Test-Time Adaptation for LLMs via Hypernetwork-Generated LoRA Weights

Midterm Progress Report

Andrews George Varghese & Dhruv Kapur & Raj Maheshwari
{ageorgev, dkapur, rajmahes}@andrew.cmu.edu
10-423/623 Generative AI Course Project

December 11, 2025

Abstract

We investigate test-time adaptation of large language models through hypernetwork-generated LoRA weights that enable instance-level parameter specialization. Our approach extends recent work on task-level LoRA generation to produce unique adapter parameters for each input during inference, eliminating the need for separate per-task adapters. At the midterm checkpoint, we have successfully implemented: (1) a comprehensive evaluation framework supporting both chat and non-chat model formats across four diverse benchmarks (GSM8K, NLI, ARC-Easy and BoolQ) with flexibility to extend to more benchmarks, (2) baseline LoRA fine-tuning pipelines for three models spanning 70M to 600M parameters (Pythia 70M, Gemma3 270M Instruct and Qwen3 0.6B), and (3) an initial implementation of the hypernetwork that generates and injects LoRA weights into an LM. Preliminary baseline results demonstrate expected performance for the various model sizes and their dataset specific PEFT counterparts, providing clear targets for our hypernetwork approach. We outline remaining work including hypernetwork training, comparative evaluation across model scales, and analysis of generated adapter diversity.

1 Introduction

Parameter-efficient fine-tuning through Low-Rank Adaptation (LoRA) [Hu et al., 2022] has become the dominant paradigm for adapting large language models to specialized tasks, reducing trainable parameters by up to 10,000x while maintaining performance comparable to full fine-tuning. However, conventional LoRA approaches require training separate adapters for each task in advance, creating engineering overhead and preventing dynamic adaptation to individual inputs.

Recent work by Sakana AI [Charakorn et al., 2025] demonstrated that hypernetworks can generate task-specific LoRA adapters from natural language descriptions, achieving 98% of specialized adapter performance without task-specific training. Building on this foundation, we investigate **instance-level adaptation**: using hypernetworks to generate unique LoRA parameters for each input during inference rather than uniform task-level adapters.

This midterm report documents our progress toward this goal, including implementation of evaluation infrastructure across three model scales (Pythia 70M, Gemma 270M-Instruction Tuned, Qwen3 0.6B) and four diverse benchmarks (GSM8K, NLI, ARC-Easy, BoolQ), baseline establishment, and initial hypernetwork development. We present our experimental setup, preliminary results, and detailed plans for completing the project.

2 Dataset, Task, and Model Selection

2.1 Benchmark Suite

To enable comprehensive evaluation while managing computational constraints, we focus on four diverse benchmarks representing different reasoning and knowledge requirements:

Mathematical Reasoning: GSM8K. GSM8K Cobbe et al. [2021] provides 8,500 grade-school math word problems requiring multi-step arithmetic reasoning (2 to 8 steps per solution). This tests the model's ability to decompose problems and perform accurate numerical computations.

Natural Language Inference: SNLI. The Stanford Natural Language Inference (SNLI) dataset Bowman et al. [2015] contains 570,000 human-annotated sentence pairs labeled for entailment, contradiction, or neutral relationships. This requires the model to understand the 2 sentences and their relationship when seen as premise and hypothesis.

Science Reasoning: ARC-Easy. ARC-Easy Clark et al. [2018] contains 2,376 grade-school science questions that require reasoning beyond direct fact retrieval. This benchmark tests scientific knowledge in a multi-step reasoning setting.

Question Answering: BoolQ. The Boolean Questions Dataset (BoolQ) Clark et al. [2019] contains around 16,000 naturally occurring yes/no questions about Wikipedia passages, collected from real Google search queries. Each question is paired with a paragraph that contains the information needed to answer it, thus testing if the model can perform reading comprehension and binary classification.

These four benchmarks provide diversity across reasoning types (mathematical, logical, scientific, extractive), output formats (free-form generation for GSM8K, three-way classification for SNLI, multiple-choice for ARC-Easy, yes/no for BoolQ) while remaining tractable for our computational budget. More importantly, it acts as an orthogonal set of datasets over which we will train our hyper-network model. We aim to cover as many such orthogonal ideas during training while remaining within our computational budget, so that the hyper-network is able to generalize to unseen tasks.

2.2 Evaluation Metrics

We use task-appropriate metrics: match-with-tolerance accuracy for GSM8K (requiring correct numerical answers within some tolerance), and accuracy for SNLI (three-way classification), ARC-Easy (multiple-choice), and SQuAD (true/false).

2.3 Model Selection

Table 1: Model configurations for the three models evaluated in this work.

Model	d	n_{layers}	h_q/h_{kv}
Pythia 70M	512	6	8/8
Gemma 3 270M-IT	640	18	4/1
Qwen3 0.6B	1024	28	16/8

Rather than evaluating a single model, we adopt a multi-model approach to understand how instance-level adaptation scales across different model sizes and training paradigms. Our three models span an order of magnitude in parameter count and include both base and instruction-tuned variants:

Pythia 70M. The smallest model in our suite, Pythia 70M [Biderman et al., 2023] is a decoder-only transformer trained on the Pile dataset. As a base pretrained model without instruction tuning, it provides insight into whether instance-level adaptation can effectively specialize models with limited innate task-following capabilities. Its small size enables rapid iteration during hypernetwork development.

Gemma 3 270M Instruct. Gemma 3 270M-IT [Gemma, 2025] represents the smallest, yet reasonably intelligent, instruction-tuned model we could find. At approximately 4x the parameters of Pythia 70M, it tests whether instance-level adaptation provides benefits when the base model already possesses instruction-following capabilities, but may not perform too well on the benchmarks.

Qwen3-0.6B. Qwen3-0.6B is the largest model we are considering, at approximately 9x the size of Pythia 70M. This model is extremely intelligent given its small size, and it will be interesting to see if test-time augmentation can further improve its performance.

This multi-model strategy enables us to assess: (1) whether hypernetwork-generated adapters have a similar behavior on base and instruction-tuned models, (2) how adaptation benefits scale with model capacity, and (3) whether our approach generalizes across different model architectures.

3 Related Work

Our work builds upon three interconnected research areas: parameter-efficient fine-tuning, hypernetwork architectures, and meta-learning for fast adaptation.

Low-Rank Adaptation. LoRA [Hu et al., 2022] introduced parameter-efficient fine-tuning by representing weight updates as $\Delta W = BA$ where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ with rank $r \ll \min(d, k)$. This approach reduces trainable parameters by orders of magnitude while matching full fine-tuning performance, establishing the foundation for modern efficient adaptation methods.

Hypernetworks for Parameter Generation. The paradigm of using neural networks to generate weights for other networks was formalized by Ha et al. [2017], demonstrating end-to-end training via backpropagation. HyperFormer Mahabadi et al. [2021] extended this to transformers by generating adapter parameters conditioned on task embeddings, layer positions, and module types, achieving strong multi-task performance with minimal per-task parameters (0.29%).

Meta-Learning Foundations. Model-Agnostic Meta-Learning (MAML) [Finn et al., 2017] introduced bi-level optimization for rapid task adaptation through few gradient steps. While MAML requires gradient computation during adaptation, hypernetwork approaches generate parameters in a single forward pass, representing a complementary trade-off between adaptation quality and computational efficiency.

Task-Level LoRA Generation. Most directly relevant to our work, Text-to-LoRA [Charakorn et al., 2025] demonstrated that hypernetworks can generate complete LoRA adapters from natural language task descriptions, compressing hundreds of task-specific adapters into a single model. HyperDecoders Ivison and Peters [2022] proposed treating parameter generation as autoregressive sequence generation conditioned on instructions, though focused on encoder-decoder architectures rather than decoder-only language models.

Our Contribution. While Text-to-LoRA operates at task-level granularity (one adapter per task description), we investigate *instance-level adaptation* that generates unique LoRA parameters for each input. This eliminates the need for task labels or descriptions while enabling fine-grained specialization to individual examples, potentially improving performance on heterogeneous data within tasks.

4 Methods

4.1 Baseline: Task-Specific LoRA

Our primary baseline is standard task-specific LoRA fine-tuning, applied independently to each benchmark dataset. For each task, we attach LoRA adapters of rank $r = 2$, and $\alpha = 8$ to all linear projection layers in the transformer. This configuration provides a highly parameter-efficient setting while remaining expressive enough to capture task-specific behavior. LoRA is applied to the query and value projection matrices except for Pythia, which has a single projection matrix for queries, keys and values.

We train using the default **AdamW** optimizer, a learning rate of 5×10^{-5}, per-device batch size of 2 with gradient accumulation of 2 (effective batch size 4 using 1 device), and a training budget of 500 gradient steps. These hyperparameters were chosen to ensure stability under low-rank adaptation, reduce overfitting on small datasets.

Overall, this per-task LoRA fine-tuning baseline represents a strong and competitive parameter-efficient adaptation method. It serves as the performance baseline for evaluating our hypernetwork-generated LoRA approach, which must be within acceptable range of these independently optimized adapters while training only once.

4.2 Hypernetwork Architecture

Our hypernetwork generates LoRA weight matrices conditioned on individual input sequences. The architecture consists of three components:

Input Encoding. We pass the input prompt through the frozen base LLM and extract the final layer's last token representation as the semantic embedding, which is the input to the hypernetwork. This self-encoding design offers two key advantages: First, it simplifies the implementation by requiring only a single tokenizer and leveraging the HuggingFace Transformers API without additional encoder dependencies. Second, and more importantly, it provides a strong inductive bias—the hypernetwork predicts LoRA weights based on the same internal representations that the target model will process, supposedly creating an alignment between the encoding space and the weight space.

Conditioning Information. Following Text-to-LoRA [Charakorn et al., 2025], we incorporate learned embeddings for layer position, module identity (for example `q_proj`, `v_proj`, etc.) and LoRA matrix identity (A or B). For a target layer ℓ, module m and matrix $k \in \{A, B\}$:

$$\mathbf{c}_{\ell,m,k} = \mathbf{h}_{\text{input}} + \mathbf{e}^{(\ell)}_{\text{layer}} + \mathbf{e}^{(m)}_{\text{module}} + \mathbf{e}^{(k)}_{\text{matrix}}$$

where $\mathbf{e}_{\text{layer}} \in \mathbb{R}^{n_{\text{layers}} \times d_h}$, $\mathbf{e}_{\text{module}} \in \mathbb{R}^{n_{\text{modules}} \times d_h}$ and $\mathbf{e}_{\text{matrix}} \in \mathbb{R}^{2 \times d_h}$ are learned embeddings, with d_h being the internal hypernetwork dimension, n_{layers} the number of transformer blocks in the target LM and n_{module} the cardinality of the set of target modules to apply LoRA to.

Weight Generation. The conditioned representation passes through a shared MLP Ψ. Each pair of target module-matrix has its own head (owing to the fact that different modules may have different matrix dimensions), with a total of $n_{\text{modules}} \times 2$ heads. Each head produces the number of outputs required for the corresponding matrix in the corresponding module.

$$\mathbf{z}_{\ell,m,k} = \Psi(\mathbf{c}_{\ell,m,k}) \tag{1}$$

$$A_{\ell,m} = W_{m,A}\mathbf{z}_{\ell,m,A} + b_{m,A} \tag{2}$$

$$B_{\ell,m} = W_{m,B}\mathbf{z}_{\ell,m,B} + b_{m,B} \tag{3}$$

Weight Injection: The target linear layers in the LM are replaced with a custom `DynamicLoraLinear` layer that allows dynamic injection of the predicted A and B weights as needed.

4.3 Training Strategy

We train the hypernetwork end-to-end by backpropagating through the frozen base language model. For each training example (x, y):

1. Predict LoRA weights $\{B_{\ell,m}, A_{\ell,m}\}$ from input x
2. Inject generated adapter weights to the frozen base model
3. Compute language modeling loss $\mathcal{L}(y|x; \theta_{\text{base}}, \{B_{\ell,m}A_{\ell,m}\})$ and backpropagate gradients, updating only the hypernetwork

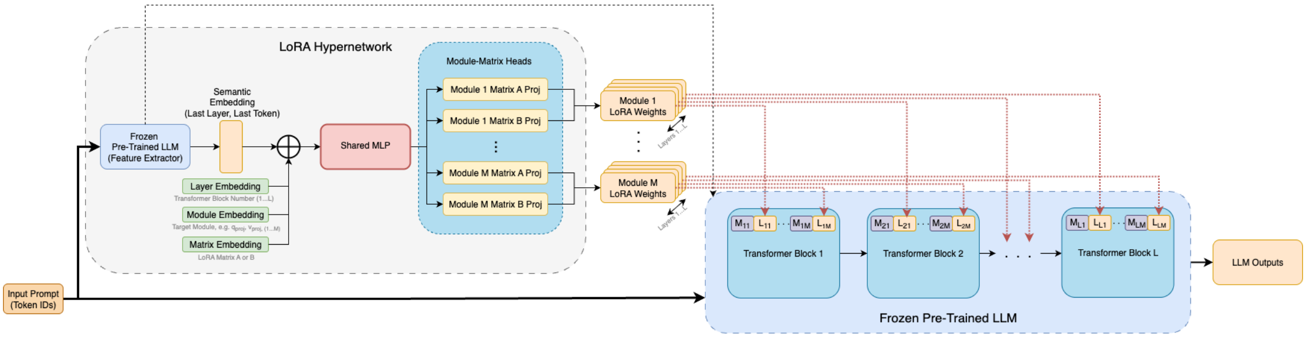

Figure 1: **Overview of the TaskWeaver architecture**. The hypernetwork (left) takes an input prompt and processes it through a frozen pre-trained LLM to extract a semantic embedding of the prompt (taken as the last layer representation of the last token). These embeddings are added to learned layer, module and matrix embeddings (A and B), then passed through a shared MLP backbone. Module-specific prediction heads generate LoRA weight matrices (A and B) for different modules (e.g., `q_proj`, `v_proj`, etc.) across all transformer layers. The generated LoRA weights are dynamically injected into the frozen base LLM (right) at inference time, adapting the model's behavior on a per-input basis without requiring gradient-based optimization.

This approach directly optimizes the hypernetwork to generate adapters that maximize base model performance on training examples. The specific hyperparameters for training the hypernetwork are currently being worked on.

Important Consideration An important consideration comes from handling different LoRA weights for each batch element during training. Text-to-LoRA sidesteps this by using the same task description for all the elements in a batch. However, since our model conditions the LoRA adapters on the instance, instead of predicting $A \in \mathbf{R}^{r \times d}$ and $B \in \mathbf{R}^{d \times r}$, the hypernetwork predicts $A \in \mathbf{R}^{b \times r \times d}$ and $B \in \mathbf{R}^{b \times d \times r}$, which are appropriately multiplied with the inputs using BMM in `DynamicLoraLinear`.

5 Experimental Setup and Infrastructure

5.1 Evaluation Framework

We implemented a flexible evaluation pipeline supporting both chat and non-chat model formats. In particular, it let's us configure our tests using simple YAML files, including the test dataset and coverage, and paths to models and LoRA parameters if necessary. We also control temperature of generation. When the model is marked as a chat model, we modify its prompts accordingly (setting user, assistant and system roles, and utilizing tokenizer's `apply_chat_template`).

5.2 Computational Resources

We have spread our training across multiple resources, including personal GPUs (an RTX 3070 and an M4 Pro Macbook), and 3090s and A100s rented from Vast.ai. Baseline LoRA training requires approximately 10-15 minutes on the A100 for the Qwen 0.6B per benchmark.

5.3 Experimental Results

Table 2 presents baseline performance across all four benchmarks and three model scales.

6 Plan Going Forward

We are currently on schedule with the original proposal plan and intend to complete the following for the final report.

6.1 Baseline Evaluation (Week +1)

Andrew, Raj:We plan to evaluate the baseline on more datasets, specifically ones used in Charakorn et al. [2025] such as HellaSwag [Zellers et al., 2019], Winogrande [Sakaguchi et al., 2020], Arc-C [Clark et al., 2018], etc. We will also evaluate the models on the all the benchmarks multiple times in order to get statistically significant results.

6.2 Hypernetwork Training and Evaluation (Week +1,2)

Dhruv, Raj: The collections of datasets will be combined into a superdataset in order to train the Hypernet-

Table 2: Baseline results comparing pretrained base models against task-specific LoRA fine-tuned models across three model scales and four benchmarks.

Model	**Variant**	**GSM8K**	**NLI**	**ARC-Easy**	**BoolQA**
Pythia 70M	Base	0.38%	6.30%	10.69%	13.64%
	+ LoRA FT	1.44%	33.66%	22.94%	41.80%
	+ **Hypernet (Ours)**	TBD	TBD	TBD	TBD
Gemma 270M-IT	Base	8.50%	31.87%	0.8%	41.81%
	+ LoRA FT	22.73%	32.37%	24.58%	53.98%
	+ **Hypernet (Ours)**	TBD	TBD	TBD	TBD
Qwen 0.6B	Base	60.8%	31.9%	29.7%	62.2%
	+ LoRA FT	72.78%	75.64%	76.56%	63.24%
	+ **Hypernet (Ours)**	TBD	TBD	TBD	TBD

work. The specific architecture of the hypernetwork's shared MLP is subject to change as we experiment with different architectures. Once these are finalized, we will do end-to-end training of the hypernetwork with our three models, and compare its performance with the baselines.

6.3 Predicted Adapter Analysis (Week +2)

Andrew, Dhruv: Once the hypernetwork is trained and evaluated, we will analyze the impact of various factors on the generated adapters, such as input prompt, layer index, target module, etc.

7 Hypothetical Compute Requirements

If allocated an additional $450 in compute credits, we would train the hyperparameter network for larger models using more datasets, and possibly also perform hyperparameter grid search. More specifically:

Larger Model Training and Validation ($300): Replicate experiments with Qwen 1.7B, 2B and 7B to validate that instance-level adaptation benefits scale beyond 600M parameters. This would incur approximately (3+3+11 (`model size contributions`)) × 15 `min/dataset` × 4 `datasets/model` × 5 `runs/model` × 3 `models` × $0.7/`hr`

Expand evaluation to more benchmarks ($15): Evaluate our approach on the full benchmark suite from our proposal (adding WinoGrande, HumanEval, MBPP, OpenBookQA, HellaSwag) to validate performance across code generation and additional reasoning domains. This would provide stronger evidence for the generality of instance-level adaptation. (15 `min/dataset` × 4 `datasets/model` × 5 `runs/model` × 3 `models` × $0.7/`hr` for just the small models we're currently looking at)

This allocation would significantly strengthen our conclusions about the viability, generality, and scalability of instance-level LoRA adaptation across diverse tasks and model sizes.

8 Conclusion

At the midterm checkpoint, we have made substantial progress toward our goal of instance-level test-time adaptation via hypernetwork-generated LoRA weights. We have successfully implemented comprehensive evaluation infrastructure supporting both chat-formatted (Gemma, Qwen) and base (Pythia) models, established baseline performance targets across three model scales (70M–600M parameters) and four diverse benchmarks, and developed a hypernetwork architecture that dynamically injects LoRA weights into a base LM for instance-level adaptation. The specific hyperparameters and architecture choice is currently being optimized.

Our multi-model approach provides unique insights into how instance-level adaptation scales with model capacity and interacts with instruction tuning. The initial experiments set up strong baselines, providing clear targets for hypernetwork performance evaluation.

The remaining work focuses on completing hypernetwork training across all three models, comprehensive comparative evaluation across multiple datasets (the 4 currently reported and another 2-4 datasets), and analysis of generated adapter characteristics including cross-model behavior and instance-level diversity. We remain confident in achieving our core objectives while preparing contingency plans to ensure meaningful contributions regardless of final performance outcomes.

References

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL `https://aclanthology.org/D15-1075`.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant transformer adaption. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=zWskCdu3QA`.

Christopher Clark, Kenton Lee, Ming-Wei Chang, and Tom Kwiatkowski. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 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. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 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.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

Team Gemma. Gemma 3. 2025. URL `https://goo.gle/Gemma3Report`.

David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=rkpACe1lx`.

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, 2022. URL `https://arxiv.org/abs/2203.08304`.

Rabeeh Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576. Association for Computational Linguistics, 2021.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

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.