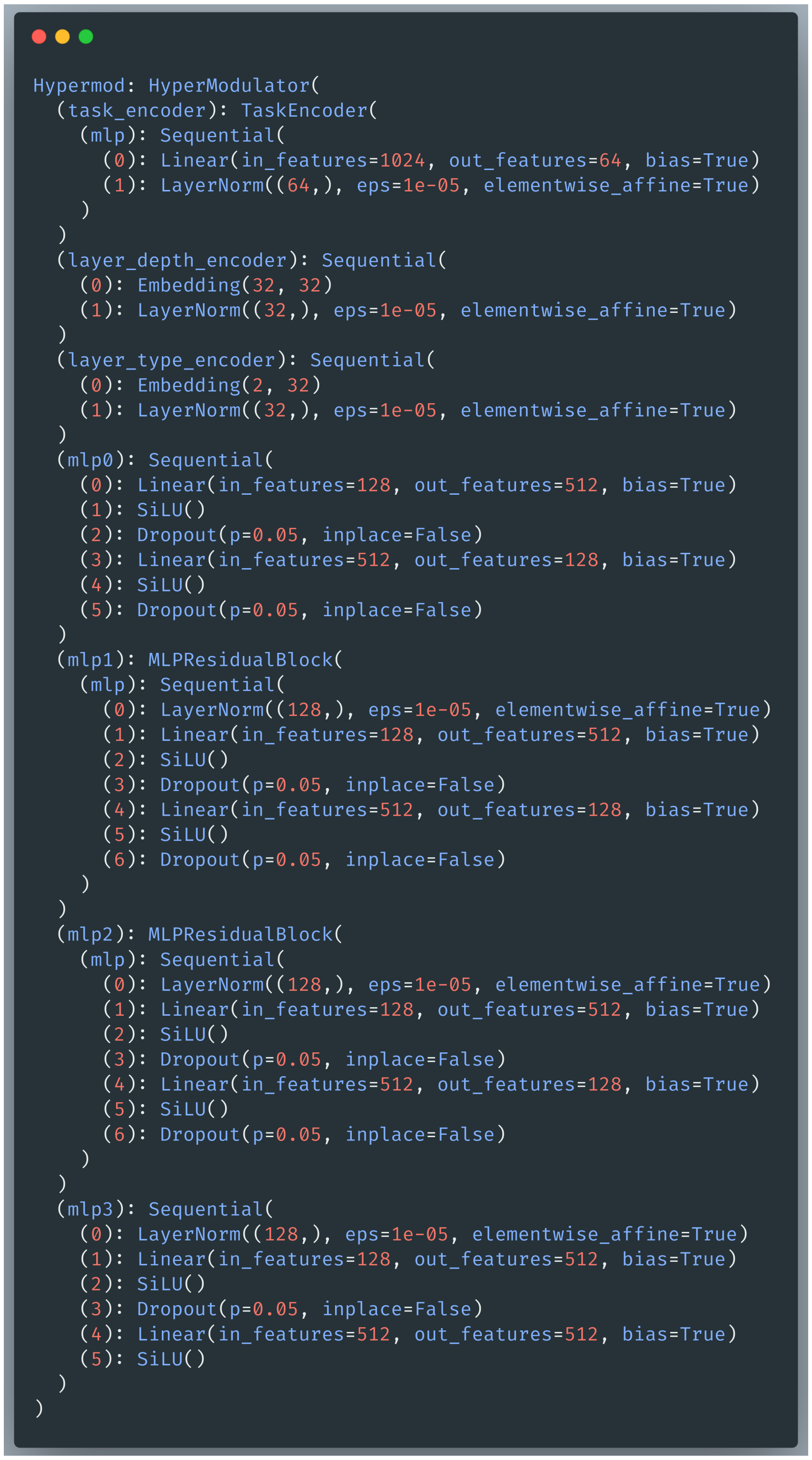
Figure 7: Parameter-efficient fine-tuning (PEFT) config for all LoRA adapters.

Table 7 and Fig. 7 show the training configuration of all models trained in this work. For LoRA reconstruction training, each prediction target is an entirety of a LoRA adapter. That is, there is a total of 489 training samples for 489 SNI tasks. Thus, we increase the epochs to $100,000$ to ensure that the **HyperLoRA** converges.

C Additional Details of HyperLoRA Architectures

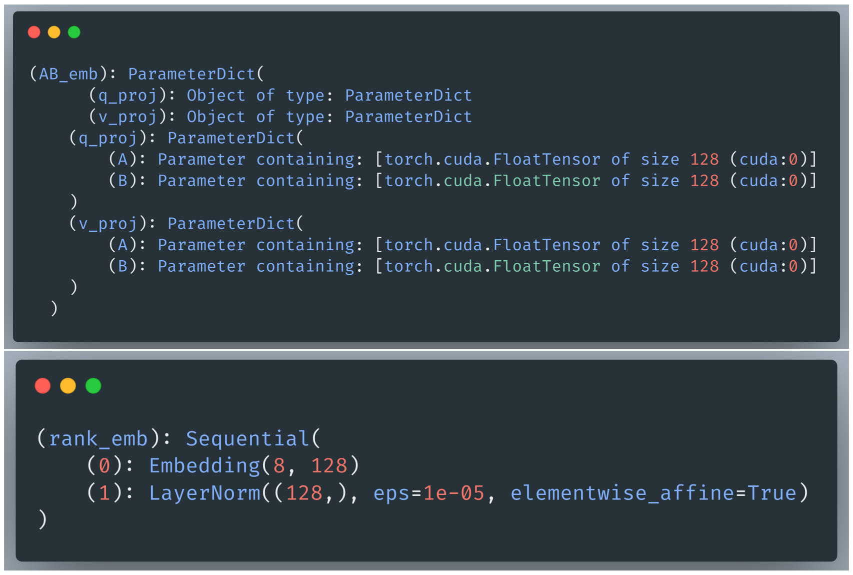
Figs. 8 and 9 shows the details of the backbone of **HyperLoRA**. Specifically, the size of the module and layer embedding ($E[m]$ and $E[l]$) is 32D. Together, they form a dictionary of 34 learnable embeddings (32 layers + 2 target modules). The task encoder is a linear layer that takes in a text embedding (1024D for the `gte` embedding and 4096D for `Mistral` embedding) and outputs a 64D vector. The encoded task, module, and layer embedding are concatenated and then fed into `mlp0` followed by a residual MLP block `mlp1`. At this point, for **M** and **S**, we add a 128D A/B embbedding to the residual stream. The output is then fed to another residual MLP block `mlp2`. At this point, for **S**, we add a 128D rank embedding to the residual stream. After this, we feed the activation to the last MLP block. The output of the last MLP block is then fed to a linear head, whose output size is as follows:

- **L**: $2 \times r \times d$ giving both $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices
- **M**: $r \times d$ giving a low-rank matrix $\boldsymbol{A}$ or $\boldsymbol{B}$ depending on the A/B embedding
- **S**: d giving a rank of a low-rank matrix depending on both the A/B embedding and the rank embedding.



```
Hypermod: HyperModulator(
  (task_encoder): TaskEncoder(
    (mlp): Sequential(
      (0): Linear(in_features=1024, out_features=64, bias=True)
      (1): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
    )
  )
  (layer_depth_encoder): Sequential(
    (0): Embedding(32, 32)
    (1): LayerNorm((32,), eps=1e-05, elementwise_affine=True)
  )
  (layer_type_encoder): Sequential(
    (0): Embedding(2, 32)
    (1): LayerNorm((32,), eps=1e-05, elementwise_affine=True)
  )
  (mlp0): Sequential(
    (0): Linear(in_features=128, out_features=512, bias=True)
    (1): SiLU()
    (2): Dropout(p=0.05, inplace=False)
    (3): Linear(in_features=512, out_features=128, bias=True)
    (4): SiLU()
    (5): Dropout(p=0.05, inplace=False)
  )
  (mlp1): MLPResidualBlock(
    (mlp): Sequential(
      (0): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
      (1): Linear(in_features=128, out_features=512, bias=True)
      (2): SiLU()
      (3): Dropout(p=0.05, inplace=False)
      (4): Linear(in_features=512, out_features=128, bias=True)
      (5): SiLU()
      (6): Dropout(p=0.05, inplace=False)
    )
  )
  (mlp2): MLPResidualBlock(
    (mlp): Sequential(
      (0): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
      (1): Linear(in_features=128, out_features=512, bias=True)
      (2): SiLU()
      (3): Dropout(p=0.05, inplace=False)
      (4): Linear(in_features=512, out_features=128, bias=True)
      (5): SiLU()
      (6): Dropout(p=0.05, inplace=False)
    )
  )
  (mlp3): Sequential(
    (0): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
    (1): Linear(in_features=128, out_features=512, bias=True)
    (2): SiLU()
    (3): Dropout(p=0.05, inplace=False)
    (4): Linear(in_features=512, out_features=512, bias=True)
    (5): SiLU()
  )
)
```

Figure 8: Detailed backbone architecture.



```
(AB_emb): ParameterDict(
      (q_proj): Object of type: ParameterDict
      (v_proj): Object of type: ParameterDict
    (q_proj): ParameterDict(
        (A): Parameter containing: [torch.cuda.FloatTensor of size 128 (cuda:0)]
        (B): Parameter containing: [torch.cuda.FloatTensor of size 128 (cuda:0)]
    )
    (v_proj): ParameterDict(
        (A): Parameter containing: [torch.cuda.FloatTensor of size 128 (cuda:0)]
        (B): Parameter containing: [torch.cuda.FloatTensor of size 128 (cuda:0)]
    )
  )
```

```
(rank_emb): Sequential(
    (0): Embedding(8, 128)
    (1): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
)
```

Figure 9: Detailed A/B and rank embedding of **`HyperLoRA`**.

For ease of explanation, we assume d is the same for the input and the output space of a linear transformation. In practice, $d_{\text{in}} = d_{\text{out}} = 4096$ for `q_proj` module and $d_{\text{in}} = 4096, d_{\text{out}} = 1024$ for `v_proj` module. $r = 8$ for all adapters in this work. Finally, we list the number of trainable parameters of each architecture: $55,252,992$ for **L**, $34,282,240$ for **M**, $4,923,392$ for **S**, $3,407,872$ for LoRA.

D Training Details

All models trained in this work fit in a single H100 GPU (80GB of VRAM). Notably, SFT requires much more memory because of the need to backpropagate the gradient through the base LLM. Reconstruction training, on the other hand, should be possible in a modern consumer-grade GPU.

For reconstruction training, we fix the training epochs to be 100K but scale the batch size to match the number of target LoRA adapters. This means the model trains much faster for a lower number of target LoRAs while maintaining the same number of optimizer steps. For reference, training to reconstruct 9 benchmark-specific LoRAs takes around 10 minutes to complete, while training to reconstruct 489 SNI LoRA adapters takes around 10 hours.

For SFT training, we aim to keep the number of optimizer steps the same as we do for reconstruction training. However, since we cannot fit all fine-tuning samples, we scale the number of epochs inverse to the number of training tasks (see Table 7).

Additionally, for reconstruction training, instead of predicting the weights directly, **`HyperLoRA`** learns to predict the z-score of a normal distribution of each weight entry in the low-rank $\boldsymbol{A}, \boldsymbol{B}$ matrices. At test time, the output is multiplied by the standard deviation of each element before adding to the mean, converting the prediction to the correct scale.

E Utilizing Full Adaptation Matrix vs Low-Rank Matrices

Similar to Fig. 4, Fig. 10 show the similarity of SNI adapters to benchmark-specific adapters, but instead of using the concatenation of flattened $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices, we use flattened $\Delta \boldsymbol{W}$ instead. With the change, we find a positive correlation between the task embedding similarity and the adapter similarity in the weight space. This is likely because, for a given $\Delta \boldsymbol{W}$ matrix, there are many possible permutations of low-rank matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. This suggests that if we compute the reconstruction loss in the full adaptation matrix space, reconstruction-trained **`HyperLoRA`** could generalize better.

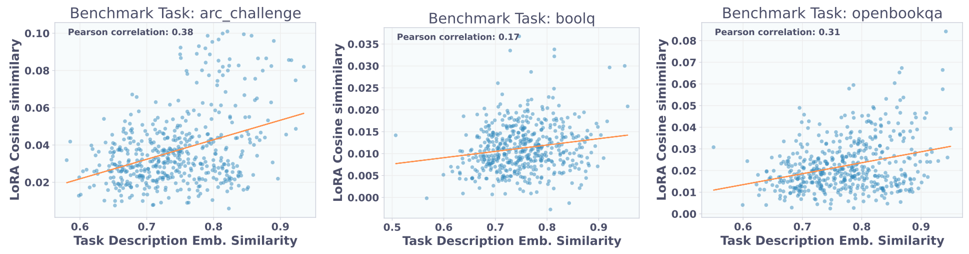


Figure 10: Each plot shows the similarity between a benchmark LoRA adapter and 489 SNI-trained adapters in the $\Delta \boldsymbol{W}$ weight space. There is a positive correlation between the two variables indicated by small positive Pearson correlation coefficients.

However, we empirically find that it does not outperform **HyperLoRA** trained to reconstruct low-rank matrices at zero-shot LoRA generation.

Furthermore, we also ablate how the prediction offset is applied to **HyperLoRA** under SFT training. First, the offset could be added to the low-rank matrices:

$$\begin{aligned} \Delta \boldsymbol{W} &= (\boldsymbol{B}_{\text{pred}} + \boldsymbol{B}_{\text{offset}})^T (\boldsymbol{A}_{\text{pred}} + \boldsymbol{A}_{\text{offset}}) && (6) \\ &= \boldsymbol{B}_{\text{pred}}^T \boldsymbol{A}_{\text{pred}} + \boldsymbol{B}_{\text{pred}}^T \boldsymbol{A}_{\text{offset}} + \boldsymbol{B}_{\text{offset}}^T \boldsymbol{A}_{\text{pred}} + \boldsymbol{B}_{\text{offset}}^T \boldsymbol{A}_{\text{offset}}. && (7) \end{aligned}$$

The other approach is to apply the offset to the full adaptation matrix:

$$\Delta \boldsymbol{W} = \boldsymbol{B}_{\text{pred}}^T \boldsymbol{A}_{\text{pred}} + \boldsymbol{B}_{\text{offset}}^T \boldsymbol{A}_{\text{offset}} \quad (8)$$

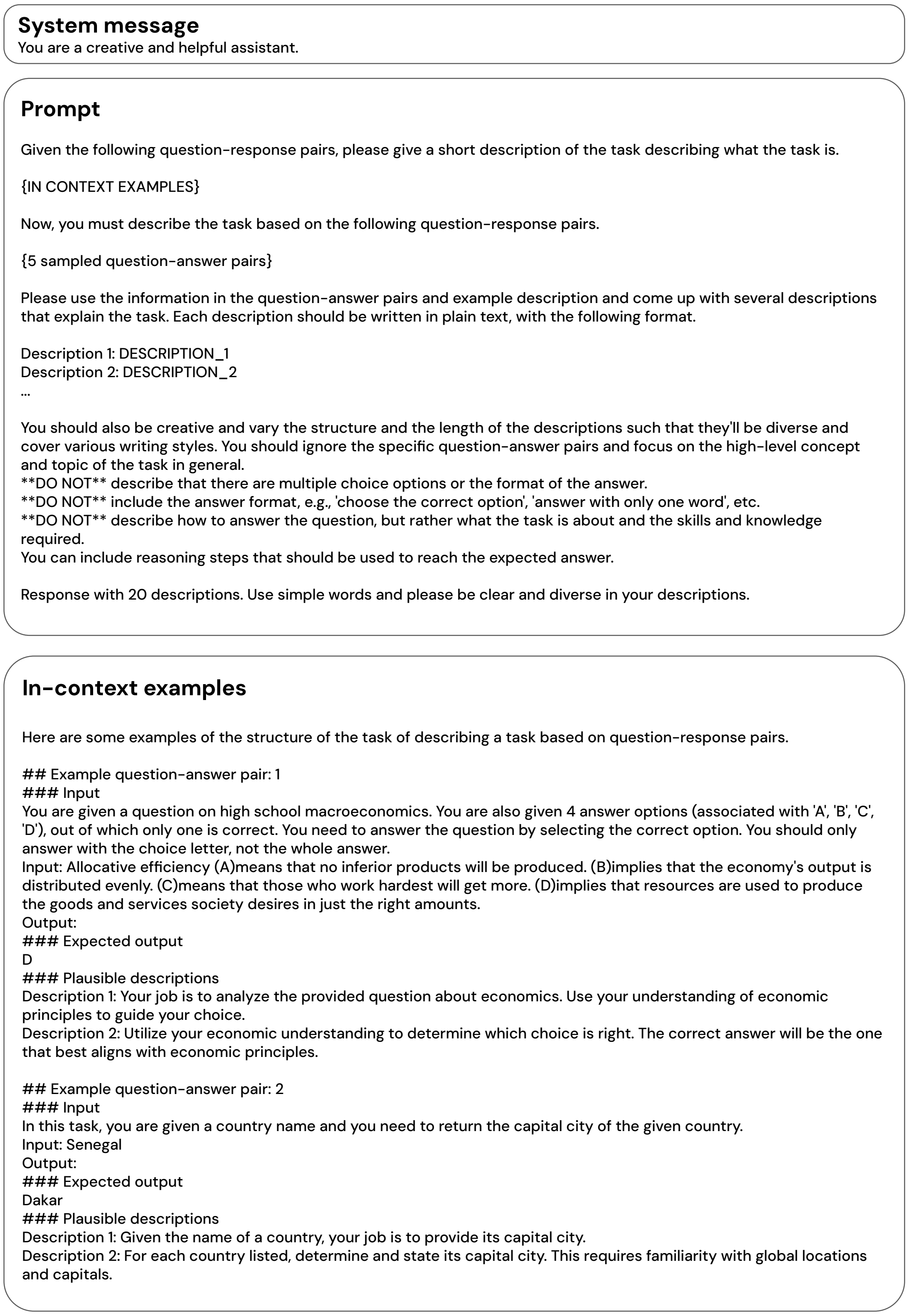
In a preliminary experiment, we find that applying the offset to the low-rank matrices performs better than the alternative for SFT training. We hypothesize that the cross terms (the middle two terms in Eq. (7)) ease the learning process. Effectively, the offset matrices in the cross terms act as ‘answer bases‘ for $\boldsymbol{A}_{\text{pred}}$ and $\boldsymbol{B}_{\text{pred}}$ to act upon.

F Generating Task Descriptions with a Foundation Language Model

We automate task description generation for each task by leveraging powerful closed-source language models (Achiam et al., 2023). We query `GPT-4o mini` with carefully constructed prompts that incentivize diversity to facilitate downstream generalization. In particular, we generate 200 descriptions per task by querying the model 10 times, each time asking for 20 descriptions given randomly sampled five question-answer pairs from the task. We leverage in-context learning by providing examples of question-answer pairs with matching descriptions. Finally, we also designed our prompts to avoid overly verbose responses and unnecessary information, such as explicit mentions of answer formats and additional instructions. We use the generated descriptions for the training and benchmark tasks.

G Scaling Number of Descriptions per Task

Fig. 12 shows mixed results on the benchmark performance when vary the number of descriptions per training task. For consistency, we always train **HyperLoRA** with 128 descriptions per training task.



System message
You are a creative and helpful assistant.

Prompt

Given the following question-response pairs, please give a short description of the task describing what the task is.

{IN CONTEXT EXAMPLES}

Now, you must describe the task based on the following question-response pairs.

{5 sampled question-answer pairs}

Please use the information in the question-answer pairs and example description and come up with several descriptions that explain the task. Each description should be written in plain text, with the following format.

Description 1: DESCRIPTION_1
Description 2: DESCRIPTION_2
...

You should also be creative and vary the structure and the length of the descriptions such that they'll be diverse and cover various writing styles. You should ignore the specific question-answer pairs and focus on the high-level concept and topic of the task in general.
DO NOT describe that there are multiple choice options or the format of the answer.
DO NOT include the answer format, e.g., 'choose the correct option', 'answer with only one word', etc.
DO NOT describe how to answer the question, but rather what the task is about and the skills and knowledge required.
You can include reasoning steps that should be used to reach the expected answer.

Response with 20 descriptions. Use simple words and please be clear and diverse in your descriptions.

In-context examples

Here are some examples of the structure of the task of describing a task based on question-response pairs.

Example question-answer pair: 1
Input
You are given a question on high school macroeconomics. You are also given 4 answer options (associated with 'A', 'B', 'C', 'D'), out of which only one is correct. You need to answer the question by selecting the correct option. You should only answer with the choice letter, not the whole answer.
Input: Allocative efficiency (A)means that no inferior products will be produced. (B)implies that the economy's output is distributed evenly. (C)means that those who work hardest will get more. (D)implies that resources are used to produce the goods and services society desires in just the right amounts.
Output:
Expected output
D
Plausible descriptions
Description 1: Your job is to analyze the provided question about economics. Use your understanding of economic principles to guide your choice.
Description 2: Utilize your economic understanding to determine which choice is right. The correct answer will be the one that best aligns with economic principles.

Example question-answer pair: 2
Input
In this task, you are given a country name and you need to return the capital city of the given country.
Input: Senegal
Output:
Expected output
Dakar
Plausible descriptions
Description 1: Given the name of a country, your job is to provide its capital city.
Description 2: For each country listed, determine and state its capital city. This requires familiarity with global locations and capitals.

Figure 11: The prompt template used to query `GPT-4o mini` for task descriptions.

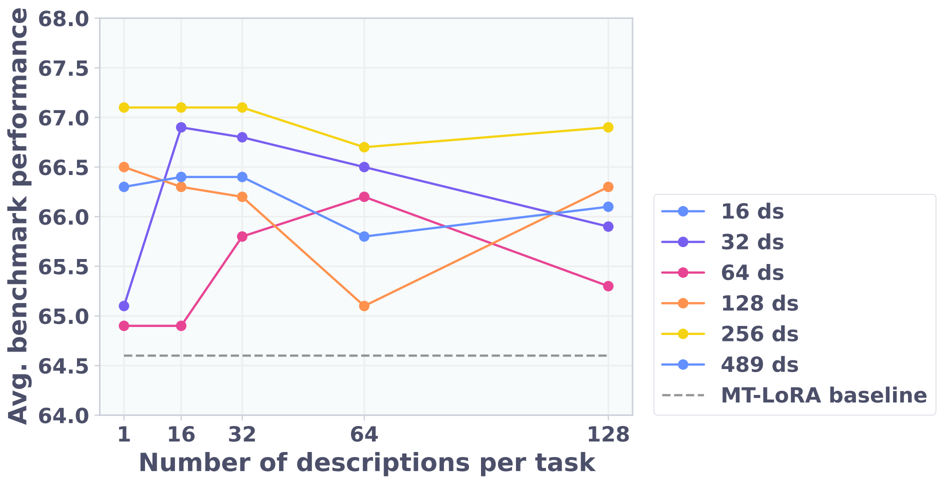


Figure 12: Zero-shot benchmark performance of SFT-trained `HyperLoRA` with varying number of descriptions per training task.