Projectable Models: One-Shot Generation of Small Specialized Transformers from Large Ones

A. Zhmoginov [1] **J. Lee** [1] **M. Sandler** [1]

Abstract

Modern Foundation Models (FMs) are typically trained on corpora spanning a wide range of different data modalities, topics, downstream tasks. Utilizing these models can be very computationally expensive and is out of reach for most consumer devices. Furthermore, most of the broad FM knowledge may actually be irrelevant for a specific task at hand. Here we explore a technique for mapping parameters of a large Transformer to parameters of a smaller specialized model. By making this transformation task-specific, we aim to capture a narrower scope of the knowledge needed for performing a specific task by a smaller model. We study our method on image modeling tasks, showing that performance of generated models exceeds that of universal conditional models.

1. Introduction

Transformer-based generative models have recently shown a remarkable success in modeling complex data distributions across a wide spectrum of modalities including images, audio and language. In the language domain, Large Language Models (LLMs) became an extremely powerful tool demonstrating impressive performance across a large scope of language tasks. These models are typically very computationally complex to train and run and require vast amounts of text data that encompass a wide range of topics, facts and downstream applications. The generality of the resulting LLMs can be extremely advantageous, but also prove redundant or even detrimental in narrow applications and in specialized tasks (Raffel et al., 2020).

Here we propose an approach to generating a task-specific Transformer model of a smaller size from a larger Transformer. In our primary setup, similar to that explored in other related publications (see Appendix A), we maintain a functional universal large Foundation Model (FM) and use it to generate smaller specialized task-dependent Transformers with an emphasis on multi-task learning and personalization scenarios. Following the HyperNetwork approach (Ha et al., 2016), we generate the specialized models on the fly from the task specification without the need for any fine-tuning.

We study this idea experimentally in the visual domain adopting IMAGEGPT approach (Chen et al., 2020a), in which images are represented as sequences of discrete tokens and an autoregressive Transformer model is used to model this sequence distribution. Here we compare performance of task-specialized generated small Transformer models with universal counterparts of the same size. We also explore zero-shot generalization and knowledge transfer across different tasks.

2. Method

In the following, we outline our approach, where we use large Transformer models for generating smaller task-specific Transformers. We simultaneously view this as (a) a model specialization technique, where we can produce a small Transformer best suited for a particular task; (b) model compression technique where a single large model can be used for producing a variety of models of different sizes, and finally (c) a modular network approach, where individual generated task-specialized model implicitly share knowledge via a single large model. A more detailed discussion of weight space manifolds and associated model specialization techniques can be found in Appendix B.

2.1. Matrix Generators

In the Transformer architecture, the embedding size typically defines the sizes of all self-attention and MLP weight matrices. Choosing some maximum embedding size d, we define a family of Transformer models $\{\mathcal{M}_s\}_{s\in S}$ with the embedding size ds. We typically choose $s = 2^{-k}$ with integer $k \geq 0$ assuming that $d/2^k$ is integer for all models of interest. We generally assume that each $\mathcal{M}_s$ is a separate model pretrained on some input data distribution $p(\boldsymbol{x})$.

Given pretrained $\mathcal{M}_1$ how can we generate a smaller Trans-

[*]Equal contribution [1]Google DeepMind. Correspondence to: Andrey Zhmoginov <azhmogin@google.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

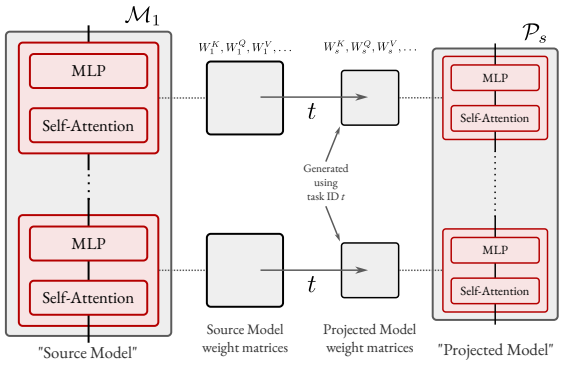

Figure 1. Model diagram. Given the task ID $\boldsymbol{t}$, the weights of the large "source model" $\mathcal{M}_1$ (Transformer with the embedding size d) are mapped to the weights of a smaller "projected model" $\mathcal{P}_s$ (Transformer with the embedding size ds). The loss of $\mathcal{P}_s$ on the input sequence is used to train the "projection operator" and tune $\mathcal{M}_1$, while a good performance of $\mathcal{M}_1$ can be maintained by continuing to train it with the original objective.

former with architecture of $\mathcal{M}_s$ and the embedding size ds? One natural way of doing this is to map each linear operator[1] $\hat{\mathrm{O}}_1^\ell(\boldsymbol{z}) := \mathrm{W}_1^\ell \boldsymbol{z} + \boldsymbol{b}_1^\ell$ from $\mathcal{M}_1$ to a corresponding linear operator[2] $\hat{\mathrm{O}}_s^\ell(\boldsymbol{z}) := \mathrm{W}_s^\ell \boldsymbol{z} + \boldsymbol{b}_s^\ell$ in the small model. Assuming that this transformation is linear, its most general form is simply

$$(\mathrm{W}_s)_{ij} := \sum_{k,l} \mathrm{V}^W_{ijkl} (\mathrm{W}_1)_{kl} + \mathrm{V}^B_{ij},$$
$$(\boldsymbol{b}_s)_i := \sum_{k} \mathrm{U}^W_{ik} (\boldsymbol{b}_1)_k + \mathrm{U}^B_{i}.$$

Projections. One natural choice of $(\mathrm{V}^W, \mathrm{V}^B, \mathrm{U}^W, \mathrm{U}^B)$ comes from analyzing linear transformations of the embedding space. Specifically, given two linear maps $\hat{\mathrm{Q}} : \mathbb{R}^{ds} \to \mathbb{R}^d$ and $\hat{\mathrm{P}} : \mathbb{R}^d \to \mathbb{R}^{ds}$ mapping embeddings between embedding spaces of two models, we can naturally define $\hat{\mathrm{O}}_s$ via $\hat{\mathrm{O}}_s(\boldsymbol{z}) \equiv \hat{\mathrm{P}}(\hat{\mathrm{O}}_1(\hat{\mathrm{Q}}\boldsymbol{z}))$. In other words, the action $\hat{\mathrm{O}}_s(\boldsymbol{z})$ is defined as a sequence of several steps: (a) mapping $\boldsymbol{z}$ to a higher-dimensional embedding space via $\hat{\mathrm{Q}}\boldsymbol{z}$, (b) acting on this activation with $\hat{\mathrm{O}}_1$ and finally (c) projecting the result down to the original low-dimensional embedding space with $\hat{\mathrm{P}}$. Ignoring biases, we can define $\hat{\mathrm{Q}}(\boldsymbol{z}) = \mathrm{Q}\boldsymbol{z}$ and $\hat{\mathrm{P}}(\boldsymbol{z}) = \mathrm{P}\boldsymbol{z}$ with Q and P being two matrices, thus simply obtaining:

$$\mathrm{W}_s = \mathrm{P}\mathrm{W}_1\mathrm{Q}, \qquad \boldsymbol{b}_s = \mathrm{P}\boldsymbol{b}_1. \tag{1}$$

In the following, we mainly adopt this model.

[1] all dense layers in self-attention and MLP operators

[2] In the following we use a hat to denote operators ($\hat{\mathrm{X}}$) and roman font for matrices (X).

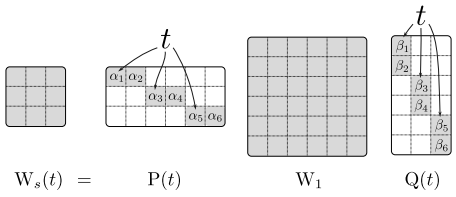

Figure 2. Projection of the source model weight W_1 into the projected model weight W_s. Here $\boldsymbol{t}$ is the task identifier and $\mathrm{P}(\boldsymbol{t})$, $\mathrm{Q}(\boldsymbol{t})$ are "projection matrices" whose diagonal elements are generated using shallow MLPs from $\boldsymbol{t}$.

2.2. Task-Dependent Matrix Generators

Our primary goal is generating small *task-specific* Transformer models, which requires that the generated weight matrices W_s depend on the task. In the following, we assume that each task is uniquely defined by a task identifier $\boldsymbol{t} \in T$ with T being a finite-dimensional vector space of all possible tasks. Given $\boldsymbol{t}$, we then define a *projection operator* ρ_s that first maps $\boldsymbol{t}$ to $\mathrm{V}(\boldsymbol{t})$ and $\mathrm{U}(\boldsymbol{t})$ at every layer as discussed below and then uses these tensors and the large Transformer *source model* $\mathcal{M}_1$ to generate the weights of a smaller *projected model* $\mathcal{P}_s = \rho_s(\boldsymbol{t}; \mathcal{M}_1)$, in effect defining a function family $\mathcal{F}(\rho_s, \mathcal{M}_1) = \{\rho_s(\boldsymbol{t}; \mathcal{M}_1) | \boldsymbol{t} \in T\}$.

If the tasks are specified explicitly as a part of the training dataset, we expect that $\rho_s(\boldsymbol{t}; \mathcal{M}_1)$ performs a task $\boldsymbol{t}$ better than a universal task-agnostic model of the same architecture. On the other hand, if task identifiers are not provided, we could design synthetic tasks with the goal of making the resulting $\mathcal{F}(\rho_s, \mathcal{M}_1)$ sufficiently rich to be useful for possible downstream tasks.

While there may be numerous choices of defining linear operators $\hat{\mathrm{V}}(\boldsymbol{t})$ and $\hat{\mathrm{U}}(\boldsymbol{t})$, here we adopt the *projection matrix* approach (1) by learning $\mathrm{P}(\boldsymbol{t})$ and $\mathrm{Q}(\boldsymbol{t})$ matrices (see below). Our experiments suggested that the best way to generate high-quality specialized $\mathcal{P}_s$ required that the weights of $\mathcal{M}_1$ are also tuned thus increasing the number of trained parameters and modifying a pretrained source model $\mathcal{M}_1$ to be compatible with generated projections. We tuned $\mathcal{M}_1$ and learned projection operators by optimizing both the original loss on $\mathcal{M}_1$ to maintain its quality and the loss on the projected model $\mathcal{P}_s$ (see Eq. (2) below).

Since the projection matrix generator with many parameters can be susceptible to overfitting on a given task distribution, we propose a simple and low-parameter method of generating $\mathrm{P} \in \mathbb{R}^{(sdn)\times(dn)}$ and $\mathrm{Q} \in \mathbb{R}^{(dn)\times(sdn)}$. Namely, we chose to train a set of MLPs that given $\boldsymbol{t}$ generated two vectors $\boldsymbol{p}(\boldsymbol{t})$ and $\boldsymbol{q}(\boldsymbol{t})$ and chose: $\mathrm{P}_{ij} = p_j(\boldsymbol{t})\delta_{i,\lfloor sj \rfloor}$ and $\mathrm{Q}_{ij} = q_i(\boldsymbol{t})\delta_{j,\lfloor si \rfloor}$ (see Fig. 2). The resulting transformation $\mathrm{W}_s = \mathrm{P}\mathrm{W}_1\mathrm{Q}$ can be seen as generating the output rows and columns by linearly combining 2^k rows and columns of the source matrix. While there may be many other ap-

proaches to generating weight matrices W_s, we leave this question to be addressed in the future work.

2.3. Optimization Objective

During training, we typically optimized the sum of the losses of the projected model $\mathcal{P}_s$ and the source model $\mathcal{M}_1$:

$$\mathbb{E}_{(\boldsymbol{\xi},\boldsymbol{t})\sim\mathcal{D}_*}\mathcal{L}_{\mathcal{P}_s(\boldsymbol{t})}(\boldsymbol{\xi}) + w_{\text{src}}\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{D}}\mathcal{L}_{\mathcal{M}_1(\boldsymbol{t})}(\boldsymbol{\xi}), \qquad (2)$$

where $\boldsymbol{\xi} = (\boldsymbol{x}, \boldsymbol{y})$, $\mathcal{D}$ is the dataset used to train the Foundation Model $\mathcal{M}_1$, $\mathcal{D}_*$ is the multi-task dataset for training projected models and w_{src} is an optional multiplier used to re-weight the importance of the Foundation Model objective. In autoregressive models, $\boldsymbol{y}$ is typically a shifted version of the sequence $\boldsymbol{x}$.

3. Experiments

In the following, we describe our experiments with model projections. All of our experiments were conducted in the image domain and followed the IMAGEGPT approach (Chen et al., 2020a) using two separate datasets: synthetic dataset SYNTHMNIST and a dataset based on IMAGENET.

3.1. Experimental Setup: Datasets

SYNTHMNIST. Our synthetic image dataset SYNTHMNIST was based on MNIST with each 32×32 image generated by a deterministic function of a task identifier $\boldsymbol{t} \in \mathbb{R}^{18}$ specifying the background and d encoding the index of the MNIST image to be overlaid on top of this texture. The vector $\boldsymbol{t}$ uniquely defined which of the three distinct types of the background textures to use and specified texture parameters such as scale, rotation, color and distortion. Treating d as a hidden variable, the image distribution defining our multi-task dataset was given by $p(\boldsymbol{i}|\boldsymbol{t})p(\boldsymbol{t}) = \sum_d p(\boldsymbol{i}|\boldsymbol{t}, d)p(\boldsymbol{t})p(d)$. Some examples are illustrated in Figure 3. For a more detailed discussion of the dataset see Appendix C.

IMAGENET. For our experiments with realistic images, we used 64×64 RGB images from the IMAGENET dataset (Deng et al., 2009). The task identifier $\boldsymbol{t}$ associated with each image was chosen to be an embedding produced by a pretrained SIMCLRV2 model (Chen et al., 2020b). Specialized IMAGEGPT models are therefore trained to model a distribution of images with a given embedding $\boldsymbol{t}$.

Figure 3. Examples of 32×32 synthetic images generated from the task ID $\boldsymbol{t}$ encoding the background texture (1 of 3 types of the texture, scale, rotation, colors, distortion, etc.) and the overlaid MNIST digit.

3.2. Experimental Setup: Model Architecture

In our SYNTHMNIST experiments, all 32×32 RGB input images were quantized by separately mapping each pixel into one of 512 discrete tokens corresponding to one of the color clusters. These tokens were then flattened into 1024-long sequences and finally modelled autoregressively using GPT-2-style models (Radford et al., 2019). Following Chen et al. (2020a), our experiments with 64×64 RGB IMAGENET images (Deng et al., 2009) used a simple learned CNN model with vector quantization for producing a final sequence of 1024 discrete tokens. For details, see Appendix C.

The base model $\mathcal{M}_1$ (that we typically used as a source foundation model) had 24 layers, 8 heads and used the embedding dimension $d = 512$. Our smaller models $\mathcal{M}_s$ and $\mathcal{P}_s$ with the multiplier $s \in \{1/2, 1/4, 1/8\}$ used the same architecture as $\mathcal{M}_1$, but had a smaller embedding size sd. The number of trainable parameters in models $\mathcal{M}_1$ through $\mathcal{M}_{1/8}$ was approximately equal to 76.7M, 19.5M, 5M and 1.3M correspondingly.

Weight Generators. In our experiments, we used a specific choice of the MLP producing P and Q projection matrices. The input task identifier $\boldsymbol{t}$ was first linearly mapped to an r-dimensional vector (with r typically chosen between 4 and 32), which after the SWISH nonlinearity was followed by a linear layer producing diagonal elements of P and Q (as described in Sec. 2.2).

3.3. Experimental Results

3.3.1. BASELINE MODELS

We started by pretraining a GPT-2 based autoregressive model $\mathcal{M}_1$ on the image distribution (SYNTHMNIST and IMAGENET). We also trained a number of smaller baseline GPT-2 models $\mathcal{M}_s$ for the target model multiplier $s \in \{1/2, 1/4, 1/8\}$.

Along with $\mathcal{M}_s$ we also trained conditional autoregressive models $\tilde{\mathcal{M}}_s$. These models were identical to $\mathcal{M}_s$, but were trained on sequences that in addition to image pixels also encoded the information about the requested task identifier $\boldsymbol{t}$ as the first embedding in the sequence. As a result, while $\mathcal{M}_s$ is a model capable of generating arbitrary images, specialized models $\tilde{\mathcal{M}}_s$ can generate images with any predefined task identifier $\boldsymbol{t}$. The losses of all unconditional and conditional pretrained models are shown in Table 1 for SYNTHMNIST and in Table 2 for IMAGENET datasets. In the following, we compare our specialized generated models $\mathcal{P}_s$ to corresponding *universal conditional* models $\tilde{\mathcal{M}}_s$.

s	$\mathcal{M}_s$	$\tilde{\mathcal{M}}_s$	$\mathcal{P}_s$	$\tilde{\mathcal{P}}$
1	0.45	**0.39**	(0.44)	(0.46)
1/2	0.51	0.46	**0.39**	**0.39**
1/4	0.62	0.58	0.47	**0.45**
1/8	0.82	0.78	**0.57**	**0.57**

Table 1. Model losses on the SYNTHMNIST image dataset: (a) $\mathcal{M}_s$ is a conventional autoregressive model using multiplier s; (b) $\tilde{\mathcal{M}}_s$ is an autoregressive model conditioned on $\boldsymbol{t}$ passed in the first input token; (c) $\mathcal{P}_s$ is a projected model co-training the original $\mathcal{M}_1$ and task-specific $\mathcal{P}_s(\boldsymbol{t})$; (d) $\tilde{\mathcal{P}}$ is a single model simultaneously co-training $\mathcal{M}_1$ and projections $\mathcal{P}_s(\boldsymbol{t})$ for all $s \in \{1/2, 1/4, 1/8\}$. Cells $s = 1$ and $\mathcal{P}_s/\tilde{\mathcal{P}}$ show "source model" $\mathcal{M}_1$ performance after co-training (average value for all s for $\mathcal{P}_s$).

s	$\tilde{\mathcal{M}}_s$	$\mathcal{P}_s$
1	2.176	–
1/2	**2.211**	**2.210**
1/4	2.264	**2.246**
1/8	2.332	**2.297**

Table 2. Results for IMAGENET with SIMCLRV2 embeddings as task identifiers (evaluated on the validation set). The statistical error of $\tilde{\mathcal{M}}_s$ is around 0.001 and of $\mathcal{P}_s$ is approximately 0.002.

3.3.2. PROJECTIONS FROM $\mathcal{M}_1$

As our first experiment, we used a pretrained $\mathcal{M}_1$ to train our projected models $\mathcal{P}_s$. Specifically, we continued to tune $\mathcal{M}_1$ and our projection operators (described in Sec. 2) using optimization objective (2). We carried out this process for each projected model size $s \in \{1/2, 1/4, 1/8\}$ with the inner dimension $r = 8$ (see Sec. 3.2). The number of parameters used by the projection operators ranged from 2.4M for the $s = 1/2$ model to 1.9M for $s = 1/8$ model.

The results of our experiments are presented in Tables 1 and 2. We observed that for both SYNTHMNIST and IMAGENET generated projected models $\mathcal{P}_s(\boldsymbol{t})$ typically outperformed universal conditional models since the former were generated from the task identifier $\boldsymbol{t}$ to only model task-specific knowledge $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n|\boldsymbol{t})$, whereas the latter had to approximate the entire $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{t})$. The difference was especially pronounced on the SYNTHMNIST dataset, where performance of projected models was seen to be roughly comparable to the performance of universal conditional models that used 4 times as many parameters[3]. On IMAGENET, the gap was smaller and we estimated that the performance of $\mathcal{P}_{1/4}$ would roughly match the performance of a $1.6\times$ larger universal model, whereas $\mathcal{P}_{1/8}$ would roughly match a $2.5\times$ larger conditional GPT-2.

These results illustrate the promise of generating small specialized Transformers that proved to be much more suitable for modeling narrow image distributions (for a fixed $\boldsymbol{t}$, all SYNTHMNIST images share a similar background texture and IMAGENET images are more likely to contain the same object or a scene). It is also worth noticing that the source model $\mathcal{M}_1$ co-trained with projections ended up having the same perplexity as a standalone model suggesting that the resulting tuned $\mathcal{M}_1$ can be used as a "foundation model" without any sacrifices to its performance.

[3] $\mathcal{P}_{1/8}$ roughly matching the performance of $\tilde{\mathcal{M}}_{1/4}$, $\mathcal{P}_{1/4}$ roughly matching $\tilde{\mathcal{M}}_{1/2}$, etc.

3.3.3. CO-TRAINING MULTIPLE PROJECTED MODELS

In the followup experiment, we trained a model $\tilde{\mathcal{P}}$ that simultaneously projected $\mathcal{M}_1$ to all sizes $s \in \{1/2, 1/4, 1/8\}$. Here, each $\mathcal{P}_s$ has it's own projection operator, while the source model $\mathcal{M}_1$ needs to be compatible with all of the projected model sizes. Results presented in Table 1 suggest that co-training projection models of various sizes did not lead to performance degradation (even showing minor improvement in some cases). As a result, the final tuned $\mathcal{M}_1$ with projection operators for all $s \in \{1/2, 1/4, 1/8\}$ provides us with the full spectrum of models: (a) a large foundation model describing the full image distribution, and (b) a capability to produce small task-specific models with a variety of different sizes.

3.3.4. ADDITIONAL EXPERIMENTS

Appendix D discusses additional experimental results with both SYNTHMNIST and IMAGENET datasets. Specifically, we discuss our studies of (a) the effect of the source model size on the specialized model performance (D.1); (b) cross-task knowledge transfer in projected models (D.2) and (c) zero-shot generalization of projected models (D.3).

4. Discussion

In this work, we propose and study a mechanism for converting the weights of a large Transformer into the weights of a smaller task-specialized model. While in the ideal case scenario, the source Transformer is an arbitrary pretrained model, our technique involves tuning this source model. We then show that the resulting "projected" task-specialized models outperform universal task-agnostic models and the improvement is more pronounced for smaller generated models and larger source models. Interestingly, we can use a single source Foundational Model and a set of "projection operators" to generate a variety of specialized models of different sizes without any noticeable degradation of the source model performance. We also demonstrate a cross-task knowledge transfer in projected models, when a wealth of training data for some tasks improves performance of projected models on related, but different tasks.

References

Alaluf, Y., Tov, O., Mokady, R., Gal, R., and Bermano, A. Hyperstyle: Stylegan inversion with hypernetworks for real image editing. In *Proceedings of the IEEE/CVF conference on computer Vision and pattern recognition*, pp. 18511–18521, 2022.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. Generative pretraining from pixels. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1691–1703. PMLR, 2020a.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. E. Big self-supervised models are strong semi-supervised learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020b.

Chen, X., Hu, Y., and Zhang, J. Streamlining redundant layers to compress large language models. *arXiv preprint arXiv:2403.19135*, 2024.

Deb, B., Zheng, G., and Awadallah, A. H. Boosting natural language generation from instructions with meta-learning. *arXiv preprint arXiv:2210.11617*, 2022.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848.

Dinh, T. M., Tran, A. T., Nguyen, R., and Hua, B.-S. Hyperinverter: Improving stylegan inversion via hypernetwork. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11389–11398, 2022.

Ha, D., Dai, A. M., and Le, Q. V. Hypernetworks. *CoRR*, abs/1609.09106, 2016. URL http://arxiv.org/abs/1609.09106.

He, Y., Zheng, S., Tay, Y., Gupta, J., Du, Y., Aribandi, V., Zhao, Z., Li, Y., Chen, Z., Metzler, D., et al. Hyperprompt: Prompt-based task-conditioning of transformers. In *International conference on machine learning*, pp. 8678–8690. PMLR, 2022.

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

Ivison, H., Bhagia, A., Wang, Y., Hajishirzi, H., and Peters, M. Hint: Hypernetwork instruction tuning for efficient zero-& few-shot generalisation. *arXiv preprint arXiv:2212.10315*, 2022.

Kang, M., Zhu, J.-Y., Zhang, R., Park, J., Shechtman, E., Paris, S., and Park, T. Scaling up gans for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10124–10134, 2023.

Knyazev, B., Drozdzal, M., Taylor, G. W., and Romero Soriano, A. Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems*, 34:29433–29448, 2021.

Li, Y., Ma, X., Lu, S., Lee, K., Liu, X., and Guo, C. Mend: Meta demonstration distillation for efficient and effective in-context learning. *arXiv preprint arXiv:2403.06914*, 2024.

Liang, C., Karampatziakis, N., Zhao, T., and Chen, W. Hart: Efficient adaptation via regularized autoregressive parameter generation. 2023.

Lin, Y., Li, Y., Wang, Z., Li, B., Du, Q., Xiao, T., and Zhu, J. Weight distillation: Transferring the knowledge in neural network parameters. *arXiv preprint arXiv:2009.09152*, 2020.

Littwin, G. and Wolf, L. Deep meta functionals for shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1824–1833, 2019.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *CoRR*, abs/2305.11627, 2023. doi: 10.48550/ARXIV.2305.11627. URL https://doi.org/10.48550/arXiv.2305.11627.

Mahabadi, R. K., Ruder, S., Dehghani, M., and Henderson, J. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

Mu, J., Li, X., and Goodman, N. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36, 2024.

Nirkin, Y., Wolf, L., and Hassner, T. Hyperseg: Patch-wise hypernetwork for real-time semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4061–4070, 2021.

Phang, J. Investigating the effectiveness of hypertuning via gisting. *arXiv preprint arXiv:2402.16817*, 2024.

Phang, J., Mao, Y., He, P., and Chen, W. Hypertuning: Toward adapting large language models without back-propagation. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 27854–27875. PMLR, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Samragh, M., Farajtabar, M., Mehta, S., Vemulapalli, R., Faghri, F., Naik, D., Tuzel, O., and Rastegari, M. Weight subcloning: direct initialization of transformers using larger pretrained ones. *CoRR*, abs/2312.09299, 2023. doi: 10.48550/ARXIV.2312.09299. URL https://doi.org/10.48550/arXiv.2312.09299.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33: 7462–7473, 2020.

Spurek, P., Kasymov, A., Mazur, M., Janik, D., Tadeja, S. K., Tabor, J., Trzciński, T., et al. HyperPocket: Generative point cloud completion. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6848–6853. IEEE, 2022.

Tack, J., Kim, J., Mitchell, E., Shin, J., Teh, Y. W., and Schwarz, J. R. Online adaptation of language models with a memory of amortized contexts. *arXiv preprint arXiv:2403.04317*, 2024.

Tang, Y., Wang, Y., Guo, J., Tu, Z., Han, K., Hu, H., and Tao, D. A survey on transformer compression. *arXiv preprint arXiv:2402.05964*, 2024.

Tay, Y., Zhao, Z., Bahri, D., Metzler, D., and Juan, D.-C. Hypergrid transformers: Towards a single model for multiple tasks. In *International conference on learning representations*, 2020.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Volk, T., Ben-David, E., Amosy, O., Chechik, G., and Reichart, R. Example-based hypernetworks for multi-source adaptation to unseen domains. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Von Oswald, J., Henning, C., Grewe, B. F., and Sacramento, J. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

Wang, Q., Yang, X., Lin, S., Wang, J., and Geng, X. Learngene: Inheriting condensed knowledge from the ancestry model to descendant models. *arXiv preprint arXiv:2305.02279*, 2023.

Wang, W., Chen, W., Luo, Y., Long, Y., Lin, Z., Zhang, L., Lin, B., Cai, D., and He, X. Model compression and efficient inference for large language models: A survey. *arXiv preprint arXiv:2402.09748*, 2024.

Xia, M., Gao, T., Zeng, Z., and Chen, D. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Xu, Z., Chen, Y., Vishniakov, K., Yin, Y., Shen, Z., Darrell, T., Liu, L., and Liu, Z. Initializing models with larger ones. *arXiv preprint arXiv:2311.18823*, 2023.

Ye, Q. and Ren, X. Learning to generate task-specific adapters from task description. *arXiv preprint arXiv:2101.00420*, 2021.

Zhang, C., Ren, M., and Urtasun, R. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

Zhang, L., Rao, A., and Agrawala, M. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3836–3847, 2023.

Zhao, H., Qiu, Z., Wu, H., Wang, Z., He, Z., and Fu, J. Hypermoe: Towards better mixture of experts via transferring among experts. *arXiv preprint arXiv:2402.12656*, 2024.

A. Related Work

HyperNetworks. One-shot generation of entire models has been popularized in (Ha et al., 2016). Since then, multiple applications including those in image generation and segmentation (Zhang et al., 2023; Alaluf et al., 2022; Dinh et al., 2022; Nirkin et al., 2021; Kang et al., 2023) (including conditioning Stable Diffusion (Rombach et al., 2022)), continual learning (Von Oswald et al., 2019), 3D space representations (Littwin & Wolf, 2019; Sitzmann et al., 2020; Spurek et al., 2022) and network architecture search (Zhang et al., 2018; Knyazev et al., 2021) have been proposed. In natural language processing with Transformers, HyperNetwork-based approaches have also been widely adopted (Ye & Ren, 2021; Deb et al., 2022; Tay et al., 2020; Mahabadi et al., 2021; Ivison & Peters, 2022). Two most recent directions target task-dependent prompt generation (He et al., 2022) and one-shot generation of model perturbations (Phang et al., 2023) given a single or few examples of a novel task. Several other related approaches have been explored since then in (Volk et al., 2023; Ivison et al., 2022; Liang et al., 2023; Phang, 2024; Zhao et al., 2024; Li et al., 2024; Tack et al., 2024; Mu et al., 2024).

Generating smaller Transformer models. With the explosion of interest in Transformer models, there has been a growing interest in techniques for generating smaller Transformers from larger capable pretrained models. For example, (Xia et al., 2023; Ma et al., 2023; Chen et al., 2024) propose LLM pruning techniques and (Lin et al., 2020; Wang et al., 2023; Xu et al., 2023; Samragh et al., 2023) outline approaches for initializing weights of a smaller model using a large Transformer model. Two recent reviews (Wang et al., 2024; Tang et al., 2024) discuss recent publications exploring Transformer model pruning among other approaches.

B. Weight Space Manifold

Consider a Transformer-based LLM trained on rich input data distribution $p(x)$. Given a specific task t, one approach to solving it relies on choosing a prompt $r(t)$ and modeling task-specific distribution $p(x|t)$ as $p_{\theta_\circ}(x; r(t))$ with the prompt r used as a fixed sequence prefix. Here the parameters $\theta_\circ$ of the trained model are kept fixed and are independent of the task even if it is known in advance and does not change often.

Another way of specializing to task t is based on modifying the model itself. Perhaps the most widely used approach is to fine-tune the model $\theta_\circ \to \theta_*(t)$ on a small number of demonstrations from the task-specific distribution $p(x|t)$. Note that the final weights $\theta_*(t)$ generally depend on the tuning procedure, employed seeds, etc.

Yet another specialization technique is based on the HyperNetwork-based approach (Ha et al., 2016). Here we assume that there exists an almost everywhere smooth *weight space manifold* $\theta(t)$ such that for each task $t \in T$ it approximately holds that $p_{\theta(t)}(x) \approx p(x|t)$. This manifold could be chosen to approximate a set of pretrained fine-tuned models $\{\theta_*(t_i)\}_i$ for some $\{t_i \in T\}_i$, or could be learned directly on a given distribution of tasks over T. Note that by learning the manifold $\theta(t)$, we store some information about the task-related knowledge in the model describing $\theta(t)$ thus potentially improving the performance of the specialized generated models since they no longer need to carry information and knowledge relevant to other potential questions and queries.

Weight manifold. The weight manifold of the model can be extremely complex. However, performing Taylor decomposition with respect to t around some fixed "base task" chosen here as $t = 0$, we can derive an approximate expression for $\theta(t)$:

$$\theta(t) \approx \theta(0) - \left(\frac{\partial^2 \mathcal{L}}{\partial \theta^2}\right)^{-1} \frac{\partial^2 \mathcal{L}}{\partial \theta \, \partial t} t, \tag{3}$$

where $\mathcal{L}(\theta, t)$ is a model loss and all derivatives are computed at $t = 0$ and $\theta = \theta(0)$. In other words, this equation explicitly defines the hyperplane tangent to the weight manifold at $t = 0$. However, performing this computation in practice can be exceptionally expensive and can only describe the local structure of the model.

Topology of the task space. The structure of the weight manifold $\theta(t)$ is defined by the model and the topology of the task space. In a typical setup, the task t could be, for example, an embedding of a conversation topic, some representation of the user writing style, image embedding, or metadata. If we choose t to be some normalized embedding $\xi(x)$, the task manifold becomes a sphere and $\theta(t)$ could also have a similar topology. Given a normalized embedding ξ, we can also construct a distribution $p(x|t)$ defined for all $\|t\| \leq 1$ as a linear interpolation between the marginalized distribution and

$p(x|\xi)$ with $\|\xi\| = 1$:

$$p(x|t) := (1 - \|t\|) \int_{\|\xi\|=1} p(x|\xi)p(\xi)d\,\xi + \|t\|p(x|\xi = t/\|t\|).$$

Using this constructed conditional distribution, the expansion around $t = 0$ becomes an expansion around the average distribution with the direction of t defining the embedding of samples that become more prevalent in the distribution as $\|t\|$ grows.

B.1. Learning the Weight Manifold

Performing the computation in Eq. (3) can be prohibitively expensive. An alternative approach to characterizing the weight manifold $\theta(t)$ is then to learn this dependence.

Linear approximation. For example, if we are only interested in the local linear structure of $\theta(t)$, we can rely on the approximation:

$$\delta\theta_\alpha(t) := \theta_\alpha(t) - \theta_\alpha(0) \approx \sum_i \mathrm{S}_{\alpha,i} t_i, \tag{4}$$

where α is an index of a specific parameter. We can then learn both $\theta(0)$ and $\mathrm{S}_{\alpha,i}$ by drawing samples from the distribution $p(x|t)p(t)$ and training the model to optimize the loss $\mathcal{L}(\theta(0), \mathrm{S}) := \mathbb{E}_{(x,t)\sim\mathcal{D}} L(x; \theta(t))$. This approach requires $\dim\theta \cdot \dim t$ total parameters to parameterize $\delta\theta(t)$.

Modular approximation. The linear approximation can be trivially reduced to a form of a *modular network* used previously in multiple contexts. Here we can represent $\delta\theta$ via a linear combination of a large collection of individual "modules" $\mathrm{S}_{\alpha,k}$:

$$\delta\theta_\alpha = \sum_k \mathrm{S}_{\alpha,k}\eta_k(t), \tag{5}$$

where $\eta_k(t)$ can now be arbitrarily complex non-linear functions of t. This modular approximation requires $\dim\theta \cdot \dim\eta$ parameters and could be "lighter" than the linear approximation if $\dim\eta < \dim t$. Notice that if η_k are linear functions, this is equivalent to a low-rank version of Eq. (4).

Low-rank approximation. The required number of parameters necessary to describe $\delta\theta$ can be reduced further by replacing kernels W of all linear operators in the model with low-rank approximations:

$$\delta\mathrm{W}_\ell(t) = \mathrm{W}_\ell(t) - \mathrm{W}_\ell(0) \approx \mathrm{L}_\ell(t)\mathrm{R}_\ell(t) \tag{6}$$

with ℓ being the operator index and $\mathrm{L}_\ell(t)\mathrm{R}_\ell(t)$ forming a low-rank matrix. Both $\mathrm{L}_\ell(t)$ and $\mathrm{R}_\ell(t)$ can be represented as shallow MLPs that accept t as input and produce all matrix components. The required number of parameters to describe an $n \times m$ matrix $\delta\mathrm{W}_\ell(t)$ is now reduced to roughly $O((n+m)r\dim t)$ with r being the rank. In the simplest cast with $r = 1$, L_ℓ is a column and R_ℓ is a single row.

B.2. Projection Approximation

Another way of generating $\delta\theta_\alpha$ is motivated by noticing that the individual module matrices $\mathrm{S}_{\ell,k}$ combined as in Eq. (5) could sometimes be arranged to form a larger matrix. This can be viewed as a way of mapping a larger Transformer into a smaller model.

One natural choice in this potentially very rich family of transformation comes from analyzing linear maps of the embedding space. Specifically, given two linear transformations $\hat{\mathrm{Q}} : \mathbb{R}^{ds} \to \mathbb{R}^d$ and $\hat{\mathrm{P}} : \mathbb{R}^d \to \mathbb{R}^{ds}$ mapping model activations between embedding spaces of two models with embedding sizes d and ds and $s = 2^{-k} < 1$, $k \in \mathbb{Z}$, we can naturally define $\hat{\mathrm{O}}_s$ via

$$\hat{\mathrm{O}}_s(\boldsymbol{z}) \equiv \hat{\mathrm{P}}(\hat{\mathrm{O}}_1(\hat{\mathrm{Q}}\boldsymbol{z})),$$

where $\hat{\mathrm{O}}_s$ and $\hat{\mathrm{O}}_1$ are two linear operators in a smaller and a larger Transformer model correspondingly. This perspective gives rise to a family of weight generators at the core of our main approach discussed in Sec. 2.

Connection to Modular Networks. Conventional modular architectures can be viewed as a special case of the described linear weight transformation. Consider an $(dn) \times (dm)$ weight matrix W_1 of $\hat{\mathrm{O}}_1$ and a smaller $(dsn) \times (dsm)$ weight matrix W_s of $\hat{\mathrm{O}}_s$. One can subdivide W_1 into $(dsn) \times (dsm)$-sized blocks and choose

$$V_{ijkl} = \sum_{\alpha=0}^{s^{-1}-1} \sum_{\beta=0}^{s^{-1}-1} r_{\alpha\beta} \delta_{k,i+dsn\alpha} \delta_{l,j+dsm\beta},$$

resulting in the weight W_s becoming a linear combination of these blocks. This particular form is frequently referred to as a *modular network* with individual blocks acting as *modules*. Here δ is a Kronecker delta and $r_{\alpha\beta}$ are s^{-2} coefficients of individual modules forming W_s.

C. Models and Datasets

C.1. Training Details

A typical model has been trained for $200\mathrm{k}$ to $800\mathrm{k}$ steps with Adam optimizer and a learning rate of order of 10^{-3} with cosine learning rate decay schedule ($10\mathrm{k}$ warmup steps). We typically used very weak weight decay (10^{-10} to 10^{-8}), but our dropout was frequently set to 10%.

C.2. VQ-VAE Model

We trained a separate VQ-VAE model (Van Den Oord et al., 2017) for mapping 64×64 RGB images to sequences of 1024 tokens taking values in a discrete set of size 512. Model encoder contained 3 CNN layers with 3×3 kernels and leaky ReLU nonlinearities. Layers had the following depths and strides: $(32, 2)$, $(64, 1)$, $(64, 1)$. The decoder was composed of 4 transpose-convolutional leaky ReLU layers. Layers had the following depths, kernel sizes and strides: $(64, 3, 2)$, $(64, 3, 1)$, $(32, 3, 1)$ and the final $(3, 1, 1)$. The full VQ-VAE model was typically trained with L_2 loss and $\beta = 0.2$ (Van Den Oord et al., 2017).

C.3. SYNTHMNIST Dataset

Each SYNTHMNIST image was generated by (a) first using t to produce an image texture and (b) overlaying one of the MNIST images on top of this texture. Examples of generated images can be found in Fig. 3.

Texture. Task identifier t was sampled uniformly from $[0, 1]^{18}$ and contained information about the image affine transformation like angle of rotation (t_0), scale (t_1), rotation center x and y coordinates (t_2 and t_3). The actual angle was equal to $2\pi t_0$, the scale was chosen as $\mathrm{sc}(t_1) := 2 + 18t_1$. The image was also "warped" by applying a transformation $x \to x + a_x \cos(s_x x)$ and $y \to y + a_y \cos(s_y y)$ with $a_{x,y} := \mathrm{relu}(\bar{a}_{x,y} - \alpha)/(1 - \alpha)$, $\alpha = 0.3$, $\bar{a}_x = t_4$, $\bar{a}_y = t_5$ and $s_x = s_y = \mathrm{sc}(t_6)$. Each color component (red, green, blue) of the image was shifted relative to others as defined by t_7 for R, t_8 for G and t_9 for B. The image texture itself was randomly chosen from one of 3 classes (depending on which of t_{10}, t_{11} and t_{12} is larger) based on the following function profiles: (a) $\cos \tilde{x}$ with $\tilde{x}$ being a transformed x coordinate, (b) $\cos \tilde{\rho}$ with $\tilde{\rho}$ being a distance to the origin in transformed coordinates, (c) $\cos(10gt_{13} + 2t_{14})$ with $g := \cos(\tilde{x}/3)\cos(\tilde{y}/3)$ and $\tilde{y}$ being a transformed y coordinate. Finally, the generated texture was multiplied by a random 3-channel RGB-color vector $(\beta + \gamma t_{15}, \beta + \gamma t_{16}, \beta + \gamma t_{17})$, where we chose $\beta = 0.5$ and $\gamma = 1.5$. The produced image was clipped to a $[-1, 1]$ output range.

Digit. In the training dataset split, we used a random augmentation by horizontally flipping digit images with 50% probability.

D. Additional Experimental Results

D.1. Projections from $\mathcal{M}_s$

The task-specialized projected model $\mathcal{P}_s(t)$ is expected to derive its capabilities from: (a) the source model $\mathcal{M}_1$, the weights of which are used to generate $\mathcal{P}_s$ and (b) the projection operator parameterizing the way that these source weights are used depending on the task identifier t. It is interesting to ask which of these sources is more important for the performance of the projected model $\mathcal{P}_s$.

We explored this question by using source models $\mathcal{M}_s$ of different sizes s for producing projected models $\mathcal{P}_{s'}$ with $s' < s$. We also varied the size of the inner dimension r to control the number of parameters in the projection operator itself. The number of parameters for the source $\mathcal{M}_s$ with a particular value of r is roughly proportional to rs. Projecting from $\mathcal{M}_1$ with $r = 8$ is thus roughly equivalent to projecting from $\mathcal{M}_{1/2}$ with $r = 16$ and from $\mathcal{M}_{1/4}$ with $r = 32$. The results of our experiments presented in Table 3 suggest that the performance of the projected model is generally better for larger source models. However, the benefit from using larger source models may be less pronounced once they exceed a particular size.

D.2. Cross-Task Knowledge Transfer

In our approach, models generated for different tasks indirectly share their weights via the source model. It is thus interesting to explore the degree to which these models exchange information and quantify knowledge transfer across different tasks. We accomplish this by noticing that providing additional training samples for some tasks may boost the performance of models generated for related but different tasks. We study this phenomenon in IMAGENET models by subdividing the training set into two parts based on the image labels: (a) part C_1 contains images with labels below 500 and C_2 with labels above 500 and (b) part C_1 containing images with labels $l \equiv 0 \pmod 2$ and part C_2 with image labels $l \equiv 1 \pmod 2$. We denote the first scenario as "$1/2 - 1/2$" and the second scenario as "$\bmod 2$". Since IMAGENET labels are not entirely random, but are ordered in a semantically meaningful way, we expect that tasks in C_1 and C_2 sets in the "$\bmod 2$" scenario are more closely related to each other compared to the "$1/2 - 1/2$" setup.

In our experiments, we trained three $\mathcal{P}_{1/2}$ models for each of the two scenarios: (a) using all training samples for both C_1 and C_2; (b) using only $1/8$ of all existing training samples for both C_1 and C_2; (c) using $1/8$ of training samples for C_1 and all training samples for C_2. The results of our experiments are presented in Table 4. We observed that providing additional examples for tasks in C_2 actually boosts the performance of models generated for tasks in C_1. This boost is especially noticeable in the "$\bmod 2$" scenario, where tasks in C_1 and C_2 are closely related. Similarly, notice that the performance on C_2 suffers from a smaller training set for C_1. In other words, there is a noticeable interaction and knowledge transfer between models generated for different, but related tasks.

D.3. Zero-Shot Generalization

An important property of any model specialization technique is it's zero-shot generalization capability. We explored model generalization for SYNTHMNIST dataset by varying the scale component t_{scale} of the task identifier t from 0 to 2, while the training range of scales was $[0, 1]$. In Figure 4 we show the average loss of different models as a function of t_{scale} (horizontal *scale* axis in all of the plots). Notice that while all models start degrading around $t_{\text{scale}} = 1$, the projected model $\mathcal{P}_{1/2}$ shows better generalization than the Conditional GPT-2 model $\tilde{\mathcal{M}}_{1/2}$. However, $\mathcal{P}_{1/4}$ and especially $\mathcal{P}_{1/8}$ have much greater difficulty generalizing to $t_{\text{scale}} \approx 2$, which could be explained by the fact that the corresponding projection operators used a large number of parameters compared to the projected model size.

	$\mathcal{M}_1$	$\mathcal{M}_{1/2}$		$\mathcal{M}_{1/4}$		
$r =$	8	8	16	8	16	32
$\mathcal{M}_{1/4}$	**0.47**	0.50	0.49	–	–	–
$\mathcal{M}_{1/8}$	**0.57**	0.59	**0.57**	0.71	0.70	0.67

Table 3. Training losses for $\mathcal{M}_{1/4}$ and $\mathcal{M}_{1/8}$ models (rows) projected from $\mathcal{M}_1$, $\mathcal{M}_{1/2}$ and $\mathcal{M}_{1/4}$ source models (columns) with different values of the projector inner dimension r. The projection operator size for the source $\mathcal{M}_1$ with $r = 8$ ($\sim 2M$ parameters) approximately matches the projection operator sizes for $\mathcal{M}_{1/2}$ with $r = 16$ and $\mathcal{M}_{1/4}$ with $r = 32$.

	1	1/8	1/8 and 1
$1/2 - 1/2$ mod 2	**2.382**, **2.040** **2.221**, **2.201**	2.395, 2.052 2.233, 2.214	2.392, 2.045 2.225, 2.206

Table 4. Losses on C_1 and C_2 of task-specific projected GPT-2 models with $s = 1/2$ computed on the IMAGENET validation set ($50k$ samples; the estimated error is approximately 0.002). The rows correspond to two different task partitions across C_1 and C_2. Three columns show results for different sizes of the training sets for tasks in C_1 and C_2: (a) all tasks use all available training samples; (b) all tasks use $1/8$ of available training samples; (c) tasks in C_1 are trained with $1/8$ of available training samples and tasks in C_2 use all training samples.

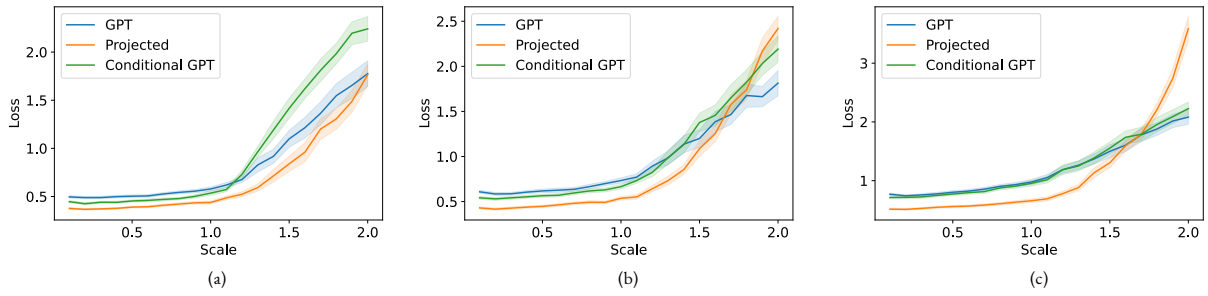

Figure 4. Average loss as a function of the scale component t_{scale} (plotted on the x axis) of the task identifier t with the following projected models: (a) $\mathcal{P}_{1/2}$, (b) $\mathcal{P}_{1/4}$, (c) $\mathcal{P}_{1/8}$.