A Semantic-Aware Layer-Freezing Approach to Computation-Efficient Fine-Tuning of Language Models

Jian Gu
Monash University
jian.gu@monash.edu

Aldeida Aleti
Monash University
aldeida.aleti@monash.edu

Chunyang Chen
Technical University of Munich
chun-yang.chen@tum.de

Hongyu Zhang
Chongqing University
hyzhang@cqu.edu.cn

Abstract

Finetuning language models (LMs) is crucial for adapting the models to downstream data and tasks. However, full finetuning is usually costly. Existing work, such as parameter-efficient finetuning (PEFT), often focuses on *how to finetune* but neglects the issue of *where to finetune*. As a pioneering work on reducing the cost of backpropagation (at the layer level) by answering where to finetune, we conduct a semantic analysis of the LM inference process. We first propose using transition traces of the latent representation to compute deviations (or loss). Then, using a derived formula of scaling law, we estimate the gain of each layer in reducing deviation (or loss). Further, we narrow down the scope for finetuning, and also, study the cost-benefit balance of LM finetuning. We perform extensive experiments across well-known LMs and datasets. The results show that our approach is effective and efficient, and outperforms the existing baselines. Our approach is orthogonal to other techniques for improving finetuning efficiency, such as PEFT methods, offering practical values on LM finetuning.

1 Introduction

With the rapid advancements and notable performance of language models, their application has extended to numerous downstream tasks (Bommasani et al., 2021). Fine-tuning techniques are pivotal in augmenting the capabilities of language models (Raffel et al., 2019; Ouyang et al., 2022). For example, CODE LLAMA is a code-specialized LM and is finetuned on 100B tokens of Python code for a language-specialized variant (Touvron et al., 2023; Rozière et al., 2023). The Python variant provides better capabilities in code understanding and generation, since Python is most popular in programming (Carbonnelle, 2024; TIOBE, 2024).

Compared to their smaller pretrained predecessors, finetuning large LMs offers both advantages and disadvantages. On one hand, the vast number of model parameters triggers the emergent abilities of large LMs (Wei et al., 2022), leading to superior performance across a variety of tasks, which serves as an excellent foundation for domain-specific finetuning. On the other hand, the extensive parameter size presents challenges for downstream finetuning. For instance, large LMs require greater memory costs and higher computational costs in finetuning.

The challenge is on finding the correlation between performance and efficiency of LM finetuning. There have been developed techniques such as model quantization and PEFT methods to improve efficiency (Rokh et al., 2022; Han et al., 2024). Model quantization reduces the precision of the model and data to reduce the burden of storage and computation. However, the performance of LM finetuning may be damaged to some extent. Most PEFT methods introduce additional parameters to learn the updates in LM finetuning, and then merge the updates into the LM. Their focus is to reduce the memory cost instead of the computational cost (Han et al., 2024). Overall, there has been relatively little work exploring the correlation between model performance and computational efficiency, that is, whether the performance of LM finetuning can be improved while saving computation cost. To mitigate the gap, we propose utilizing the semantics in LM latent space to specify the layers that are more in need of finetuning being trainable, and freeze other layers.

Our intuition is that, by interpreting the LM's functionality as a transition of semantics and comparing it with a set of special latent representations, we can estimate the gains of each layer in reducing deviations. The deviations can be used to evaluate the convergence degree of model layers, and further, as the evidence to decide which layers shall be trainable. Based on empirical experience and theoretical analysis, the deviations in semantic transitions greatly decide the effects of LM finetuning. By freezing model layers with the maximum gains

Findings of the Association for Computational Linguistics: ACL 2025, pages 8019–8033
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

in reducing deviation and shortening the process of backpropagation, the computation cost may be reduced and meanwhile, the finetuning effects can be improved. Computational-efficient finetuning via layer-freezing is orthogonal with existing techniques, including model quantization and PEFT methods, so can combine with these techniques to achieve more efficient performance.

In this paper, we realize computation-efficient model finetuning by proposing an effective and reliable layer-freezing approach, referred to as Semantic-Aware Layer-Freezing (SALF). First, on the shoulder of LM semantics (Gu et al., 2024, 2025), we study the phenomenon of semantic transition in LMs. By deriving the scaling law of LM pretraining, we estimate the gains of reducing deviations in each model layer; Next, our layer-freezing approach finds the model layer whose gain is the maximum and only finetunes the deeper layers; Last, to support a flexible cost-benefit tradeoff in LM finetuning, we propose a shallow-to-deep policy for layer-freezing under a given budget. We also propose better budget plans for the tradeoff.

We evaluate our approach in fine-tuning diverse datasets on a wide range of modern LMs. Based on the results, our semantic-based layer-freezing approach performs better than baselines. Combined with budget plans, our approach can further reduce the computation cost and improve LM performance. We discuss the insights of efficient finetuning from the perspective of semantics and conclude the findings in LM finetuning. The replication artifact is available online for open science [1].

Our contributions are as follows:

- We propose using semantic transition to describe the process of LM inference, and the derived formula of scaling law to estimate the capability of model layers, and further study the cost-benefit tradeoff in LM finetuning;
- We emphasize the importance of knowing where to finetune, through which we can improve the performance of LM finetuning and save the computation cost. We propose semantic-based layer-freezing as a solution;
- We conclude some findings on the behavior of LMs, which can contribute to future work in finetuning and analyzing LMs. Also, we propose planning the budget for a better cost-benefit tradeoff of LM finetuning.

2 Preliminaries

2.1 Semantic Field in LM Latent Space

Based on vocabulary-defined semantics, the semantics of hidden states can be regarded as the overlapping impact of "semantic fields" (Gu et al., 2024, 2025). The semantic field is similar to the field term in physics, such as electric field, where the electric strength relies on the distance to the center of the field (the electric pole). The corresponding probabilities on the vocabulary of a representation can be directly computed with its locations in the semantic fields in the latent space, as shown in Figure 1. In contrast, in common practice, the representations in last-layer latent space will undergo a dimensional change to be computed as logits, and then be normalized as the probabilities on the vocabulary. The dimensional change causes entanglement of semantics, and exacerbates the computation complexity.

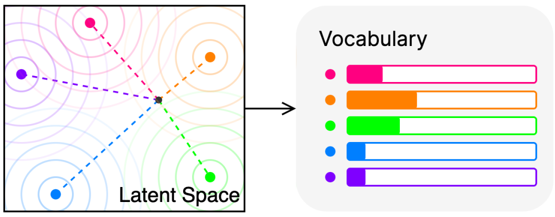

Figure 1: Vocabulary-defined semantics is demonstrated with a LM, whose vocabulary is a collection of colorful labels: (1) in the latent space (left), large color dots are the corresponding semantic bases of vocabulary labels. The small dark dot is the hidden state of a given data. The similarities of the data with semantic bases are regarded as logits; (2) on the vocabulary (right), the logits are normalized as probabilities, and the argmax label is *orange*. Consistently, the nearest semantic basis to the hidden state is the *orange* one.

The semantics of LM latent space is decided by semantic fields (Gu et al., 2024, 2025). For each label in the vocabulary, there is a corresponding semantic field in the latent space. The pole of each semantic field is called "semantic basis", representing an unmixed and purest meaning. If the nearest semantic basis to latent representations is the same one, they tend to share the meaning of that semantic basis. The semantic meaning of a representation in the latent space is decided by the overlapping impact of multiple semantic fields.

The computation of semantic bases is simple. At the LM input side, we multiply onehot embedding $\vec{e}$ by the embedding matrix $\mathbb{W}_i$ to obtain the semantic basis $\vec{r_i} = \vec{e} \cdot \mathbb{W}_i$. At the LM output side, due to the opposite operation direction between the

[1] https://github.com/jianguda/salf

embeddings and the representations, we turn to use the pseudoinverse of the LM-head matrix $\mathbb{W}_o^+$. We multiply onehot embedding $\vec{e}$ by the pseudoinverse matrix to obtain the semantic basis $\vec{r}_o = \vec{e} \cdot \mathbb{W}_o^+$. Since LM vocabulary is required in the computations, semantic bases only exist in the embedding latent space and the last-layer latent space.

2.2 Semantic-based Loss Computation

Based on the local isotropy of LM latent space (Cai et al., 2021), the logits in LM training and inference can be computed via similarity measurement (with semantic basis), instead of matrix multiplication (with LM-head matrix). The logits computed in this way is termed as "similarity-based logits" (Gu et al., 2024). It proves to have the same effects as the common practice of logits computation, and shows advantages in disentangling the semantics.

Algorithm 1 Semantic Cross-Entropy Loss

```
Require: $N$ semantic bases $\vec{b_i}$; ground truth label $l$; last-layer latent repr $\vec{r}$
Ensure: optimization target $loss$
  logits ← init_1d_tensor($N$)
  for $i \leftarrow 0$ to $N$ do
    logits[$i$] ← cosine_similarity($\vec{r}$, $\vec{b_i}$)
  end for
  loss ← cross_entropy_loss(logits, $l$)
```

In LM finetuning, the logits will be used in loss computation. Taking the cross-entropy loss as an example, we compute the similarities between a given hidden state with semantic bases as similarity-based logits, and then compute with the ground truth for the loss, as shown in Algorithm 1. In terms of numerical calculations, when computing in the last-layer latent space, it is equivalent to the logits computed via matrix multiplication.

Algorithm 2 Semantic Cosine-Distance Loss

```
Require: $l$-th semantic base $\vec{b_i}$ (for ground truth label $l$); last-layer latent repr $\vec{r}$
Ensure: optimization target $loss$
  loss ← 1 - cosine_similarity($\vec{r}$, $\vec{b_l}$)
```

Further, leveraging the disentanglement effects of similarity-based logits, we can compute the loss merely with the corresponding ground truth. In the loss computation, the hidden state is only computed with one semantic basis solely, instead of with all semantic bases, as shown in Algorithm 2. In terms of effect, it optimizes the hidden state to make it steer towards the corresponding semantic basis. The cosine-distance loss is better in computation cost, and its computation shows an intuitive geometric meaning in the latent space.

3 Computation-Efficient Fine-Tuning

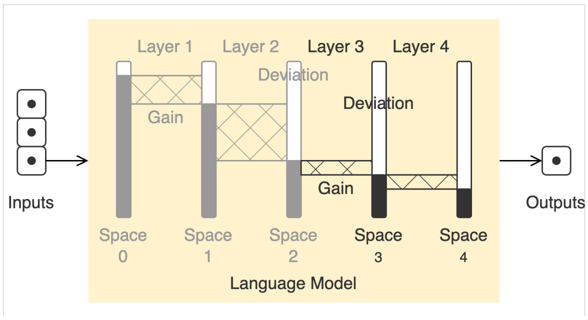

Figure 2: Our SALF approach is demonstrated with a LM with 4 layers (so there are 5 latent spaces). The rectangles with dark bars are the deviations, and the rectangles with cross-hatching are the gains in reducing deviations. In LM finetuning, SALF uses a semantic-based analysis to compute the deviations in each latent space, and then uses a derived formula of the scaling law to estimate the gains of each model layer. SALF will find the layer with the maximum gain and only finetune the deeper layers. In the illustration, layer 2 is chosen and the first two layers are frozen, so only layer 3 and layer 4 will be finetuned. The layers and spaces marked in gray color mean their gains and deviations will remain unchanged in LM finetuning.

Semantic-Aware Layer-Freezing, shortened as SALF, is a novel technique to reduce the computation cost in LM finetuning, by freezing certain layers. The core idea is dropping the unnecessary computation in LM backward-pass. Due to the chain rule in loss backpropagation, the computation on shallower layers requires the computation in deeper layers. That is, SALF realizes a computation-efficient LM finetuning by freezing the first few layers. To guarantee that layer-freezing will not damage the finetuning effects, and even improve the finetuning effects, we proposed a semantic-based analysis on LM inference and a derived formula of scaling law to estimate the convergence of layers. An illustration of our SALF approach is shown in Figure 2. We also introduce strategies of assigning data samples under a given budget, to obtain a good cost-benefit balance for layer-freezing.

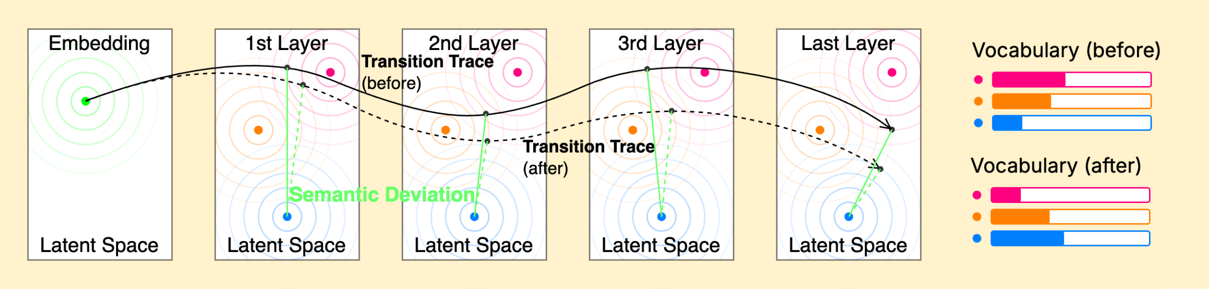

Figure 3: The transition of semantics is illustrated with a 4-layer LM, whose vocabulary is a collection of colorful labels. The green dot is the medium token (input-side semantic basis) and the blue dot is the ground truth (output-side semantic basis). The solid/dashed black curves (transition trace) represent the semantic transition of the medium token, defined by the dark dots (hidden states) in each latent space. The solid/dashed green lines (semantic deviation) indicate the differences between the hidden states and the ground truth, and they differ before and after LM finetuning: Comparing solid green lines (before finetuning) with dashed green lines (after finetuning), the semantic deviation in each layer is reduced. Through LM finetuning, the hidden state in the last-layer semantically approaches to the ground truth, and the argmax label becomes from *red* to *blue*.

3.1 Transitions on Semantics

In next-token prediction, the last token in the given input is used as the medium to compute the next token, denoted as *medium token*. Influenced by the embeddings of other tokens and the parameters in model layers, the medium token will undergo a layer-by-layer transition on its semantic meaning, denoted as *semantic transition*. LM finetuning has effects on semantic transition, and the differences before and after finetuning are illustrated in Figure 3. We define the involved concepts as below.

Transition Trace. For a given sequence of n tokens, $t_1, t_2, ..., t_n$, assume a m-layer LM will predict the next token t_{n+1}, the representation of t_n undergoes a semantic transition from semantic meaning i to j. The hidden state in each layer is denoted as $f_0, f_1, f_2, ..., f_m$ (f_0 is the onehot embedding, equals to i; while f_m is the last-layer representation, equals to j), so the semantic transition defined by these representations is a *transition trace*.

Transition Deviation. For a semantic transition of a m-layer LM, the deviation of the hidden state in the k-th layer to the semantic basis of the ground truth, called *semantic deviations*, is denoted as d_k. It is measured using cosine similarity, that is, $d_k = 1 - \texttt{cosine\_similarity}(f_k, \vec{v})$, where $\vec{v}$ is the semantic basis of ground truth label. Concerning computation, the semantic deviation is equivalent to the semantic cosine-distance loss, but can also be measured using other metrics.

The semantic deviations before and after finetuning differ. Theoretically and empirically, LM finetuning tends to reduce the deviations. For a given medium token, in LM finetuning, the transition trace will approach the semantic basis of the corresponding ground truth. The approach will be reflected in the deviation in each layer. By probing the situation of each layer, the semantic deviation will be reduced as well. That means, the hidden state will approach the semantic basis of the corresponding ground truth.

Further, semantic deviations can be regarded as the evaluation metrics of the capability of model layers. In the latent space of the LM last-layer, the representation of the medium token is intended to be close enough to the semantic basis (namely the ground truth). If the hidden states in the middle layers are close to the semantic basis of the corresponding ground truth, then the hidden state in the last layer is likely to be close to the semantic basis as well. Therefore, leveraging the semantic deviations, model layers can be finetuned selectively.

3.2 Layer-level Convergence Estimation

We propose an intuitive method to measure the performance of each model layer leveraging scaling laws. Scaling laws refer to empirical relationships that describe how the model performance improves with increasing resources, including data amount, model size, and convergence degree (which is often revealed as computational power).

According to the compute-optimal scaling law of LM pretraining (Hoffmann et al., 2022; Zhang et al., 2024), the training loss follows a parametric function of the information entropy of training data E, the number of model parameters N and the amount of data tokens D. The function is shown as Equation (1). In terms of definition, the second term $\frac{a}{N^\alpha}$ is the ideal capability of model, and the third term $\frac{b}{D^\beta}$ is the finite optimization of data.

$$\hat{L}_{pretrain} \triangleq E + \frac{a}{N^\alpha} + \frac{b}{D^\beta} \tag{1}$$

By performing a slight derivation on the definition of the scaling law, the relationships between the finetuning loss (of the data for finetuning) and resources (used in LM pretraining) can be described as shown in Equation (2). The first term $L_0(E')$ is the loss when interpreting the information entropy of the given data. It only depends on the embedding process and the LM-head, excluding all model layers, since the information entropy is converted to predictable tokens by LM vocabulary. The second term $C(N, D)$ is the capability of models in finetuning, which is the degree close to the convergence. It is a function of data amount D and model size N, corresponding to the latter two terms in Equation (1). In the derived formula, the first term remains stable, while the second term will be larger as the finetuning goes on. It indicates an improved convergence, leading to a smaller loss.

$$\hat{L}_{finetune} \triangleq L_0(E') - C(N, D) \tag{2}$$

By targeting a given model, the capability of different layers can be estimated and compared. For a m-layer LM, where layers are denoted as $l_1, l_2, ..., l_m$, we define "virtual submodel" as the truncated models starting from the shallowest layer. The k-th virtual submodel, denoted as v_k, is composed of $l_1, l_2, ..., l_k$ (as well as the embedding layer and the LM-head). Meanwhile, the loss of all m virtual submodels are computed in one-time of LM forward-pass, so the capability of v_k are computed as $C_{v_k} \triangleq L_0 - L_k$. Further, we can compare the capability of model layers. The loss gain of l_k, denoted as G_{l_k}, indicates the capability difference between v_k and v_{k-1}, so we have $G_{l_k} \propto C_{v_k} - C_{v_{k-1}}$. Since L_0 the remains same, the loss gain are reduced as $G_{l_k} \propto L_{k-1} - L_k$. When the gain of l_k is positive, the capability of v_k is better than v_{k-1}. A larger gain indicates stronger improvement between neighboring submodels.

3.3 Semantic-Aware Layer-Freezing

In next-token prediction, via LM finetuning, the last-layer hidden state of the medium token is close enough to the ground truth. The finetuning process can be explained as divide-and-conquer: If the hidden state is closer to the virtual one in the k-th layer, then they tend to be closer as well in the $(k+1)$-th layer. It is consistent with an existing empirical finding that, when viewed through the output lens, the hidden states across LM layers yield distributions that converge monotonically to the final prediction (Belrose et al., 2023). By making the hidden state close enough to the semantic basis of the ground truth in each layer, the representation in the last-layer tends to be close to the ground truth as well.

Based on the explanation, we propose a layer-freezing method to accelerate finetuning. The idea is simple: *instead of finetuning from the first-layer, we find the layer where the gain is the largest and then finetune from there to the last-layer*. We call the layer having the largest gain as the end-of-freezing layer, short as *eof-layer*. The shallow layers will be frozen so only the eof-layer and deeper layers are trainable, as shown in Algorithm 3.

Algorithm 3 Semantic-Aware Layer-Freezing

```
Require: model, datum
# (a) compute deviations of latent spaces
deviations ← empty list
latent_reprs ← model(datum)
semantic_bases ← VDS(model)
for id ← 0 to layer_num+1 do
    deviation ← compute_deviation(
        latent_reprs[id],semantic_bases)
    deviations.add(deviation)
end for
# (b) compute gains of model layers
layer_gains ← empty list
for id ← 0 to layer_num do
    gain ← deviations[id] -
        deviations[id+1]
    layer_gains.add(gain)
end for
# (c) freeze layers and backpropagate
eof_layer ← argmax(layer_gains)
freeze_layers(range(eof_layer))
backpropagate(model, datum)
```

For a given dataset, the computation cost of backpropagation is decided by the depth of eof-layers, we can count the depths to know the cost-saving of layer-freezing. On the opposite, we can have a budget plan and force the depths of eof-layers to fulfill the budget. In this way, we control the cost-saving by planning the depth of eof-layers.

3.4 Budget for Layer-Freezing

To balance the effectiveness and cost of model finetuning, we incorporate a budget to determine the extent of layer-freezing based on specific requirements (see Appendix A). This budget represents the number of model layers to fine-tune for a given dataset. It controls the efficiency of LM finetuning, for example, we tend to give a low budget for LM finetuning if we want a high efficiency.

Budget Plan. Similar to the common practice of finetuning half layers, we design budget plans to control the cost-benefit tradeoff. For a given model of m layers, we make the amount of data, that is assigned to finetuning layers between the `eof_layer` to the last layer, following the relative proportion of the growth sequence: (1) Following geometric growth, we take the growth ratio as 2. Then, the amount of data assigned for finetuning follows the relative proportion of $1, 2, 4, ..., 2^{m-1}$; (2) Following arithmetic growth, we make the initial term the common difference between terms. Then, the amount of data assigned for finetuning follows the relative proportion of $1, 2, 3, ..., m-1$.

Budget Infilling. For a given dataset, if the budget cannot be infilled completely with the data, the infilling order will affect the cost-benefit tradeoff. We introduce two practices for budget infilling: (1) Breadth-First (BF) fills eof-layers in the shallow layers, and then deeper layers; (2) Depth-First (DF) fills eof-layers in all layers evenly, until layers are infilled successively from shallow to deep. We illustrated with a model having four layers, following geometric growth, as shown in Figure 4.

4 Experiments and Results

4.1 Setup

Datasets. We use 5 established datasets, covering the common natural language tasks: emotion recognition: CARER (Saravia et al., 2018); similarity detection: MRPC (Dolan and Brockett, 2005); sentiment analysis: SST5 (Socher et al., 2013); and general text classification: TREC (Voorhees and Tice, 2000) and WebSS (Phan et al., 2008). The statistics of datasets are shown in Table 1.

Models. We use the recently released LLMs, including Qwen2 (0.5B-7B) (Yang et al., 2024), Gemma2 (2B-9B) (Riviere et al., 2024), and the state-of-the-art Llama-3 (8.0B) [2]. They are performant in massive comparisons with other competitors, and leading in the popularity statistics (especially, most downloads per month) in the hugging-face website [3]. The details are available in Table 2.

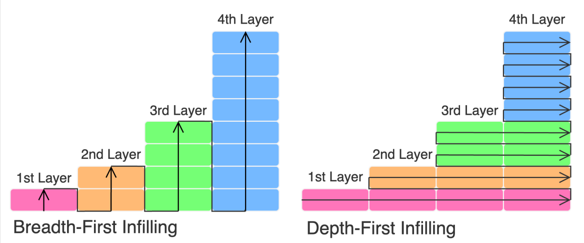

Figure 4: The order of budget infilling for a 4-layer model is: first red, then orange, then green, and finally blue shares. In breadth-first infilling, the color of shares is decided by layer. First let eof-layers be in first-layer until the layer is full (red); then let them be in second-layer until full (orange); then be in third-layer (green); and finally be in last-layer (blue). In depth-first infilling, the color of shares is decided by the position in layers. First let eof-layers be in first-share of all layers, from shallow to deep layers; then let them be in second-share of all layers; and then repeat the practice in the third share, forth share, until the budget of each layer is satisfied in the proper order (red, orange, green, and blue).

		CARER	MRPC	SST5	TREC	WebSS
Class Num.		6	2	5	6	8
Data Num.	Train	16,000	4,076	8,544	5,452	10,060
	Test	2,000	1,725	2,210	500	2,280
Avg. Prompt Length		25.6	61.0	28.0	17.1	27.8

Table 1: Stats of natural language datasets.

	Qwen2			Gemma2		Llama3
	0.5B	**1.5B**	**7B**	**2B**	**9B**	**8B**
Model Size	0.49B	1.54B	7.62B	2.61B	9.24B	8.03B
Head Num.	14	12	28	8	16	32
Layer Num.	24	28	28	26	42	32
Dimension	896	1,536	3,584	2,304	3,584	4,096
Vocabulary	151,936		152,064	256,000		128,256

Table 2: Stats of Qwen2, Gemma2, and Llama3 models.

Baselines. LIFT is the state-of-the-art in layer-wise LM finetuning on saving the computation cost. It takes a front-to-end selection policy to prioritize the layer to finetune (Zhu et al., 2024). However, it only finetunes one layer each time, which may damage its performance. We relax its restrictions for a stronger baseline by letting more layers be trainable while the computation cost is the same. We mark the vanilla one as `LIFT[half]`, and the enhanced one as `LIFT⋆[half]`. In addition, we also compare our approach SALF with two common finetuning practices with LoRA: full-layer finetun-

[2] https://github.com/meta-llama/llama3

[3] https://huggingface.co/

LLM	Method	Dataset					Avg.
		CARER	MRPC	SST5	TREC	WebSS	
Qwen2-0.5B	`LoRA[full]`	0.765	0.454	0.302	0.779	0.837	0.627
	`LoRA[half]`	0.746	0.663	0.335	0.795	0.847	0.677
	`LIFT[half]`	0.241	0.399	0.239	0.701	0.583	0.433
	`LIFT⋆[half]`	0.806	0.755	0.456	0.805	0.891	0.743
	`SALF[half]`	0.807	0.785	0.444	0.941	0.847	0.765
Qwen2-1.5B	`LoRA[full]`	0.835	0.750	0.293	0.787	0.749	0.683
	`LoRA[half]`	0.687	0.769	0.373	0.793	0.856	0.696
	`LIFT[half]`	0.469	0.566	0.318	0.727	0.705	0.557
	`LIFT⋆[half]`	0.823	0.779	0.503	0.808	0.917	0.766
	`SALF[half]`	0.815	0.735	0.520	0.940	0.876	0.777
Qwen2-7B	`LoRA[full]`	0.820	0.399	0.075	0.252	0.029	0.315
	`LoRA[half]`	0.794	0.690	0.388	0.796	0.866	0.707
	`LIFT[half]`	0.532	0.739	0.320	0.747	0.699	0.607
	`LIFT⋆[half]`	0.797	0.781	0.534	0.802	0.915	0.766
	`SALF[half]`	0.823	0.831	0.542	0.951	0.852	0.800
Gemma2-2B	`LoRA[full]`	0.867	0.494	0.281	0.711	0.771	0.625
	`LoRA[half]`	0.865	0.498	0.243	0.711	0.694	0.602
	`LIFT[half]`	0.328	0.399	0.082	0.505	0.453	0.353
	`LIFT⋆[half]`	0.872	0.518	0.280	0.737	0.797	0.641
	`SALF[half]`	0.877	0.399	0.199	0.734	0.778	0.597
Gemma2-9B	`LoRA[full]`	0.801	0.399	0.193	0.706	0.765	0.573
	`LoRA[half]`	0.865	0.399	0.201	0.725	0.741	0.586
	`LIFT[half]`	0.382	0.399	0.187	0.577	0.484	0.406
	`LIFT⋆[half]`	0.862	0.399	0.281	0.729	0.791	0.612
	`SALF[half]`	0.860	0.399	0.190	0.658	0.797	0.581
Llama3-8B	`LoRA[full]`	0.394	0.399	0.402	0.256	0.029	0.296
	`LoRA[half]`	0.818	0.664	0.338	0.784	0.861	0.693
	`LIFT[half]`	0.590	0.466	0.476	0.779	0.837	0.630
	`LIFT⋆[half]`	0.843	0.468	0.552	0.799	0.900	0.712
	`SALF[half]`	0.872	0.399	0.571	0.945	0.885	0.734

Table 3: F1 Scores of layer-freezing methods (on the diverse datasets and models).

ing and half-layer finetuning. The former is to finetune all model layers, while the latter is to finetune only the last half model layers. We marked them as `LoRA[full]` and `LoRA[half]`.

Metrics. For effectiveness, we use *F1 score* to measure whether the predicted next-token is the ground truth because of the class imbalance in the datasets. F1 score is the harmonic mean of precision and recall, and considers the effects of both false positives and false negatives. For efficiency, we use the "cost-saving" ratio as a new metric, representing the saved computation cost in backpropagation. Large ratios mean better effects.

Pipeline. We conduct LM finetuning experiments to compare our approach with other layer-freezing practices. Since LIFT is designed to save around 50% computation cost in backpropagation, we restrict our approach to the same computation cost for a fair comparison (on effectiveness). Our implementation is as we described, and does not use other techniques, such as instruction tuning. The details on the implementation are in Appendix A.

4.2 Performance Evaluation

We evaluate the performance of our approach and the baselines in LM finetuning: finetune the LMs on the training set, and do inference on the test set.

As shown in Table 3, based on the average F1 score, on 4 out of 6 models, SALF performs better than others, while on the other model, its performance is very close to the best. Compared with the common practices `LoRA[full]` and `LoRA[half]`, LIFT shows superiority in the performance while our approach SALF shows stable and obvious improvements. Besides, the advantages of SALF vary on the datasets. On WebSS, SALF performs the best only in the case where the model is Llama3, but the performance gap to the best is not obvious. However, SALF cannot show stable improvements on MRPC, especially when with Gemma2 and Llama3. The reason is that, the class number of the MRPC dataset is only 2, meaning the semantic transition is very simple, thereby the deviations in the process may not be very helpful. All methods cannot perform well in SST5 with Gemma2, which may caused by the bad semantic property of Gemma2 models due to its risk of semantic degradation in Gemma2 design. Gemma2 models use a non-standard FFN, which has an extra gate on top of GEGLU. In contrast, Qwen2 and Llama3 models use SwiGLU, which is a standard FFN design. Another possible reason is that, Gemma2 uses SentencePiece as the tokenizer, which is not as good as the BPE tokenizer used in Qwen2 and Llama3, especially in token granularity and consistency.

It is noteworthy that `LoRA[full]` performs worse than others, and even worse than `LoRA[half]`. It is counter-intuitive since full-layer finetuning is updating all layers and requires a larger computation cost than `LoRA[half]`. However, in our understanding, it may caused by the difference in the effects of shallow and deep model layers. Usually, shallow layers learn the macro features while deep layers learn the micro features (So et al., 2019; Brown et al., 2020). It means, *when the learning rate is fixed in LM finetuning, the update in shallow layers shall be less frequent than that in deep layers*. It also explains the reason why both `LoRA[full]` and `LoRA[half]` perform not as well as LIFT⋆ or SALF: `LoRA[full]` updates shallow layers too often while `LoRA[half]` updates them too seldom.

Meanwhile, the results of the baseline LIFT is not as good as the enhanced implementation LIFT⋆. Their difference is that, the former only makes the eof-layer trainable, while the latter finetunes all layers between eof-layer to last-layer. It indicates that, merely finetuning shallow layers cannot guarantee smaller deviations in the deep layers, or the deviations require further processing.

In Appendix B, we compared the effects of taking different metrics of deviations. Further, we analyzed the advantages of SALF in LM finetuning with the illustrations on semantic deviations.

5 Analysis on Cost-Benefit Tradeoff

We study the performance and cost-benefit tradeoff of budget plans and infilling practices to layer-freezing. For example, a geometric-growth budget with breadth-first infilling is denoted as `geom[bf]`.

As shown in Table 4, the budget for cost-benefit tradeoff is useful to both LIFT⋆ and our approach SALF, while our approach still shows better performance. In comparison, the arithmetic-growth budget shows similar performance to the geometric-growth budget. Meanwhile, the practice of depth-first infilling tends to perform better and more stably than breadth-first infilling.

Budget	Dataset					Avg.
	CARER	MRPC	SST5	TREC	WebSS	
`LIFT⋆[half]`	0.843	0.468	0.552	0.799	0.900	0.712
`LIFT⋆[arith][bf]`	0.817	0.516	0.548	0.798	0.893	0.714
`LIFT⋆[arith][df]`	0.835	0.581	0.543	0.789	0.906	0.731
`LIFT⋆[geom][bf]`	0.763	0.729	0.404	0.791	0.876	0.713
`LIFT⋆[geom][df]`	0.845	0.625	0.559	0.795	0.899	0.745
`SALF[half]`	0.872	0.399	0.571	0.945	0.885	0.734
`SALF[geom][bf]`	0.906	0.665	0.586	0.962	0.855	0.795
`SALF[geom][df]`	0.920	0.711	0.607	0.970	0.921	0.826
`SALF[arith][bf]`	0.921	0.752	0.391	0.964	0.911	0.788
`SALF[arith][df]`	0.914	0.751	0.588	0.964	0.914	0.826

Table 4: Accuracy of layer-freezing methods with different budget plans and infilling practices (on the diverse datasets, using Llama3-8B).

As shown in Table 5, compared with geometric-growth, the budget of arithmetic-growth saves more computation costs. The reason is that, for a model of the same number of layers, the arithmetic-growth increases slower than the geometric-growth, so the budget of the latter is not likely to be fulfilled. For geometric-growth, eof-layers can fill in shallow layers but cannot fill in deep layers. Also, depth-first infilling saves more than the breadth-first infilling. The reason is similar, more eof-layers tend to be in deep layers than in shallow layers.

Considering the efficiency and cost-benefit tradeoff, the budget of arithmetic-growth shows equivalent performance but saves more computation costs. Also, the practice of depth-first infilling is better than breadth-first infilling. Based on the results, an arithmetic-growth with depth-first infilling saves around $1/3$ more computation cost and has a slightly better performance. The reason explaining why the combination is performant is the same as discussed, *when the learning rate is fixed in LM finetuning, the update in shallow layers shall be less frequent than that in deep layers.*

Budget	Dataset					Avg.
	CARER	MRPC	SST5	TREC	WebSS	
`LoRA[full]`	0.000	0.000	0.000	0.000	0.000	0.000
`LoRA[half]`	0.500	0.500	0.500	0.500	0.500	0.500
`LIFT[half]`	0.484	0.483	0.484	0.484	0.484	0.484
`LIFT⋆[half]`	0.484	0.483	0.484	0.484	0.484	0.484
`SALF[half]`	0.484	0.483	0.484	0.484	0.484	0.484
`[geom][bf]`	0.374	0.312	0.346	0.328	0.355	0.343
`[geom][df]`	0.616	0.583	0.601	0.589	0.604	0.598
`[arith][bf]`	0.614	0.613	0.613	0.609	0.615	0.613
`[arith][df]`	0.644	0.640	0.644	0.642	0.645	0.643

Table 5: Backpropagation cost-saving of layer-freezing methods with different budget plans (on the diverse datasets, using Llama3-8B).

6 Related Work

Leveraging the layered structure of neural models, the concept of layer-freezing was proposed decades ago, but mainly for deep belief networks (DBN) (Hinton, 2009). DBN is a stack of directed sigmoid belief network (SBN) (Neal, 1992) and an undirected restricted boltzmann machine (Hinton, 2017). The backpropagation is only applied to finetune the restricted boltzmann machine, while the dependencies between other layers are not bidirectional. Therefore, progressively training each layer is proposed as a greedy strategy for training DBN (Hinton et al., 2006; Bengio et al., 2006).

In the era of language models, there has been little significant work studying layer-freezing for efficient finetuning, while the focus often lies on parameter-efficient, namely reducing the amount of trainable parameters, instead of computation-efficicent (Pan et al., 2024; Zhu et al., 2024). One reason is the complexity and interpretability of language models. Besides, the correlation between model layers is not intuitive, and the effects of bidirectional dependencies on layer-wise finetuning have not been studied. Another reason is that, the prior work on PEFT shows similar effects on reducing the number of trainable parameters, or even making the trainable parameters detachable.

7 Conclusion

In this paper, we have proposed the novel concept of semantic transition. By defining transition trace to describe the change of semantic meaning of the next token, we explain LM finetuning as the process of letting the representation gradually steer to the corresponding ground truth in latent space.

Meanwhile, based on a derived law of scaling law, we can reasonably estimate and compare the capability of model layers, so to better allocate the computation resources in LM finetuning. Further, we propose layer-freezing to accelerate LM finetuning, by finding the layer with the maximum gains of reducing deviation and finetuning deeper layers.

Based on our results on diverse datasets and multiple models, semantic-aware layer-freezing provides better performance than the state-of-the-art as well as common practices. Moreover, our work explores the effects of budget plans on the cost-benefit tradeoff for layer-freezing. In return, the effectiveness of our lay-finetuning approach validates the usefulness of semantic transition.

Limitations

In this paper, we proposed semantic transition as a new perspective on the LMs' functionality. Besides, estimated and compared the capability of model layers. We suggest using the gains of reducing deviations in semantic transition to reduce the computation cost of LM finetuning. Our approach maintains and even improves the performance of LM finetuning.

In our understanding, our approach is leveraging the derived formula of scaling law to estimate and compare the capability of model layers. However, the capability cannot be strictly seen as the convergence degree, namely the expected benefits of finetuning a certain model layer. Besides, freezing the layer with the maximum gains of reducing deviation and finetuning shallower layers is an empirical wise practice, but there is no proof saying this is optimal. Meanwhile, in a high-dimensional latent space, the representations tend to be orthogonal to each other (Vershynin, 2018). Therefore, using the cosine distance between latent representation and the semantic basis as the deviation may not be optimal practice. There possibly exists potential evidence to support other better choices.

The semantic transition is based on the similarity measurement between latent representations and semantic bases. The theoretical support is the local isotropy of LM latent space (Cai et al., 2021), therefore for the language models whose latent space cannot fulfill local isotropy in terms of semantics (even though they seem not to exist, to the best of our knowledge), our approach may not stand.

References

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor V. Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. 2023. Eliciting latent predictions from transformers with the tuned lens. *ArXiv*, abs/2303.08112.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and H. Larochelle. 2006. Greedy layer-wise training of deep networks. In *Neural Information Processing Systems*.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel J. Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the opportunities and risks of foundation models. *ArXiv*, abs/2108.07258.

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, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff 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. *ArXiv*, abs/2005.14165.

Xingyu Cai, Jiaji Huang, Yu-Lan Bian, and Kenneth Ward Church. 2021. Isotropy in the contextual embedding space: Clusters and manifolds. In *International Conference on Learning Representations*.

Pierre Carbonnelle. 2024. Pypl popularity of programming language index.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *ArXiv*, abs/2305.14314.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *International Joint Conference on Natural Language Processing*.

Jian Gu, Aldeida Aleti, Chunyang Chen, and Hongyu Zhang. 2023. Neuron patching: Semantic-based neuron-level language model repair for code generation.

Jian Gu, Aldeida Aleti, Chunyang Chen, and Hongyu Zhang. 2024. Vocabulary-defined semantics: Latent space clustering for improving in-context learning. *Preprint*, arXiv:2401.16184.

Jian Gu, Aldeida Aleti, Chunyang Chen, and Hongyu Zhang. 2025. Seme: Training-free language model merging via semantic alignment. *Preprint*, arXiv:2505.20144.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *ArXiv*, abs/2403.14608.

Geoffrey E. Hinton. 2009. Deep belief networks. *Scholarpedia*, 4:5947.

Geoffrey E. Hinton. 2017. Boltzmann machines. In *Encyclopedia of Machine Learning and Data Mining*.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and L. Sifre. 2022. Training compute-optimal large language models. *ArXiv*, abs/2203.15556.

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

Radford M. Neal. 1992. Connectionist learning of belief networks. *Artif. Intell.*, 56:71–113.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. 2024. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. *ArXiv*, abs/2403.17919.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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 *Neural Information Processing Systems*.

Xuan Hieu Phan, Minh Le Nguyen, and Susumu Horiguchi. 2008. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *The Web Conference*.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Gemma Team Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, L'eonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ram'e, Johan Ferret, Peter Liu, Pouya Dehghani Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stańczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Boxi Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Christoper A. Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozi'nska, D. Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Pluci'nska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost R. van Amersfoort, Josh Gordon, Josh Lipschultz, Joshua Newlan, Junsong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, L. Sifre, L. Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Gorner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, S. Mc Carthy, Sarah Perrin, S'ebastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomás Kociský, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Brian Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeffrey Dean, Demis Hassabis, Koray Kavukcuoglu, Cl'ement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. 2024. Gemma 2: Improving open language models at a practical size. *ArXiv*, abs/2408.00118.

Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. 2022. A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Transactions on Intelligent Systems and Technology*, 14:1 – 50.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, I. Evtimov, Joanna Bitton, Manish P Bhatt, Cristian Cantón Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre D'efossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *ArXiv*, abs/2308.12950.

Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. 2018. Carer: Contextualized affect representations for emotion recognition. In *Conference on Empirical Methods in Natural Language Processing*.

David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. In *International Conference on Machine Learning*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, A. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing*.

TIOBE. 2024. Tiobe index | tiobe - the software quality company.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971.

Roman Vershynin. 2018. *Random Vectors in High Dimensions*, page 38–69. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

Ellen M. Voorhees and Dawn M. Tice. 2000. Building a question answering test collection. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models. *ArXiv*, abs/2206.07682.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Transformers: State-of-the-art natural language processing. In *Conference on Empirical Methods in Natural Language Processing*.

Xiaoxia Wu, Cheng Li, Reza Yazdani Aminabadi, Zhewei Yao, and Yuxiong He. 2023. Understanding int4 quantization for language models: Latency speedup, composability, and failure cases. In *International Conference on Machine Learning*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. 2024. Qwen2 technical report. *ArXiv*, abs/2407.10671.

Yunzhi Yao, Peng Wang, Bo Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu

Zhang. 2023. Editing large language models: Problems, methods, and opportunities. In *Conference on Empirical Methods in Natural Language Processing*.

Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. 2024. When scaling meets llm finetuning: The effect of data, model and finetuning method. *ArXiv*, abs/2402.17193.

Ligeng Zhu, Lanxiang Hu, Ji Lin, and Song Han. 2024. LIFT: Efficient layer-wise fine-tuning for large model models.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Troy Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, Zico Kolter, and Dan Hendrycks. 2023. Representation engineering: A top-down approach to ai transparency. *ArXiv*, abs/2310.01405.

A Implementation Details

A.1 Environments

Our implementation uses deep learning framework PyTorch (Paszke et al., 2019), Transformers (Wolf et al., 2019), and use PEFT [4] to conduct the LoRA experiments. The LM finetuning experiments are based on existing PEFT methods, specifically LoRA (Hu et al., 2021). We use quantization techniques (INT4) to load Qwen2-7B, Gemma2-9B, and Llama3-8B, with the default settings (Dettmers et al., 2023; Wu et al., 2023), which reduces the memory requirements in LM finetuning with slight performance loss.

The experiments are conducted via a single run, with the global random-seed 42. The computation is based on a single Nvidia V100 (32 GB), and the computation budget is around 2200 GPU hours.

A.2 License and Terms

We understand and respect the licenses used in our experiments, including the Apache-2.0 license for Qwen2 models and Gemma2 models, as well as the Llama3 community license for Llama3 models [5]. We confirm that our use of existing artifacts was consistent with their intended use.

A.3 SALF Algorithm with Budget

By introducing the budget for LM finetuning, our semantic-based layer-freezing approach can fulfill the intended computation cost. Then, to guarantee improved performance, we propose the SALF algorithm with the budget consideration, as shown in Algorithm 4.

Algorithm 4 SALF w/ Budgets

```
Require: model, data, budgets
tabu_data ← empty list
for layer ← 0 to layer_num do
    # (a) freeze layers from deep to shallow
    freeze_layers(range(layer))
    # Backpropagation of Matching Data
    for datum in data do
        # (b) check whether to jump the loop
        if budgets[layer] == 0 then
            break
        end if
        if datum in tabu_data then
            continue
        end if
        # (c) execute line 1-16 in Algorithm 3
        eof_layer ← SALF(model, datum)
        if eof_layer > layer then
            continue
        end if
        # (d) backpropagate
        backpropagate(model, datum)
        budgets[layer] -= 1
        tabu_data.append(datum)
    end for
    # Backpropagation of Remaining Data
    sampled_data ← random_sample(
        data, filter=tabu_data,
        amount=budgets[layer])
    finetune(model, sampled_data)
    budgets[layer] ← 0
    tabu_data.extend(sampled_data)
end for
```

The intent of the code is intuitive: first, compute the deviations to find the eof-layer for each data; then, arrange the data with the similar eof-layers into the budget; last, gradually narrow down the scope of finetuning (freezing more model layers), and use the arranged data to backpropagate the loss. For the sake of the sequential access restriction of `data-loader`, the algorithm is described with the for-loops and the repeated iterations. In the implementation, we can use random access and caching techniques to remove the for-loops and reduce the number of iterations.

[4] https://github.com/huggingface/peft

[5] https://llama.meta.com/llama3/license/

B More Analysis

B.1 Differences between Parameter-Efficiency and Computation-Efficiency

Different from PEFT methods proposed for better parameter-efficiency, our approach SALF (as well as the baseline LIFT) is a layer-freezing method proposed for better computation-efficiency. The focus of parameter-efficiency is reducing the memory cost of finetuning, while in contrast, the focus of computation-efficiency is reducing the computation cost of backpropagation. For newly-emerging topics, including knowledge editing (Yao et al., 2023), representation engineering (Zou et al., 2023), and language model repair (Gu et al., 2023), computation-efficiency is critical in realizing the flexibility and adaptability.

Compared with full-parameter finetuning, PEFT methods cannot guarantee computation-efficiency. The computation cost of finetuning covers the cost of forward-inference and back-propagation. The forward-inference cost cannot be reduced, so any method for better computation-efficiency must deal with the back-propagation cost. Then, based on the *chain rule* of calculus to compute gradients, which is the mathematical foundation of back-propagation, if the gradients of the k-th layer are needed, the gradient computation of any deeper layers (whose layer index is larger than k) cannot be skipped. Therefore, PEFT methods like LoRA are not computation-efficient since they cannot reduce the cost of back-propagation. For the same reason, layer-freezing is intuitive and reliable in guaranteeing the computation-efficiency.

B.2 Metrics for Computing Deviations

SALF represents a common practice to detect how the model capability improves across different layers. That is, probing the hidden states in middle layers, and using them for logits computation as an estimation for LM interpretability.

When computing the deviations in LM inference, there are alternatives to the used semantic cosine-distance loss. We check the case where letting the cross-entropy loss be the deviation measurement, denoted as SALF[half][ce]. Since cross-entropy loss is do computation with all ground truths, not merely with the corresponding one, as did by cosine-distance loss, the former one involves more constraints than the latter one. It indicates that SALF[half][ce] will be slower in convergence, and further explains why this variant cannot perform as well as SALF[half] when training for the same epoch. Based on our analysis, they tend to have similar performance when doing model finetuning for an unlimited number of epochs until convergence. In our understanding, *a less constrained loss function indicates a more straightforward convergence process, and therefore tends to perform better in LM finetuning*. Since the cross-entropy loss is commonly used in logits computation, the advantages of SALF indicate that, cosine-distance loss is a notable alternative for its better efficiency.

Variant	Dataset					Avg.
	CARER	MRPC	SST5	TREC	WebSS	
SALF[half]	0.872	0.399	0.571	0.945	0.885	0.734
SALF[half][ce]	0.213	0.595	0.419	0.952	0.835	0.603
SALF[half][rank]	0.086	0.753	0.455	0.946	0.876	0.623

Table 6: F1 scores of layer-freezing variants (on the diverse datasets, using Llama3-8B).

Meanwhile, we experimented with a variant using the customized metric: SALF[half][rank] measures the ranking of the ground truth in the output probabilities. Theoretically, in LM finetuning, the ranking of the ground truth shall keep increasing until becoming the first. As shown in Table 6, it fails to realize the equivalent performances to SALF. Based on our analaysis, it is caused by the small output space and the large model size. For example, Llama3-8B has 32 model layers while the class number of datasets is smaller than 10, so the deviations tend to be very small, and so do the gains in reducing the deviations. The variant SALF[half][rank] cannot be numerically sensitive, since its deviations tend to remain unchanged in neighboring layers and the gains cannot express useful information. In contrast, the cosine-distance loss is numerically sensitive, and focuses on cosine similarity with the corresponding ground truth.

B.3 Heuristics in Layer Freezing

Based on the analytical results on semantic transition in LMs, namely deviations and gains, there are heuristic practices to decide which layers to freeze.

In our approach, we find the model layer where the gain is the largest, and freeze it and other deeper layers. Concerning the effects, after multiple turns of layer-freezing, deeper layers are more likely to become the largest-gain layer, especially the deepest layer. Considering the functionality of each LM layer contributes to the convergence monotonically (Belrose et al., 2023), our approach tends to force each layer contribute to the converge. Correspondingly, the gain of a deep layer will be larger than that of a shallow layer.

We conducted a contrastive study on other similar decisions, such as finding the least-deviation layer, and then freeze it and other deeper layers. We observed that the located layer is usually the first model layer, because the first model layer usually improves the largest model capability, which means, the deviation gain is the least. Therefore, if the strategy of layer-freezing is to find the layer of the least gain, the effect is like freezing no layers.

As for the heuristic options which decide layer-freezing directly using deviations, we find their benefits to LM finetuning are not stable. It indicates that, the comparisons in adjacent layers provide more useful information than directly using the deviations. Based on our understanding, layer-by-layer comparison serves the function of regularizing deviations. The necessity of regularization is because the layer-wise deviations correspond to different latent spaces, so the deviations cannot be compared across model layers.

B.4 Semantic Effects of LM Finetuning

To study the effects of our SALF approach on LM finetuning, we illustrate the deviation changes in LM finetuning of two settings: one is making all layers trainable, corresponding to `LoRA[full]`, as shown in Figure 5; while the other one is taking our approach for layer-freezing, corresponding to `SALF[half]`, as shown in Figure 6.

In the illustrations, the deviations are in the range of $[0, 2]$, since it is derived from the cosine similarity. Besides, in a high-dimensional latent space, the representations tend to be orthogonal to others (including the semantic bases) (Vershynin, 2018), so when the deviations are smaller than 1, it means the corresponding data representations are steering towards the ground truth, then the corresponding LM predictions may be correct. Otherwise, if the deviations are larger than 1, then the corresponding LM predictions are not likely to be correct.

By comparing the illustrated two situations of the blue shapes, we conclude the advantages of our semantic-based layer-freezing approach to LM finetuning as: our approach avoids the side effects of LM finetuning to shallow layers, and tends to make the semantic deviations in deep layers small. Taking the illustrated situation of the red shapes as a reference, we believe that the first advantage (on the side effects to the shallow layers) may be the cause of the second advantage (on the small deviations in deep layers). It explains why our approach lead to small deviations in deep layers, and also, it also emphasizes the importance of reducing the deviations in shallow layers. Further, the causation explains how to achieve better performance while reducing the computation cost in LM finetuning.

In addition, based on the illustrations, we see the accumulated effects of our approach in reducing the deviations in the last few model layers, where the blue shapes gradually move to lower positions, which indicates lower deviations of the data and a higher likelihood of correct LM predictions.

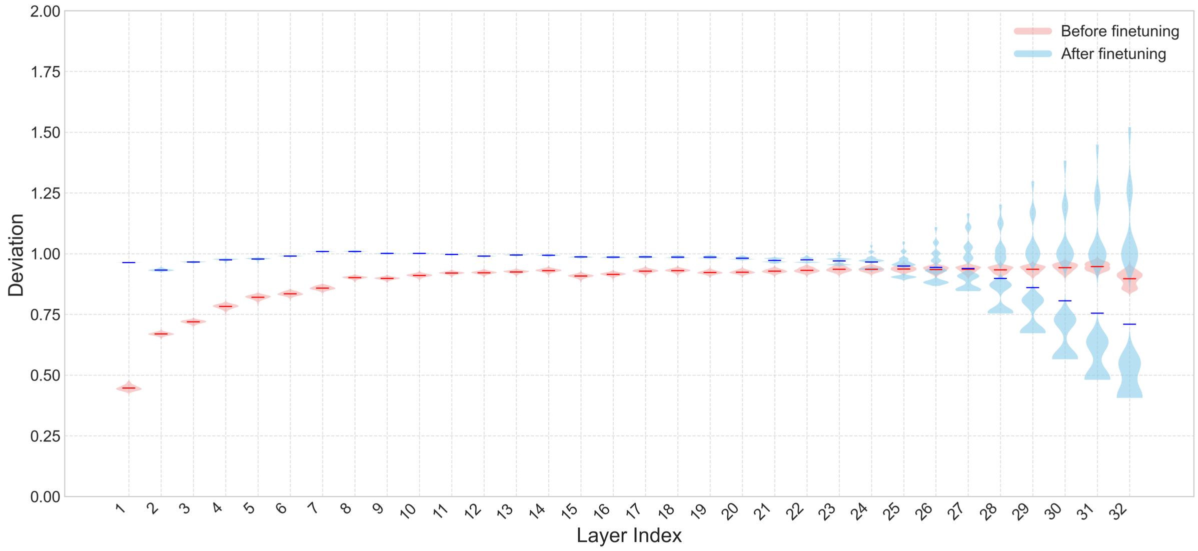

Figure 5: Violin plot of the deviations of each layer in LM finetuning, where the crossbars represent the mean of deviations, when making all model layers trainable (on the CARER dataset, using Llama3-8B). The phenomena include: (1) The red crossbars usually lie at lower positions than the blue crossbars (in the first 27 layers). It means, the deviation changes by LM finetuning are negative in most layers. (2) The blue shapes are flattened in the last few layers (from the 25-th layer to the last-layer) but some areas in the shapes lie at higher positions. It means, the distribution of the deviations in the last layers is forming multiple peaks, no longer centered in only one peak, and lots of data show higher deviations; (3) The differences between red and blue are large and show a reversal (first red is better, then blue is better) in the first and last few layers. It means, the deviation changes by LM finetuning are significant, which are worse in the shallower layers but better in the deeper layers.

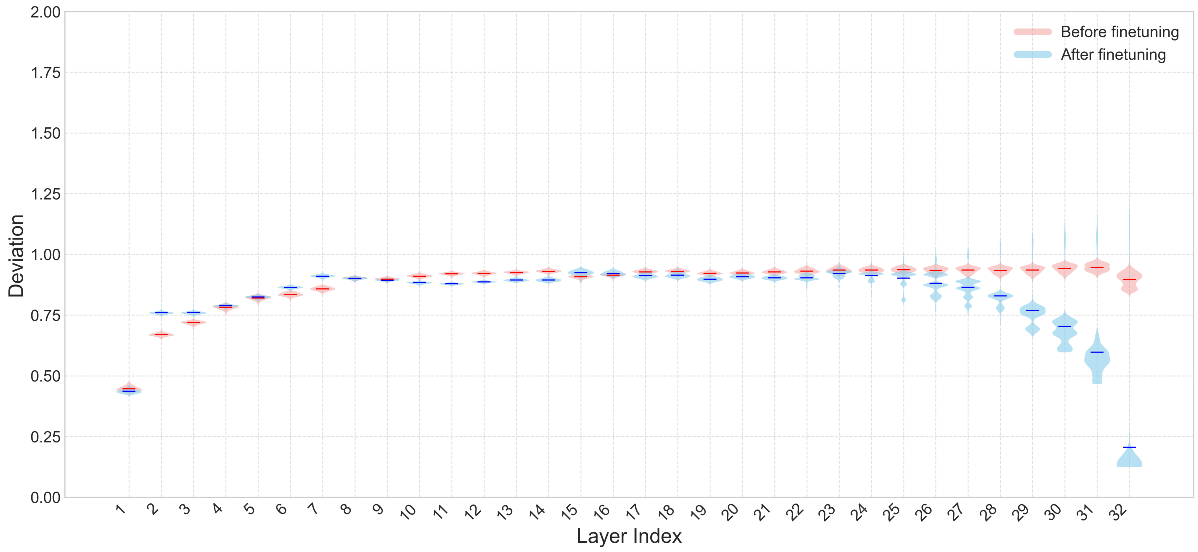

Figure 6: Violin plot of the deviations of each layer in LM finetuning, where the crossbars represent the mean of deviations, when taking semantic-based layer-freezing (on the CARER dataset, using Llama3-8B). The phenomena include: (1) The red crossbars usually lie at the same positions as the blue crossbars (in the first 27 layers). It means, the deviation changes by LM finetuning are very small in most layers. (2) The blue shapes are flattened in the last few layers (from the 25-th layer to the last-layer) but almost all areas in the shapes lie at lower positions. It means, the distribution of the deviations in the last layers is forming multiple peaks, no longer centered in only one peak, and almost all data show lower deviations; (3) The differences between red and blue are only getting large (blue is better) in the last few layers. It means, the deviation changes by LM finetuning are positive and highly targeted, which are mainly in the deeper layers.