Personalized Transformers for Everyone

Anonymous ACL submission

Abstract

Personalized Intelligence (PI) is the problem of providing customized AI experiences tailored to each individual user. A related problem is the *compartmentalization* of intelligence that maintains a partition between the personalized and the general models. Existing personalization approaches involve fine-tuning pre-trained models to create new customized models. However, these require a significant amount of computation to train, which scales with model size and the number of users, inhibiting PI to be realized widely. A compartmentalized approach enables a small model to be specialized for each individual user, which needs to be used together with a larger model to provide personalization. By separating personalized and general models, we enable higher accuracy, scalability, and stronger privacy guarantees. In this paper, we aim to design a compartmentalized personalization approach that can scale to millions of users and beyond. We investigate the landscape of model fine-tuning techniques and construct new design adaptations based on the requirements of PI. We then introduce Personalized Head (PH), a new model training/inference framework designed for scalable PI. We explore the design space of these techniques and evaluate their efficacy under various production-level constraints. Specifically, we break down the trade-off between accuracy, scalability and production deployment limitations. We present several production-ready personalization approaches suited for various production use case scenarios.

1 Introduction

Personalized Intelligence (PI) is the problem of providing unique and customized AI experience tailored for each individual user. Today's AI in production is often served with an unified model shared among all users and the experience remains largely homogeneous across users. However, certain problems are highly personal and the task scope can vary from user to user, which limits the effectiveness of this single model approach. For instance, in productivity software, users often use personalized category labels and tags to organize their to-do items. For the task of classifying a new unseen item to a category, a shared model is limited

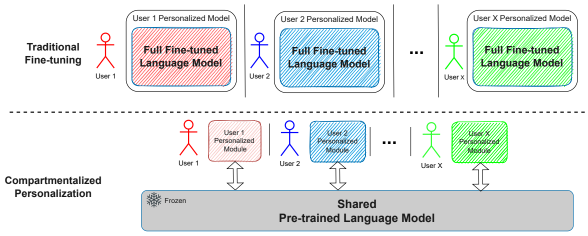


Figure 1: Full-model fine-tuning approach vs. compartmentalized personalization training approach.

as each user can have unique category labels and can interpret them differently. More PI use case examples are described in Section 2.

In order to capture the user-specific knowledge, PI tasks require personalized weights and module to learn from user-specific data. This presents several key requirements for solving PIs. First, the training of the personalized models needs to be lightweight and fast, as they often happen in an online settings. Second, the personalized models need to be parameter efficient. The amount of personalized models scales with the number of users, which can be millions or even billions, so the personalized model size needs to be small so they can be stored and served at large scale. Third, the personalized weights need to be compartmentalized from the original model, in order to protect the privacy of user data and maintain the integrity of each user's model quality.

The goal of this paper is to design an approach for achieving personalized intelligence at the production scale of millions of users and beyond. We examine a landscape of personalization techniques and aim to answer the following research questions:

1. How scalable can we achieve with the current landscape of compartmentalized personalization training techniques?
2. What are the design knobs of each technique and what kind of trade-off do they represent in the design space?
3. How do the techniques compare under various

production level constraints?

We investigate the current landscape of parameter-efficient fine-tuning techniques, including Adapters (Houlsby et al., 2019) and Prefix Tuning (Li and Liang, 2021). We identify key limitations of these approaches to be the compartmentalization of the personalized weights as they are intertwined with the layers and weights of the original model. In addition, these approaches are designed for task-level fine-tuning and it remains an open question whether they are scalable enough for creating user-level personalized models. To address these, we make two contributions. We first construct new design adaptations specifically for personalization, namely Personalized Adapter (PA) and Personalized Prefix (PP). Secondly, we propose a new model training and inference framework, Personalized Head (PH), specifically for personalization at scale. In PH, the personalized weights are completely separated from the original base model.

We explore the design space of all three techniques (PA, PP and PH) and compare their accuracy and scalability. We found that Personalized Prefix (PP) outperforms Personalized Adapter (PA) and Head (PH) when no deployment constraints are considered and all layers of the original model are augmented with new weights. On the other hand, under production level deployment constraints, there are several viable options across the three techniques, each with unique advantages depending on the use case.

2 Personalized Intelligence

We describe in details the concept of Personalized Intelligence and provide three concrete example use cases that can be found in production today.

2.1 Definition

Personalized Intelligence (PI) is the problem of creating unique and customized experience of an AI capability for each individual user. To provide experience tailored to each user, PI requires user-specific weights that are trained on user-specific data. The user-specific weights also serve as compartmentalization for the learnt knowledge of each user to not influence and affect other users' models and experience. PI requires every user-specific models to be query-able and incoming requests are routed accordingly based on their source. This creates significant scalability challenges for traditional fine-tuning approach. Specifically, it is not feasible to have full size model replicas for each user due to the ever increasing size of state-of-the-art models.

A similar problem that has been studied recently is task-specific fine-tuning (Houlsby et al., 2019; He et al., 2021; Li and Liang, 2021; Hu et al., 2021). While both problems involve adapting pre-trained models to new data, task-specific fine-tuning generates one model per task and only that model needs to be hosted and served to the users. In contrast, in personalized intelligence, unique weights and learning is required for each user and total number of query-able models scales with number of users. Furthermore, the task scope is fully defined at the model creation time for task-specific fine-tuning, while for personalized intelligence, part of the problem scope is unique for each user.

In summary, Personalized Intelligence tasks generally have one or more of the following characteristics:

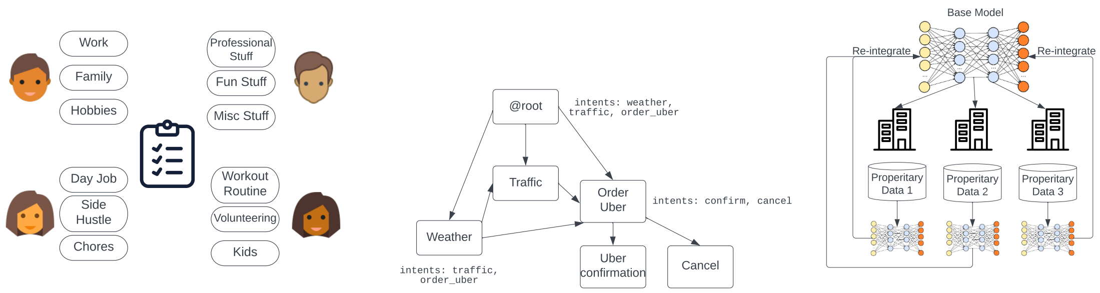
- Part of the problem scope is user-specific and not fully defined at model creation and pre-training time. The scope can also evolve overtime, pre- and post-production deployment.
- Unique weights are required on a user-by-user basis to capture the personalized knowledge of each user.
- The knowledge learnt from the user-specific data needs to be compartmentalized as to avoid potentially contaminating the behavior of other users' models and experience.

2.2 Example use-cases of PI

We describe three example production use cases of Personalized Intelligence, as illustrated in Figure 2.

Category Prediction in Productivity Tools - In productivity apps such as Todoist and OmniFocus, users are encouraged to create and organize their tasks into custom categories or tags (e.g., work, family, hobbies, etc). The task of classifying an item to a category/tag by learning from past user behavior is an example of Personalized Intelligence, as the category labels can be drastically different between users and are not fully defined beforehand (Figure 2a). A successful category prediction model needs to train on each user's data to capture user's personal preference. Furthermore, this personalized knowledge need to be compartmentalized to avoid affecting other users.

Intent Classification in Dialogue Systems - For intent classification in dialogue systems, each state has its own set of candidate intents corresponding to the set of next possible states, as shown in Figure 2b. Based on the user, the map of dialogue states can also be different and as a result, the scope



(a) Each user use unique task categories/tags to organize their items in a todo list app.

(b) State-aware classifier have unique candidate intents per state and per user.

(c) Federated learning creates customized model based on proprietary data.

Figure 2: Example use cases of Personalized Intelligence (PI) in production today.

of the intent classification for a given user is unique. Furthermore, the set of candidate intents generally evolve and grow as new topics and capabilities are introduced. A fully built-out dialogue system can feature hundreds or more states (Larson et al., 2019), which combined with the conversation paths unique to each user, can lead to a large number of unique classification problems within one dialogue system. A personalized intent classification model is customized to the context of each state and each user and trained to focus on the state's candidate intents and specific type of utterances encountered from a given user.

Federated Learning - The goal of federated learning is to train a central model based on data that is distributed over a large number of clients (Konečný et al., 2016) (Figure 2c). Clients independently compute model updates based on local data, and client-side updates are eventually aggregated to update the central model. The original motivation for this technique was driven by the needs of distributed systems, where learning takes place based on data at the edge (e.g. in mobile phones) which may not always be connected to the central model. Follow-on work (Gao et al., 2019) emphasized the privacy-preserving aspects of federated learning, as the training data itself does not need to leave its original location at the edge. Personalized Intelligence techniques can be used to support federated learning use-cases, as the information contained in each personalized model can be transferred to the general model, if needed, under user control. Instead of assuming that all knowledge learned at the edge is going to end up in the general model, PI approach emphasizes privacy-preservation and compartmentalization to provide user-specific inference.

3 Training for Personalization

We define the process of training personalized models. Let M be a pre-trained language model (LM) and Θ_M its trainable parameters. Consider the scenario of fine-tuning for an individual user to create a personalized model. We define $U = \{(D_1, L_1), (D_2, L_2), ..., (D_N, L_N)\}$ as the collection of personalized tasks where $i \in [1, ..., N]$ for N users, D_i is the unique data for user i and L_i is the loss function.

3.1 Traditional Fine-Tuning

When applying model M to a downstream task T with labelled training data D_T, an output layer K with parameters Θ_K is usually appended to M. Then M and K are trained jointly:

$$\Theta\prime_M, \Theta\prime_K \leftarrow \arg\min_{\Theta_M, \Theta_K} L_T(D_T; \Theta_M, \Theta_K) \quad (1)$$

This generates $\Theta\prime_M$ and $\Theta\prime_K$. $\Theta\prime_M$ is the fine-tuned parameters of M, which are the same size as Θ_M but with distinct values. Let $\Omega(\Theta)$ be the computation complexity required to train a set of parameters Θ. We then define the training complexity of the above fine-tuning operation as:

$$\Omega(\Theta\prime_M) + \Omega(\Theta\prime_K) \quad (2)$$

The problem of fine-tuning the model to personalize for user i is defined as

$$\Theta\prime_{M,i}, \Theta\prime_{K,i} \leftarrow \arg\min_{\Theta_M, \Theta_K} L_i(D_i; \Theta_M, \Theta_K) \quad (3)$$

The aggregated training complexity scales linearly with the number of users, N:

$$\sum_1^N \Omega(\Theta\prime_{M,i}) + \sum_1^N \Omega(\Theta\prime_{K,i}) \quad (4)$$

The collection of model parameters to be stored scales linearly with N as well:

$$\sum_{1}^{N} |\Theta\prime_{M,i}| + \sum_{1}^{N} |\Theta\prime_{K,i}| \tag{5}$$

3.2 Compartmentalized Personalized Training

When training compartmentalized and personalized model for user i, a new set of weights W_i are introduced. During training, the base LM parameters are frozen and only W_i and K_i are updated:

$$\Theta\prime_W, \Theta\prime_K \leftarrow \underset{\Theta_W, \Theta_K}{\arg\min}\ L_T(D_T; \Theta_M, \Theta_W, \Theta_K) \tag{6}$$

Note that, compared to the traditional fine-tuning defined in Equation 1, no $\Theta\prime_M$ is generated. During inference, $\Theta\prime_W, \Theta\prime_K$ is combined with the original Θ_M to generate prediction. The aggregated training complexity for training the personalized model is:

$$\sum_{1}^{N} \Omega(\Theta\prime_{W,i}) + \sum_{1}^{N} \Omega(\Theta\prime_{K,i}) \tag{7}$$

and the total parameters is:

$$\Theta_M + \sum_{1}^{N} |\Theta\prime_{W,i}| + \sum_{1}^{N} |\Theta\prime_{K,i}| \tag{8}$$

The main goal for a scalable personalization approach is minimizing size of W_i, which will reduce the model size and training cost and increase the maximum users supported N given a certain amount of compute resources, and optimizing for prediction accuracy on the collection of personalized tasks U.

4 Personalization Model Architectures

We describe the landscape of three personalization techniques studied in this work, Personalized Adapter (PA), Personalized Prefix (PP) and Personalized Head (PH), as illustrated in Figure 3.

4.1 Personalized Adapters

Personalized Adapter (PA) is constructed based on the Adapter approach, which involves inserting small trainable feedforward layers into every layer of the base transformer model (Houlsby et al., 2019; Pfeiffer et al., 2021; Hu et al., 2021). During training, the inserted adapter layers are updated while the base transformer is frozen. In the context of Personalized Intelligence, the out-of-the-box adapter approach are not compartmentalized because the new layers are interleaved with the layers of the original model. This limits adapter's applicability to training for personalization. Specifically, it is technically challenging to have many adapters sharing the same base model for inference because the inference execution flow switches back and forth between the original model and the adapter.

To address this limitation, we formulate Personalized Adapter (Figure 3a), where only a subset of transformer layers are augmented with adapters. This way, in production, the augmented layers can be replicated for each user to create compartmentalization and the untouched layers in the original models can be shared across users for inference. In order to understand the impact of selectively applying Adapters, we construct a range of different PA configurations and evaluate their trade-offs in Section 6.2.

4.2 Personalized Prefix

The Prefix tuning approach prepends trainable weight vectors to the keys and values weight matrices of the multi-head attention block in each transformer layer (Li and Liang, 2021). The new prefix vectors are used in the attention calculation for attention heads of every transformer layer. Similar to Adapters, Prefix tuning has the same compartmentalization limitation and we construct **Personalized Prefix**, where selected layers are augmented with the prefix vectors, as shown in Figure 3b. We experiment with a range of different PP configurations in Section 6.3.

4.3 Personalized Head

Inspired by the above approaches and considering their limitation, we propose a new framework, Personalized Head (PH). We propose to append a single layer transformer module after the base transformer. During training, only the PH is updated and the base model weights are frozen. During inference, the base model processed the input first and the output embeddings are sent to the PH to apply the personalized knowledge and generate the output. Our intuition is we can aggregate and focus the personalized knowledge into the PH module and avoid augmenting any of the base model layers. Figure 3c shows an overview of the PH architecture.

PH follows the Transformer architecture defined in the original transformer paper (Vaswani et al., 2017). Each PH has a multi-head self-attention layer and two fully connected layers, followed by

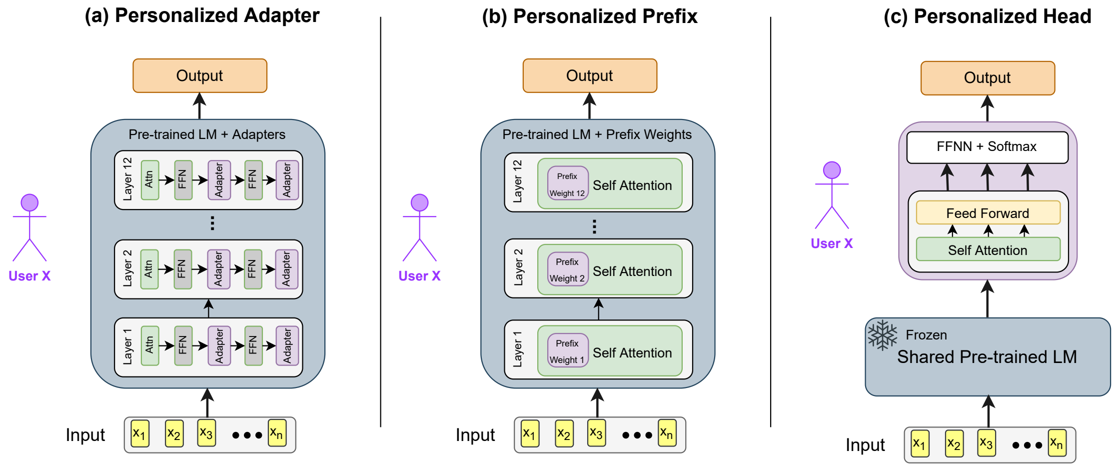


Figure 3: Three personalization approaches. The purple colored block in each approach represents the personalized module for each user. (a) Personalized Adapter insert adapter layers inside selective transformer layers; (b) Personalized Prefix augment the K,V weight matrices of the self-attention block with new personalized weights in selective transformer layers; (c) Personalized head attach a new personalized transformer layer after the original transformer.

Dataset	Description	# Classes	# Train	# Test
SNIPS	Smart assistants questions	7	13,034	1,442
Clinc150	Production VA tasks	150	15,100	1,500

Table 1: Datasets

layer normalization (Ba et al., 2016). Dropout (Srivastava et al., 2014) is applied to the output of the fully connected layers.

To help us explore the PH design space and understand its key design factors, we parameterize the size of the hidden dimension of the feed-forward network in the encoder and the number of attention heads in the attention layer. We investigate the impact of these design decisions in detail in Section 6.4.

5 Experiments

To investigate the effectiveness of these techniques, we apply them to personalize the pre-trained BERT LM to two new classification tasks as the personalized problems. We keep the BERT LM frozen during personalized training and apply each technique separately to train on the new data. In this section, we describe in details the experiments and dataset setup.

5.1 Universal Binary Classification Task

We focus on personalized classification as the PI task to evaluate the suite of personalization techniques. We aim to design a personalization framework that is generalizable to arbitrary classification tasks without requiring modification to the model architecture. To that end, we formulate the multi-class classification problem as a series of binary classification tasks. We concatenate the class label and the text as input and the output layer generates a binary True/False prediction with a confidence score. The class with the most confident True prediction is selected as the classification prediction. We apply this binary classification across all personalization approaches.

5.2 Datasets

We use the SNIPS dataset (Coucke et al., 2018) and Clinc-150 dataset (Larson et al., 2019). We select SNIPS because its intents cover many common classification topics and it is a representative dataset widely studied in the literature. We select Clinc-150 for its focus on the complexity of production use cases. It has 150 intents and features intents and sentences inspired by real virtual assistants in production. An overview of the datasets is shown in Table 1. Specifically for Clinc, to make training and evaluation time more manageable, we randomly sample 10 examples (out of 30) per class to construct the test set and include the True examples and randomly sample 2 False examples for every True example to construct the training set. The same train and test set construction is used across all techniques and experiments.

5.3 Implementation

We use the AdapterHub framework (Pfeiffer et al., 2020) for the experiment on Adapters and Prefix Tuning. We implement the PHs using the Flair NLP framework (Akbik et al., 2019) with an underlying pytorch runtime (Paszke et al., 2019). We use the uncased BERT encoder as the base LM (Devlin et al., 2018) for all three approaches. We train for 50 epochs (unless noted otherwise) and report F1 score and accuracy on the test set. In terms of hyper-parameters for PH, we use a batch size of 16 and a learning rate of 0.02, following the standard in (Halder et al., 2020).

6 Results

We evaluate the landscape of personalization techniques, including Personalized Adapters (PA), Personalized Prefix (PP) and Personalized Head (PH). We aim to understand 1) how scalable can we push these techniques to be, 2) what are the design knobs that have the most impact on performance and 3) how do these techniques compare under production-level constraints.

6.1 Naive Personalization Approaches

Table 2 shows the results for several naive personalization approaches. **Full Model Fine-tuning -** Fine-tuning both LM and linear layer achieves strong performance but it requires, for each user, the training of 109 million parameters which generates a 417MB model. The large computation cycles and storage capacity becomes untenable for production use cases that require scaling to many (millions) users.

Personalization via the linear layer only - We experiment with only training the linear classification layer. It achieves F1-score of 71.12 for SNIPS and 52.43 for Clinc, significantly underperform the full fine-tuning approach but only requiring per user 1.5K parameter.

Personalization via Zeroshot - We also investigate the efficacy of the zeroshot approach. We use the TARS zeroshot classifier from (Halder et al., 2020), which uses BERT as the underlying language model and is pre-trained on a suite of datasets including AGNews and DBPedia (Zhang et al., 2015). We test the zeroshot performance of both the trained TARS classifier and BERT out-of-the-box. BERT(OOB) achieves F1-score of 2.99 and 0.78 on SNIPS and Clinc, respectively, and the TARS classifier achieves an F1-score of 35.27 and 23.98 on SNIPS and Clinc, respectively, which are significantly lower than the reported state-of-the-art results. This shows that zeroshot approach requires significant improvement to reach the level of performance needed for production usage.

6.2 Personalized Adapter

Table 3 shows the performance of Personalized Adapters on SNIPS and Clinc. As described in Section 4, we construct Personalized Adapters by selectively insert adapters to a subset of layers in the transformer. We observe that the adapter's ability to learn personalized knowledge depend heavily on the number and location of the inserted adapters. When only adding adapters to a selected set of layers, we see that inserting adapters to early layers achieve better performance compared to later layers, given the same number of adapters. Specifically, inserting adapters for the first half of the BERT model (1-6 layers) outperform inserting adapters to the second half (7-12). Moreover, if the situation only allows for a single transformer layer to be augmented with new weights, adding adapter to first layer outperforms the last layer.

Limitation and Trade-off for Production In Personalized Adapters, personalized layers and original layers are interleaved and the model needs to switch between them many times during an inference. This makes it challenging to have many PAs live in production to share the same base model. In addition, the tightly integrated models require the personalized weights and the original model to be co-located in the same environment, limiting the possible deployment configurations, e.g. deploying personalized weights on user's devices and sharing the base model in the cloud. When only augmenting selective layers, those layers can be replicated across users to create a more compartmentalized approach, at the cost of accuracy. Under this approach, the most scalable design is applying PAs on the first layer or first several layers.

6.3 Personalized Prefix

Table 5 shows the result of Personalized Prefix. Similar to PA, we construct variations of Personalized Prefx by applying prefixes to different layer combinations. Overall, PP achieves similar performance as PA. However, PP requires significantly more parameters per user. In addition, at lower parameter count, PA outperforms PP with less parameters per user. We also observe that applying prefixes at early layers of the transformers yields higher accuracy than early layers, similar to PA.

Limitation and Trade-off for Production Personalized Prefix approach requires 9.87M parameters (37MB) per user when all layers are augmented. This is a substantial amount of parameters when considering potentially scaling to millions of users. However, this technique has the unique property

Naive Personalization Approach		# Params / User	Size / User	SNIPS F1 / Acc.	Clinc F1 / Acc.
ZeroShot	BERT	NA	NA	2.99 / 1.60	0.78 / 0.47
	TARS	NA	NA	35.27 / 26.70	23.98 / 23.67
Fine-tuning LM + Linear Layer	BERT	109M	417MB	98.61 / 98.61	95.74 / 95.07
	TARS	109M	417MB	98.13 / 98.06	95.22 / 94.27
Fine-tuning Linear Layer Only	BERT	1.5K	7KB	68.70 / 58.67	52.43 / 50.27
	TARS	1.5K	7KB	71.12 / 63.11	33.27 / 33.20

Table 2: F1 score and accuracy of zeroshot, fine-tuning the LM + linear head and fine-tuning the linear head only, evaluated on SNIPS and Clinc-150 datasets. We also show the number of parameters that are required to be fine-tuned for each approach, as well as the size of the personalized model to be managed for each user as a representation of the scalability of each approach.

Personalized Adapter	# Params / User	Size / User	SNIPS F1 / ACC.	Clinc F1 / ACC.
Full (1-12)	894K	3.6MB	99.02 / 99.02	89.13 / 89.13
1st half (1-6)	447K	1.8MB	98.19 / 98.19	89.26 / 89.26
2nd half (7-12)	447K	1.8MB	98.54 / 98.54	89.33 / 89.33
last layer (12)	74.5K	0.3MB	97.22 / 97.22	64.80 / 64.80
first layer (1)	74.5K	0.3MB	98.06 / 98.05	82.00 / 82.00

Table 3: Performance and size of personalized adapters.

Personalized Prefix	# Params / User	Size / User	SNIPS F1 / ACC.	Clinc F1/ ACC.
Full (1-12)	9.87M	37MB	98.95 / 98.95	91.80 / 91.80
1st half (1-6)	4.94M	19MB	98.74 / 98.75	89.40 / 89.40
2nd half (7-12)	4.94M	19MB	98.13 / 98.12	88.70 / 88.60
last layer (12)	823K	3.1MB	95.61 / 95.63	61.04 / 61.06
first layer (1)	823K	3.1MB	97.45 / 97.43	80.55 / 80.53

Table 4: Performance and size of personalized prefix.

where the prefix weights can be stored on the user side and then pass to the base model along with the input thus achieving complete compartmentalization of the personalized weights (Li and Liang, 2021). In a production settings, each user's prefix weights can be stored in a separate database and during inference, first fetch the prefix weights and then send it to the shared base model. Note that this approach introduces an additional inference latency overhead for the step of fetching the prefix weights for the querying user.

6.4 Personalized Head

For PH, we experiment with a wide range of configurations by varying the hidden dimension of the feed-forward layer and the number of attention heads in the PH. For brevity, we include results for configurations with hidden dimensions from 128 to 2048 and # attention heads of 2, 4, and 8. Results for additional configurations are included in the Appendix. We observe that PHs, across all configurations, significantly outperform fine-tuning only the linear layer on both datasets. When comparing to the baseline of fine-tuning the entire model stack of the base language model and the linear output layer, PH achieves similar results for the SNIPS dataset, while requiring orders of magnitude less training cost. PH performs similarly as PA and PP on SNIPS, while underperforms the previous two approaches on Clinc, which is a more difficult dataset. As for scalability, PH requires less parameters than all of PP configurations except for single layer prefix and requires more parameters than PA. We have included a deep dive into how to effectively design and train a PH in the Appendix.

Limitation and Trade-off for Production PHs are completely detached from the base model and inference uses both models in combination to generate its prediction. By having a simple and clear boundary between the base model and the personalized, PH makes it trivial to share the base model among many users, using existing out-of-the-box model implementation. It also enables more flexible deployment options, such as cloud and edge collaboration which has been shown to improve latency and energy efficiency (Kang et al., 2017), where PH is deployed on user's own mobile/edge device while keeping the large-scale base model in the cloud.

7 Discussion

We compare across the three personalization techniques and summarize their advantages and disadvantages in different production scenarios.

Personalized Adapters (PA) is the most parameter-efficient approach, achieving the highest accuracy per parameter. It also slightly outperforms the other two techniques and the full fine-tuning baseline for the simpler SNIPS dataset. The disadvantage of this approach is the difficulty to share one base model across many users when multiple layers are augmented with PA. Therefore, an option here to use the last layer PA variant and replicate it across users, when dealing with low complexity personalization tasks and compute resources is limited.

Personalized Prefix (PP) achieves the highest accuracy among all three approaches but it requires

Model			# Params / User	Size / User	SNIPS F1 / ACC.	Clinc F1 / ACC.
	Hidden Dim	# Attn. Heads				
Personalization Head (PH) w/ frozen LM	2048	8	5.52M	21MB	96.52 / 96.12	76.36 / 75.93
		4			97.18 / 96.74	76.46 / 76.47
		2			97.33 / 97.16	76.36 / 75.93
	1024	8	3.94M	15MB	97.46 / 97.09	75.07 / 74.40
		4			96.76 / 96.39	75.82 / 75.73
		2			96.77 / 96.67	76.61 / 76.60
	512	8	3.15M	12MB	97.05 / 97.02	75.95 / 76.33
		4			96.26 / 95.49	70.79 / 71.20
		2			96.94 / 96.74	75.29 / 75.27
	256	8	2.76M	11MB	95.90 / 95.70	66.99 / 67.53
		4			95.64 / 95.15	68.64 / 67.87
		2			97.06 / 96.32	67.68 / 67.13
	128	8	2.57M	9.8MB	95.32 / 95.28	63.43 / 64.53
		4			96.32 / 96.32	62.70 / 63.67
		2			96.36 / 96.36	63.82 / 64.60

Table 5: F1 score and accuracy of personalization head.

the most parameters per user. The accuracy advantage is most prominent on harder tasks, such as Clinc-150. Therefore, Personalized Prefix is the preferred solution for more sophisticated personalization tasks with less demanding scalability requirement (e.g., smaller user count and/or more compute resources available).

Personalized Head (PH) represents an unique position among the three personalization techniques. It achieves better or similar performance than PA and PP on the SNIPS dataset and under-perform them on the Clinc dataset. Because of its complete compartmentalized design, PH is the easiest to deploy for production. It also enable on-device deployment and training of the personalized module, which protect privacy of user's data, while the other techniques require the personalized weights to co-locate with the base model.

8 Related Work

One of the most common solutions to many natural language understanding problems today is leveraging large-scale pre-trained transformer-based language models (Devlin et al., 2018; Liu et al., 2019) which are typically trained on language understanding objectives such as Masked Language Modeling and Next Sentence Prediction. These language models are then fine-tuned for a specific task. This transfer of learning to the task of interest is achieved by tuning all model weights on that new task. The performance of these LMs have shown to scale with model size (Kaplan et al., 2020), resulting in massive models consisting of billions of parameters (Brown et al., 2020; Raffel et al., 2020; Sanh et al., 2021). When applied to the online setting of personalization training, the applicability of these language models is severely constrained as it results in a dedicated model for each user. This section explores existing works improving the applicability of transformer models at scale.

Zero-shot learning approaches aim to provide generalized models for a range of language-based tasks without needing additional training steps required by traditional transfer learning approaches. Recent approaches to this problem frame this as a text-to-text generation task (Brown et al., 2020; Raffel et al., 2020; Sanh et al., 2021) with focuses on prompt design (Perez et al., 2021; Khashabi et al., 2020). While shown to be effective in tasks such as QA and summarization, zero-shot performance is still lacking when it comes to text classification (Halder et al., 2020; Wenpeng Yin and Roth, 2019). This is further shown in our zero-shot experimental results. Halder et al. (2020) explores the shortcomings of the existing transfer learning mechanisms for text classification, proposing the formalization of text classification as a general binary classification problem.

9 Conclusion

In this work, we introduce and define a new research problem, Personalized Intelligence (PI). PI is the problem of creating customized AI experience that is tailored to each individual user. PI brings a new host of challenges for existing fine-tuning techniques, increasing the pressure on model scalability. We examine a landscape of personalization techniques and investigate their performance and trade-off and present limitations and considerations for using each technique under production level constraints. Finally, we compare across all techniques to provide concrete recommendations on the best approach for Personalized Intelligence given various production scenarios.

References

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. Flair: An easy-to-use framework for state-of-the-art nlp. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dashan Gao, Yang Liu, Anbu Huang, Ce Ju, Han Yu, and Qiang Yang. 2019. Privacy-preserving heterogeneous federated transfer learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2552–2559.

Kishaloy Halder, Alan Akbik, Josip Krapac, and Roland Vollgraf. 2020. Task-aware representation of sentences for generic text classification. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3202–3213.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *CoRR*, abs/2110.04366.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. Unifiedqa: Crossing format boundaries with a single qa system.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*.

Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. 2019. An evaluation dataset for intent classification and out-of-scope prediction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1311–1316, Hong Kong, China. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True few-shot learning with language models.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020): Systems Demonstrations*, pages 46–54, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. 2021. Multitask prompted training enables zero-shot task generalization.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Jamaal Hay Wenpeng Yin and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *EMNLP*.

Jack Schofield ZDNet. 2012. Is your Gmail really worth $3,600? Backup now! `https://www.zdnet.com/article/is-your-gmail-really-worth-3600-backup-now/`. [Online; accessed 16-Jan-2022].

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*.

A Appendix

A.1 How to effectively train a PH

We conduct experiments aimed at understanding the learning behavior of PHs and gain insights into how to design and deploy an effective PH for real-world use cases. We aim to answer the following questions: 1) How to effectively train PHs in production? 2) How does the PH configuration affect its learning behavior? 3) Do larger PHs achieve better performance and does there exist a sweet spot of PH design that is the most compute and data efficient?

Impact of Data vs. Epoch on Training PHs

We study the impact of the scale of training data and the number of training epochs on the PH performance. In real-time training in production, there is often limited training data and training cycles available. Therefore, it is imperative to understand how to train a PH in a compute and data efficient manner. To this end, we construct a SNIPS sub-dataset by random sampling 100 training examples per class (1400 samples in total) and keep the full SNIPS test set. We train the spectrum of PH designs for 50 epochs and then 50 more epochs (100 epochs in total) and record the test set F1-score at both points. We then select another 100 training samples per class to add to the training set and repeat the same experiment. Table 6 shows the average F1 scores, as well as improvement gained by increasing training data, training epochs, and both.

We observe that increasing the training data from 100 per class to 200 per class provides a significantly higher F1 score increase (+19.85 on average), compared to training for more epochs (+5.98 average). This behavior is consistent across the various PH configurations. This is intuitive because 100 training samples/class represents only 5.3% of the full SNIPS training set and does not provide robust coverage of the problem space. Increasing the training data scale should be the priority over more training iterations in the early stages of applying a PH to a personalized problem.

PH Design Analysis Two main design choices for PHs are the hidden dimension of the encoder block and the number of attention heads in the multi-attention layer. We study how these design choices impact the learning behavior of the PH. We conduct a set of experiments where we gradually increase the amount of training data or epochs and measure the F1-score at each stopping point. This is to simulate a training setup in production, where the model gradually gets exposed to more training data as the applications collect more personalized data from the users.

Hidden Dimension Size Figure 4 shows the F1 score of PHs with different hidden dimensions as they are trained with more epochs on the same amount of training data. We experiment with 50, 100, 150, and 200 training examples per class for 25, 50, 75, and 100 epochs. Conversely, Figure 5 shows the F1 score of the same suite of PHs as they are trained with more training examples for the same number of training epochs. We make several observations.

First, we observe that when exposure to training data is limited in the early stages of training, PH training can exhibit unpredictable behavior. This is shown in the leftmost graphs of Figure 4 and 5, where the model performance is not improved with additional training data or more training epochs.

Furthermore, larger PHs perform better than smaller PHs but with diminishing returns at higher ends, indicating a sweet spot of PH design. We observe 512 to be the sweet spot of PH design for SNIPS as it performs better or similar to the other configurations across all experiments. This finding is corroborated with results on the Clinc-150 dataset. Figure 6 shows the F1 score of PHs with varying hidden dimensions on both SNIPS and Clinc-150. We observe similar trends for diminishing return in performance for Clinc as the PH design gets larger. Similarly, 512 is the inflection point of F1 score improvement, making it the sweet spot PH design for Clinc. Furthermore, we observe a slower rate of F1 score improvement with respect to hidden dimension size for Clinc than SNIPS. This can be explained with the observation that Clinc is a more diverse and challenging task with significantly more classes than SNIPS, as described in Section 5.2,

# of Attention Heads We study the impact of attention heads on PHs performance. Figure 7 shows F1 score w.r.t hidden dimensions per attention head. We follow the design in (Vaswani et al., 2017) where the hidden dimensions of the feed-forward layer are effectively distributed evenly among the available attention heads. We observe that PHs achieve better performance with higher hidden dimensions per head but eventually see diminishing returns.

A.2 Scalability

We study the scalability of the PH approach and its impact on production deployment. To help us holistically evaluate a personalization approach,

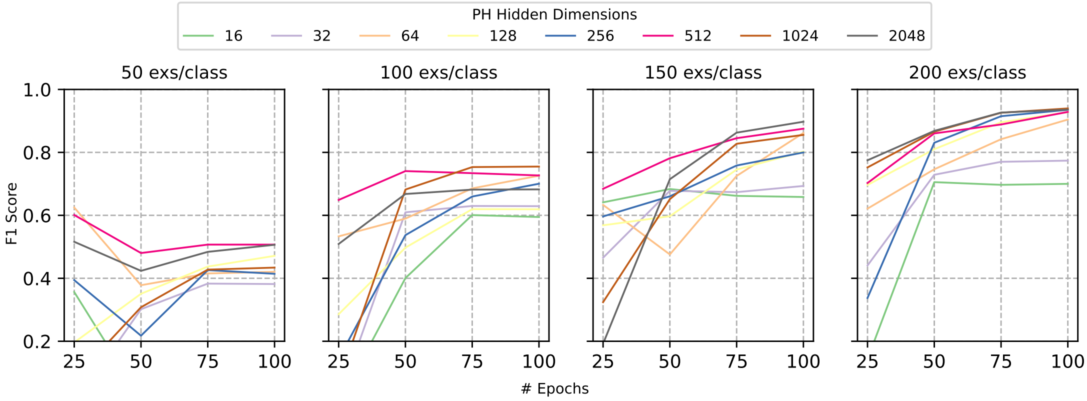


Figure 4: F1 Score w.r.t # training epochs, for fixed amounts of data.

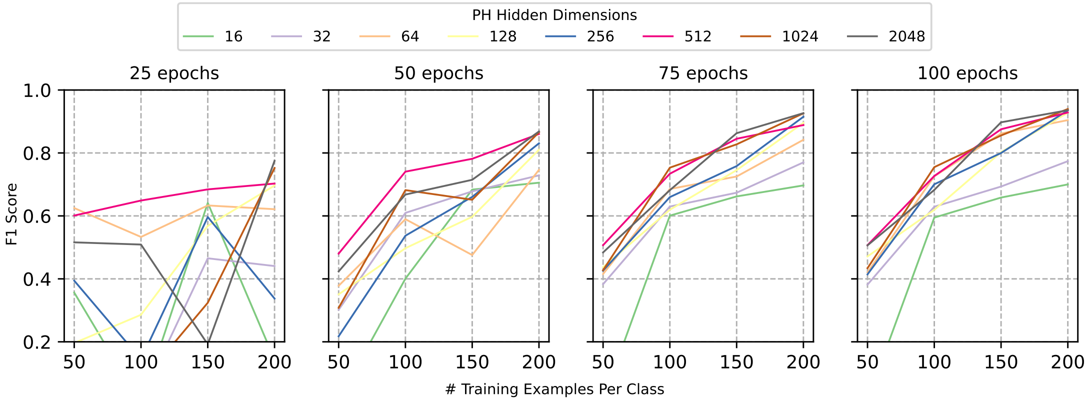


Figure 5: F1 Score w.r.t # training data, for fixed amounts of epochs.

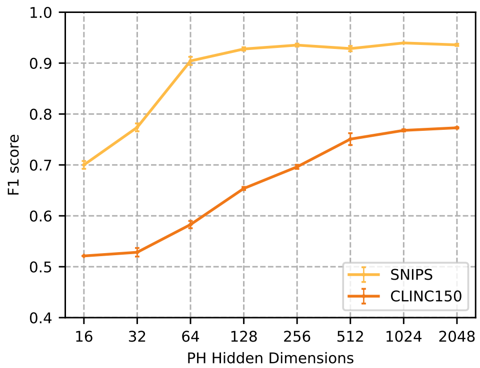


Figure 6: F1 score of PH w.r.t. hidden dimension size

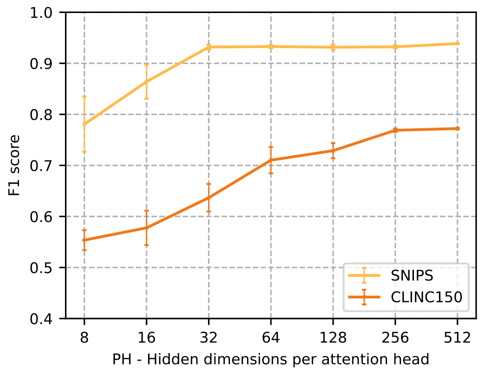


Figure 7: F1 score of PH w.r.t. hidden dims per head

we first introduce a new metric, Personalization Efficiency (PE):

$$PE = \frac{F-score^2}{Training\ Cost \times Model\ Size} \qquad (9)$$

This new metric considers both the model performance and the computation requirements of training and inference. We use the number of trainable parameters as an approximation for training cost in this study. Figure 8 shows the efficiency of 4 PH configurations normalized to the fine-tuning BERT baseline. We show that PHs achieve efficiency up to 155X compared to the fine-tuning baseline.

Furthermore, for SNIPS we observe smaller PHs generally measure higher in efficiency than larger PHs but see a diminishing return. For Clinc, 512 achieves the highest efficiency of the PHs tested. This corroborates our recommendation earlier that

Hidden Dims	# Params	Size	epoch=50 # data/class=100	epoch=100 # data/class=100	epoch=50 # data/class=200	epoch=100 # data/class=200
2048	2.7M	10.5MB	65.92	70.43 (+4.50)	78.12 (+12.20)	92.99 (+27.70)
1024	3.2M	12.0MB	57.42	65.39 (+7.97)	80.17 (+22.75)	93.72 (+36.30)
512	3.9M	15.0MB	61.75	67.01 (+5.26)	83.72 (+21.97)	93.99 (+32.24)
256	5.5M	21.0MB	57.40	63.58 (+6.18)	79.85 (+22.45)	94.58 (+37.18)

Table 6: F1 score (and differential) with increasing training data and/or training for more epochs

512 is the sweet spot for PH design.

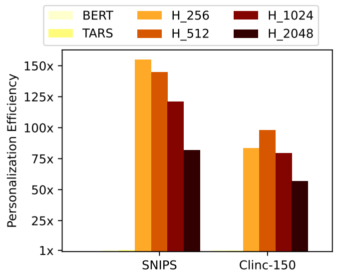


Figure 8: Personalization Efficiency

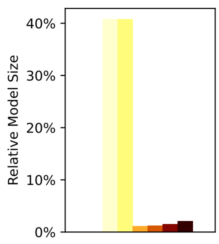


Figure 9: Data Scalability

We also quantify the potential storage overhead required for personalized models. Figure 9 shows the additional storage overhead required by the personalized models per individual user relative to the existing user data in production. We use Gmail as an example application. We calculate approximately the current per-user data usage based on a report that Gmail user creates up to 1.4MB of data per day and 3 years is the average account lifetime (ZDNet, 2012). Figure 9 shows that the proposed PHs constitute 1% - 1.5% of additional storage overhead across all 4 sizes, while the fine-tuning baselines would incur around 40% additional storage overhead per user.

A.3 More PH configurations

hidden dim	# attn. heads	SNIPS	Clinc
64	8	95.89	57.85
	4	96.32	54.41
	2	96.03	56.60
32	8	96.32	51.37
	4	96.24	50.83
	2	95.15	46.92
16	8	94.87	49.92
	4	94.43	49.99
	2	94.94	50.28
8	8	83.09	52.58
	4	80.86	52.66
	2	86.34	53.29

Table 7: F1 score of PHs with smaller sizes.

Table 7 shows the F1 score of PHs with 64, 32, 16, and 8 hidden dimensions, on SNIPS and Clinc datasets. This is an extension to the result shown in Table 5. We observe similar trends carry over to this set of even smaller PHs. This shows that even a tiny PH can adapt LM well to the SNIPS task. On the other hand, the smaller PHs are not as effective for the more challenging Clinc datasets.

A.4 Training for more epochs on Clinc

Table 8 shows the F1 scores of PHs of 4 different sizes on Clinc when training for an additional 50 epochs (100 epochs in total). This shows that PHs performance continues to improve with more training iterations, indicating that continuing training for more iterations are beneficial in improving PH performance.

hidden dim	# attn. heads	Clinc
2048	2	78.29
	4	77.25
	8	77.78
512	2	75.82
	4	75.05
	8	74.98
128	2	65.29
	4	65.72
	8	63.63
32	2	51.35
	4	53.24
	8	52.84

Table 8: F1 score of PHs trained on Clinc-150 dataset for an additional 50 epochs (100 epochs in total)

A.5 Analyzing # of attention heads

Figure 10 and 11 analyze the impact of # attention heads on the performance of PHs. We conduct experiments similar to that in Section A.1. We gradually increase the amount of training data while holding the training epochs fixed and measure the F1 score at each stopping point, and vice versa. We observe that, compared to hidden dimension sizes, # of attention heads has less effect on the learning behavior and capacity of the PHs.

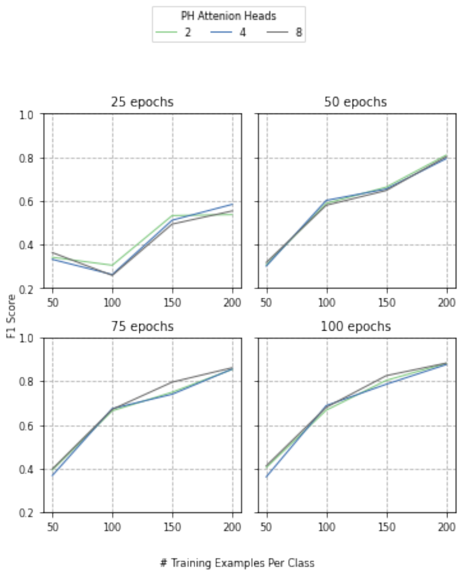


Figure 10: With the same number of training epochs, the impact of training data on performance.

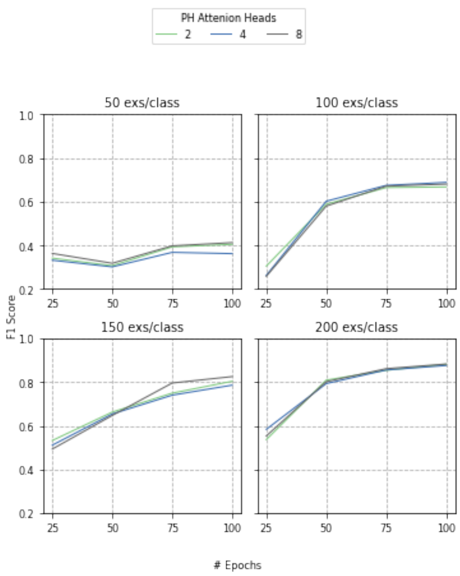


Figure 11: With the same amount of training examples per class, the impact of epochs on performance.