Suppressing Final Layer Hidden State Jumps in Transformer Pretraining

Keigo Shibata[1], Kazuki Yano[1], Ryosuke Takahashi[1,2], Jaesung Lee[1],
Wataru Ikeda[1], **Jun Suzuki[1,2,3]**.
[1]Tohoku University, [2]RIKEN, [3]NII LLMC,
`is-failab-research@grp.tohoku.ac.jp`

Abstract

This paper discusses the internal behavior of Transformer language models. Many recent pre-trained models have been reported to exhibit only slight changes in the angular distance between the input and output hidden state vectors in the middle Transformer layers, despite a disproportionately large "jump" in the angular distance occurring in or around the final Transformer layer. To characterize this, we first introduce a quantitative metric for the jump strength around the final layer, and then demonstrate its prevalence across many open-weight models, as well as its amplification throughout pre-training. Assuming such jumps indicate an undesirable property, we propose the jump-suppressing regularizer (JREG) which penalizes this jump during pre-training, thereby encouraging more balanced capability usage across the middle layers. Empirical evaluations of three model sizes of Llama-based models, trained with the proposed JREG method, reveal improved task performance compared to the baseline without altering the model architecture.

1 Introduction

Transformer-based language models (Transformer-LMs) have demonstrated outstanding performance across a broad spectrum of artificial intelligence (AI) tasks (Achiam et al., 2023; Grattafiori et al., 2024a). This success has motivated research to elucidate the internal mechanisms that enable these models to generate appropriate and fluent responses to a wide range of instructions (Tigges et al., 2024; Stolfo et al., 2024). As a result, studies interpreting various aspects of Transformer-LMs have attracted considerable attention and emerged as one of the most vibrant areas in recent AI research (Belrose et al., 2023; Kobayashi et al., 2024).

Regarding the model structure of Transformer-LMs, many studies have reported the existence of redundant and ineffective parameters in pre-trained
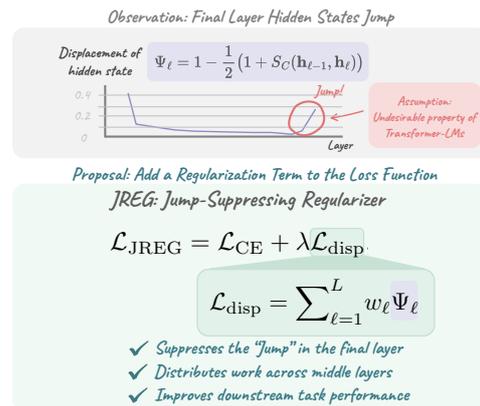

Figure 1: Many recent open-weight language models exhibit minimal hidden state displacement in their middle layers, but exhibit a pronounced "jump" at their final layer. Suppressing this jump during pre-training could improve overall performance by fostering a more balanced use of capabilities across the middle layers.

LMs (Tyukin et al., 2024; Men et al., 2025; Sun et al., 2025a), while other studies have suggested that such redundancy plays a critical role in enabling effective and efficient pre-training (Aghajanyan et al., 2021; Song et al., 2024; Lad et al., 2025). In particular, pre-trained Llama models (Grattafiori et al., 2024a) have been observed to contain many redundant and ineffective Transformer layers in the middle of the models (Sun et al., 2025a). Specifically, this observation can be rewritten to state that these middle layers produce output hidden state vectors that are highly similar to their input vectors in terms of angular distance. In contrast to such behavior in the middle layers, several studies (Sun et al., 2025a; Tyukin et al., 2024) have also recently observed and reported the "jump" behavior, which we define as a pronounced large change in angular distance, occurring in (and, in some cases, around) the final layer(s). We hypothesize that a small angular distance in the middle layers leads the final layer to exhibit a dispropor-

tionately large angular distance change because the relative representational load on those middle layers is reduced, causing the final layer to bear more of the workload to compensate for their less effective contribution. This imbalance in layer contributions likely limits the overall capacity of the model and increases parameter redundancy.

This paper focuses on investigating this jump behavior. Figure 1 shows an overview of the study presented in this paper. We first introduce a metric to quantitatively compare the jump strength in and around the final layers (Section 3.1). We then demonstrate that such a jump behavior is often observed in many open-weight pre-trained models, including well-known and widely used ones (Section 3.2), and tends to become increasingly pronounced as pre-training progresses (Section 3.3). We assume that this jump indicates an undesirable property of pre-trained models and thus propose a regularizer, referred to as **Jump-Suppressing Regularizer (JREG)**, that strongly penalizes the emergence of the jumps in and around the final layer during pre-training (Section 4), and indirectly encourages more balanced usage of the model's capabilities across the middle layers.

To empirically investigate our research question, that is, *whether mitigating the jump behavior in and around the final layer of Transformer-LMs and encouraging more even utilization across the middle layers can improve overall model capacity and leads to performance improvements*, we conduct experiments on three sizes of Llama-based architectures (170M, 1B and 3.4B parameters) under two pre-training regimes: (1) using the standard cross-entropy loss only, which serves as the baseline for comparison, and (2) using the standard cross-entropy loss combined with JREG. We evaluate each model based on both (a) the jumping behaviors around the final layers and (b) performance on widely used downstream tasks. The empirical results and our analyses reveal that incorporating JREG can improve downstream task performance and enhance the capabilities of models without altering the model architecture, by strongly penalizing jump behaviors around the final layers, which also mitigates ineffective middle layers in Transformer-LMs.

2   Related Work

Layer redundancy.   Transformer-LMs often exhibit layer redundancy, where multiple layers may learn similar or redundant operations. Models like Universal Transformer (Dehghani et al., 2019) and ALBERT (Lan et al., 2020), which share parameters across layers, achieve high parameter efficiency. The linear relationship between the layer input and the output vectors (Razzhigaev et al., 2024), as well as the small angular distance between layers (Sun et al., 2025a; Tyukin et al., 2024), indicate that pre-trained models exhibit layer redundancy. These findings substantiate the hypothesis that Transformer layers can learn nearly identical operations. The location of these redundant layers appears to be architecture dependent. For example, redundancy may appear in shallower layers in some models (Sajjad et al., 2023), while in others, such as Llama and GPT, removing deeper layers (except for the final layers) causes only a slight performance degradation (Tyukin et al., 2024; Men et al., 2025; Sun et al., 2025a) as the output magnitude in Pre-LN Transformers grows with depth (Kedia et al., 2024; Xie et al., 2023; Sun et al., 2025b) Consequently, methods to reduce such redundancy have been proposed, including architectural modifications like Mix-LN (Li et al., 2025), approaches suppressing deep-layer hidden state norms (Sun et al., 2025b), and training techniques such as LayerDrop (Zhang and He, 2020; Fan et al., 2020; Elhoushi et al., 2024) and LayerShuffle (Sun et al., 2025b). Unlike these previous approaches, this paper introduces a regularization technique that suppresses final layer displacement, which also addresses the common issue of underutilized middle layers and improves downstream task performance without changing the model architecture.

Early exit.   To reduce the per-token latency of Transformer-LMs, a large body of work equips middle layers with the ability to produce final predictions and halt computation once a satisfactory confidence threshold is met. This idea was first popularized in convolutional nets by BRANCHYNET (Teerapittayanon et al., 2016) and later inspiring a series of BERT-style architectures such as PABEE (Zhou et al., 2020), DEEBERT (Xin et al., 2020), and FASTBERT (Liu et al., 2020). These methods attach lightweight classifiers to each layer, train them with knowledge distillation to approximate the final output, and decide at run time whether to exit early according to some criteria. For autoregressive language models, early exit must preserve sequence consistency and hidden state caching. CALM (Schuster et al., 2022) introduces confi-

dence adaptive halting. More recently, LAYER-SKIP (Elhoushi et al., 2024) reuses the original LM head at every layer, whereas Jiang et al. (2025) adds separate linear heads; both apply per-layer cross-entropy losses. Whereas early exit techniques aim to maximise middle layers' performance to match that of the final layer, our study focuses on the excessive jump observed in the final layer of many open-weight models.

Internal trajectory. Analyzing the trajectories of hidden state vectors at each Transformer-LM layer and their projections into the vocabulary space is important from both a fundamental perspective, namely, understanding internal mechanisms, and an applied perspective, that is, improving performance and ensuring safety. By projecting each layer's hidden states into the vocabulary space, the inference process inside a Transformer-LM can be visualized, and it has been observed that the outputs of the middle layers gradually converge toward the final layer's predicted word (Belrose et al., 2023; Elhoushi et al., 2024). Instead of relying solely on the final layer at inference time, extrapolating the token probability trends over the last few layers to obtain a more mature predictive distribution has been shown to mitigate hallucinations (Das et al., 2025). It has been reported that models with backdoors exhibit unnatural trajectories of hidden state vectors, and that applying a small amount of fine-tuning to correct these trajectories can neutralize the backdoor (Min et al., 2025). Taken together, internal trajectories are important targets for both basic and applied research, and their analysis and control can contribute to improving factuality and ensuring safety. Accordingly, we address the characteristic "jump" observed in the final layer hidden states of pre-trained models through a trajectory regularizing objective.

3 Final Layer Hidden State Jump

In the following, we introduce an analytical framework to elucidate the inter-layer displacement characteristics that are consistently observed in open-weight large language models. First, we introduce the displacement Ψ_ℓ, a metric that quantifies the magnitude of hidden state variation in each layer, jump rate ζ_ℓ, a metric that compares displacement magnitudes between the middle and final layers. Using the Ψ_ℓ and ζ_ℓ, this study empirically demonstrates a property common to open-weight models, namely, "jump", in which the displacement of the

final layer markedly exceeds that of the middle layers.

3.1 Definition of displacement and jump rate

We formalize the layer-wise transformation of hidden state vectors in Transformer-LMs. Hereafter, let h denote a D-dimensional vector, that is, $h \in \mathbb{R}^D$. Let $h_{\ell-1}$ be the input hidden state vector of layer ℓ, and simultaneously the output hidden state vector to layer $\ell - 1$, where L denote the total number of layers and the relation $1 \leq \ell \leq L$ holds. We can represent the relationship between the input and output hidden state vectors as $h_\ell = \mathtt{TL}_\ell(h_{\ell-1})$, where $\mathtt{TL}_\ell(\cdot)$ denotes the ℓ-th Transformer layer, which primarily consists of an attention mechanism and a feed-forward network, along with layer normalizations. Moreover, for $\ell = 0$, h_0 represents the word embedding.

We geometrically interpret hidden state transitions. Transformer-LMs generate text via an autoregressive next-token prediction process. Viewed geometrically, each token prediction traces a trajectory in the embedding space that starts from the input layer embedding, h_0, and terminates near the output layer embedding. Building on this trajectory perspective, we characterize each step of the path by the angular change between consecutive hidden state vectors, i.e., $(h_0, h_1, \ldots, h_L)$. High-dimensional geometry defies direct visual intuition. To obtain a tractable scalar summary, we therefore introduce the displacement metric, which we construct in accordance with prior work (Tyukin et al., 2024; Li et al., 2025; Simoulin and Crabbé, 2021; Godey et al., 2024) by adopting cosine similarity based measures. Let $S_C(a, b)$ be the cosine similarity between two arbitrary vectors a and b, expressed as:

$$S_C(a, b) = \frac{a^\top b}{\|a\|_2 \|b\|_2}, \qquad (1)$$

where $\| \cdot \|_2$ represents the Euclidean norm of a given vector.

Definition 1 (Displacement between the input and output hidden state vectors). *We define the displacement between the input and output hidden state vectors at layer ℓ, denoted as Ψ_ℓ, as follows:*

$$\Psi_\ell = 1 - \frac{1}{2}\big(1 + S_C(h_{\ell-1}, h_\ell)\big). \qquad (2)$$

By construction, $\Psi_\ell \in [0, 1]$ and a larger Ψ_ℓ indicates a greater change in the angular distance between $h_{\ell-1}$ and h_ℓ.

(a) Llama-3.2-1B

(b) Llama-3.2-70B

(c) Gemma-2B

(d) Gemma-7B

(e) DeepSeek-7B

(f) DeepSeek-67B

(g) Qwen3-32B
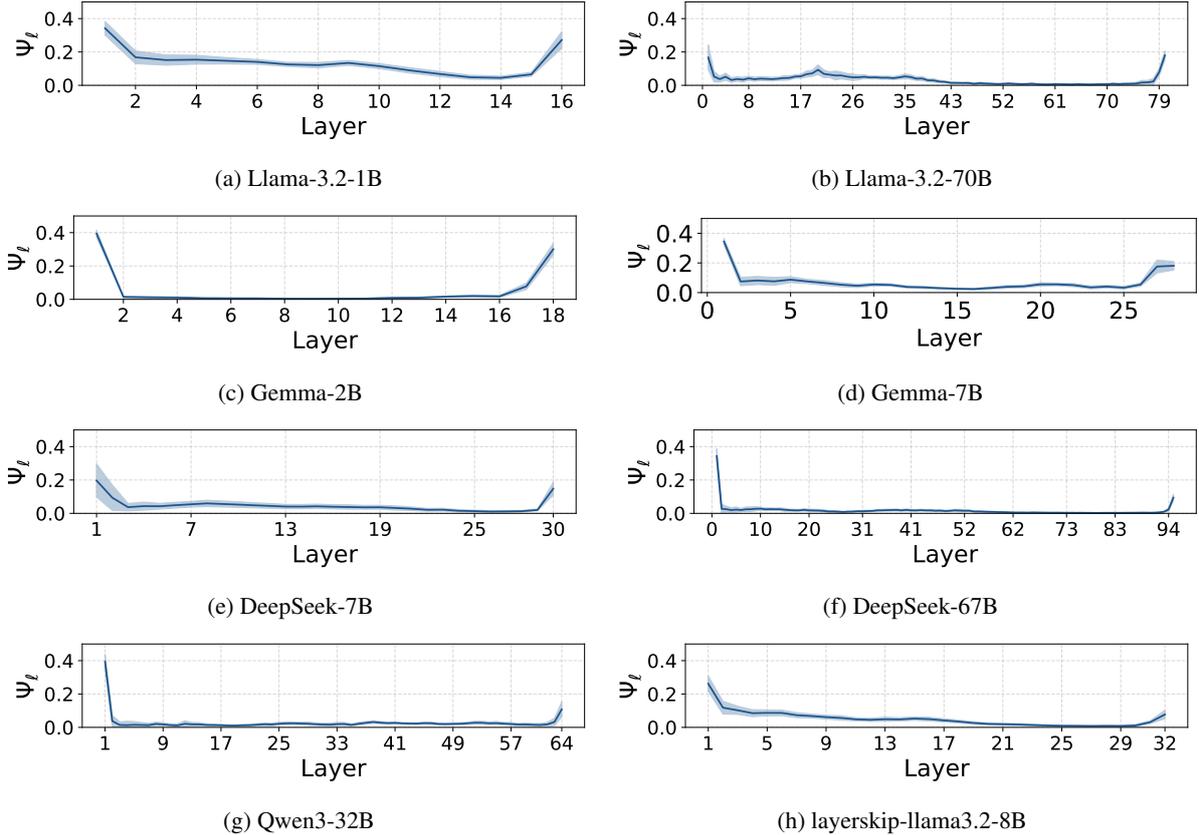

(h) layerskip-llama3.2-8B

Figure 2: Layer-wise hidden state displacement Ψ_ℓ for next-word prediction on 100 samples from the LAMBADA dataset. Across all model architectures, the displacement at the final layer tends to be larger than that of the middle layers.

Next, we quantify the jump rate ζ_ℓ, which measures the cumulative positive increase in displacement from layer $\ell - 1$ up to the final layer, scaled by 100.

Definition 2 (Jump rate). *For a model with L layers where $2 \leq \ell \leq L$, we define the jump rate ζ_ℓ as*

$$\zeta_\ell = \sum_{k=\ell}^{L} \max(0, \Psi_k - \Psi_{k-1}) \times 100 \quad (3)$$

For example, ζ_ℓ will take the minimum value of 0, i.e., $\zeta_\ell = 0$ if the displacement at layers k and $k-1$, where $k \in \{\ell, \ldots, L\}$, satisfy the non-increasing condition (i.e., $\Psi_{k-1} \geq \Psi_k$). Moreover, while the range of Ψ_ℓ is $[0, 1]$, and most Ψ_ℓ values in the middle layers are smaller than 0.1, $\zeta_\ell \geq 10$ are considered to be significantly larger values.

3.2 Analysis of open-weight models

We measured the displacement Ψ_ℓ across multiple open-weight models of various sizes to show that these models exhibit a pronounced hidden state jump near the final layer. Specifically, we evaluated the Llama-3.2-1B, Llama-3.2-70B (Grattafiori

Model	ζ_L	ζ_{L-1}	ζ_{L-2}
Llama-3.2-1B	20.6	22.7	22.7
Llama-3.2-70B	10.1	15.7	16.2
Gemma-2B	22.2	28.2	28.2
Gemma-7B	0.48	12.7	14.8
DeepSeek-7B	13.6	13.5	12.7
DeepSeek-67B	7.16	8.48	8.87
Qwen3-32B	7.50	9.00	9.36
layerskip-llama3.2-8B	4.62	6.64	6.94

Table 1: Jump rates ζ_L, ζ_{L-1}, and ζ_{L-2} of the six open-weight models. All models exhibit a pronounced hidden state jump at the final layer.

et al., 2024b), Gemma-2B, Gemma-7B (Team et al., 2024), DeepSeek-7B, DeepSeek-67B (DeepSeek-AI et al., 2024), Qwen3-32B (Yang et al., 2025) and Layerskip-Llama3.2-8B (Elhoushi et al., 2024). Layerskip is a multi-exit model based on the Llama architecture that adds additional classifiers to each Transformer layer and jointly optimizes all layers with the same cross-entropy loss as the final layer to enable early exit inference.

Figure 2 shows the displacement Ψ_ℓ for each model, computed as the mean per-token dis-

(a) Pythia 1B

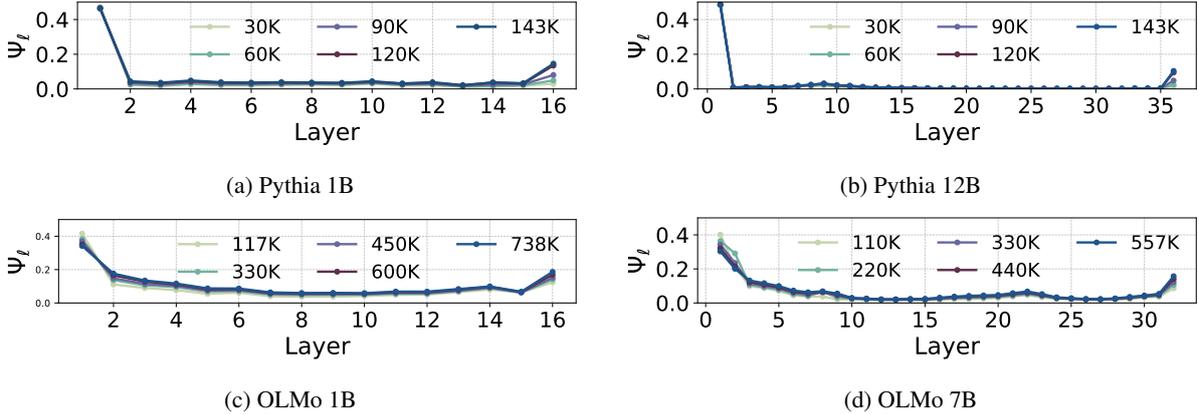(b) Pythia 12B

(c) OLMo 1B

(d) OLMo 7B

Figure 3: Analysis of checkpoint-wise hidden state displacement using fine-grained Pythia and OLMo pre-training checkpoints, revealing that as training progresses, the final layer exhibits large "jump" displacements.

Model	Size	20%	40%	60%	80%	100%
Pythia	1B	1.03	2.79	5.37	10.4	11.2
	12B	1.10	2.23	4.55	9.11	10.0
OLMo	1B	6.06	7.84	8.84	10.3	11.9
	7B	4.94	6.40	7.63	9.14	10.3

Table 2: Jump rate values ζ_L for the Pythia and OLMo models, whose pre-training checkpoints are publicly available, measured at 20%, 40%, 60%, 80%, and 100% (the final checkpoint) of the total training steps. For each model, we confirmed that the jump rate increases monotonically as training progresses.

placement during inference on the LAMBADA dataset (Paperno et al., 2016) and Table 1 shows jump rate ζ_L, ζ_{L-1}, and ζ_{L-2} for each model. Across all models, the final layer exhibits significantly greater displacement than the middle layers. Additionally, deeper models tend to have jumps from one or two layers before the final layer as well. This observation suggests that large displacement in the final layer persists not only in conventional models but even in Layerskip, which is trained to equip its middle layers with vocabulary prediction performance on par with the final layer. A similar phenomenon is also observed on the Wikitext (Merity et al., 2017) and ARC-easy (Clark et al., 2018) datasets, indicating that this behavior is not specific to a particular dataset.[1]

3.3 Conditions for Jump Formation

We analyzed how displacement changes throughout pre-training using open-weight models for which intermediate checkpoints are available. We employed the Pythia (Biderman et al., 2023) and
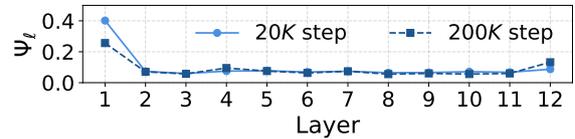

Figure 4: Hidden state displacement for the 170M Llama-based model pre-trained on 100B tokens with two different update steps. (light blue): 20K steps, (dark blue) 200K steps. Jump rates were $\zeta_L = 1.93$ for 20K steps and $\zeta_L = 7.24$ for 200K steps. Despite being trained on the same number of tokens, models with more update steps exhibit larger jumps in hidden state displacement.

OLMo (Groeneveld et al., 2024) to analyze intermediate checkpoints. The Pythia 1B and 12B models were trained on 300B tokens for 143,000 steps. We evaluate them every 30,000 steps. The OLMo 1B and 7B models were trained on 2T tokens for 738,000 steps and 2.46T tokens for 557,000 steps, respectively. We evaluate the 1B and 7B models every 150,000 and 110,000 steps[2], respectively.

Figure 3 shows the displacement values at each checkpoint and Table 2 shows these model jump rates. The jump rate grows monotonically with the number of updates, indicating that larger step counts yield larger ζ_L.

3.4 Effect of Training Steps on Jump Behavior

We investigated how the number of training steps influences the jump behavior in scratch pre-trained models to identify baseline training conditions that can reproduce the jump phenomenon observed in open-weight models. We examined two 170M

---

[1]Detailed results are shown in Section B.

[2]The final checkpoints for all models are not exactly at every 30,000, 150,000, or 110,000 steps. See Figure 3.
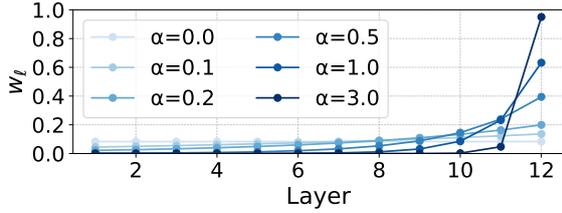
Figure 5: Weight coefficients w_ℓ for different values of the hyperparameter α in Equation (5). When $\alpha = 0.0$, the weight coefficients are uniform, when $\alpha = 0.1$, they are almost linear, and for larger α, they increase exponentially.

Llama-based models.[3] These models were pre-trained on the same corpus of 100B tokens but with distinct step counts by varying the batch size.

Figure 4 shows the displacement for models trained for 20K and 200K steps. The resulting jump rates are $\zeta_L = 1.93$ for 20K steps and $\zeta_L = 7.24$ for 200K steps. These results show that, even with an identical token budget, a greater number of update steps produces a larger jump and hence the properties of displacement more closely resemble those of open-weight models.

4 Proposed Method

As discussed in Section 3.2, the increased hidden state displacement at the final layer reflects abrupt changes in the angular distance. Motivated by this observation, this paper focuses on explicitly modeling and controlling this behavior through the design of an appropriate training objective.

Previous studies have shown that (1) the angular distance between consecutive hidden states in the middle layers is typically small (Sun et al., 2025a; Tyukin et al., 2024), and (2) layers that exhibit small angular distances can often be pruned without reducing the performance of the model (Men et al., 2025). (3) modifying the architecture so as to increase the angular distance change between the input and output hidden states in middle layers can improve model performance (Sun et al., 2025b; Li et al., 2025). These findings suggest that small angular distances are closely associated with redundancy and underutilization in the middle layers.

Building on these findings, we hypothesize that the phenomenon of jump may indicate that the network relies too heavily on the final layer while underutilizing the middle layers. To test this hy-

---

[3]The bacic experimental setup can be fonund in Section 5.1.

pothesis, we add a penalty on the final layer displacement to the loss function and experimentally evaluate the performance of models that suppress this jump.

Let $\mathcal{V}$ be the vocabulary, and let $|\mathcal{V}|$ denotes the number of tokens in the vocabulary. In general, pre-training minimizes the cross-entropy loss, $\mathcal{L}_{\text{CE}}$, that is:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{|\mathcal{V}|} y_i \log p_i, \qquad (4)$$

where y_i is the i-th element of the one-hot vector $\boldsymbol{y} \in \{0, 1\}^{|\mathcal{V}|}$ that represents the ground-truth token id, and p_i is the i-th element of the vector $\boldsymbol{p} = \texttt{softmax}(\boldsymbol{W}\texttt{RMS}(\boldsymbol{h}_L))$. Here, $\boldsymbol{h}_L$ denotes the final layer hidden state vector, and the operator $\texttt{RMS}(\cdot)$ refers to RMSNorm (Zhang and Sennrich, 2019), which rescales the hidden state vectors. Finally, the matrix $\boldsymbol{W} \in \mathbb{R}^{|\mathcal{V}| \times D}$ projects this normalized vector to the unnormalized vocabulary (logits) vector, and the softmax function, $\texttt{softmax}(\cdot)$, then converts the logits vector into a valid probability distribution, $\boldsymbol{p}$. Obviously, the cross-entropy loss, $\mathcal{L}_{\text{CE}}$, considers only output performance and does not, in any way, optimize for internal states. Here, in addition to $\mathcal{L}_{\text{CE}}$, we introduce a loss function $\mathcal{L}_{\text{disp}}$ that suppresses the displacement in the final layer:

$$\mathcal{L}_{\text{disp}} = \sum_{\ell=1}^{L} w_\ell \Psi_\ell, \qquad (5)$$

where w_ℓ is the ℓ-th element of $\boldsymbol{w} = \texttt{softmax}(\alpha \boldsymbol{l})$, and $\boldsymbol{l}$ is the L-dimantional vector, where $\boldsymbol{l} = (1, 2, \dots, L)$. Note that, intuitively, adding a regularization term over Ψ_L would eliminate the pronounced displacement in the final layer, however confining the regularizer to the final layer simply shifts this displacement to the penultimate layer, which would impede our ability to investigate the relationship between the final layer displacement and model performance.[4] Therefore, we employed Ψ_ℓ for all layers in our regularizer, but weighted layers closer to the final layer more heavily. Figure 5 shows examples of the w_ℓ distribution over the layer index ℓ for each α. As α increases, w_L approaches 1, focusing the strongest penalty on the final layer displacement Ψ_L, and yields uniform weights when $\alpha = 0$. Finally, the loss function of the proposed Jump-Suppressing Regularizer (JREG) is defined as follows:

$$\mathcal{L}_{\text{JREG}} = \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{disp}}, \qquad (6)$$

---

[4]Details are discussed in Section D.

where the hyperparameter λ is employed to adjust the relative importance of the two loss functions.

5 Experiments

In the following, we outline the experimental settings for applying the JREG method during pretraining. These experiments were performed to quantitatively evaluate the proposed method's impact on suppressing final layer displacement, the overall model performance, and further assess the performance after supervised fine-tuning (SFT) [5].

5.1 Pre-training from scratch

In this study, all experiments employed Llama-based models, and three variants differing in terms of the parameter count and depth were selected. The 170M model comprises 12 layers with a model dimension of 768, the 1B model comprises 16 layers with a model dimension of 2048, and the 3.4B model comprises 30 layers with a model dimension of 3072.[6] This difference in depth facilitated the evaluation of how both representation scale and depth influence the effectiveness of the proposed JREG method. For pretraining data, we use 100B tokens extracted from the FineWeb-Edu (Penedo et al., 2024). We set the total number of parameter update steps to 200K to make the "jump" more pronounced in the baseline model.

5.2 Evaluation metrics

Downstream task performance. For our evaluation, we employed several downstream tasks. We measure next-token prediction performance on the LAMBADA in term of the top-1 accuracy (Paperno et al., 2016). In addition, we assessed the accuracy of binary-choice QA on BoolQ (Clark et al., 2019) and multiple-choice QA on ARC-easy (ARC-e) (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), RACE (Lai et al., 2017), Social IQA (Sap et al., 2019), SciQ (Welbl et al., 2017), SWAG (Zellers et al., 2018).

Jump rate. We measure the improvements in the jump rates ζ_L, ζ_{L-1}, and ζ_{L-2} in Equation (3).

5.3 Supervised fine-tuning (SFT)

Training settings We performed SFT on the 3.4B model, updating all parameters during training. For the SFT data, we employed the Tulu-v1 SFT mixture dataset (Wang et al., 2023) which contains diverse instruction-following examples. [7]

Evaluation metrics We use three benchmark, MT-bench (Zheng et al., 2023), Vicuna bench (Zheng et al., 2023) and WizardLM testset (Xu et al., 2024). For evaluation, we used GPT-4 as the judge model. [8]

6 Results

Unless otherwise stated, we fix $\lambda = 1.0$.[9] for all experiments. We vary α as described in each subsection.

6.1 Pre-training on 170M model

For the 170M model, we conducted a sweep over the JREG hyperparameter α, testing values $\alpha \in \{0.0, 0.1, 0.3, 0.5, 1.0, 3.0\}$, to evaluate the effect of the weighting parameter.

As shown in Table 3, the proposed JREG method consistently outperformed the baseline in average downstream task performance. The performance tended to peak or stabilize with $\alpha \in \{1.0, 3.0\}$. In addition, statistical validation confirmed that the observed performance improvements were significant[10] and not dependent on the specific training dataset[11].

For the jump rates ζ_L, ζ_{L-1}, and ζ_{L-2}, Table 4 indicates that increasing α led to a substantial reduction. This suppression of the jump rates, which is attributed to JREG's stronger penalty on deeper layer displacement (Figure 5), correlates with improved downstream performance. These findings suggest that suppressing displacements around the final layers improves the model's overall capability.

6.2 Pre-training on larger model

To evaluate how the proposed JREG method scales, we applied it to larger 1B and 3.4B models, and performed the same evaluations as in the 170M model. First, we applied the proposed JREG method to

---

Size	Method	α	ARC-e	BoolQ	HellaSwag	LAMBADA	PIQA	RACE	SocialIQA	SciQ	SWAG	**avg**
170M	Baseline	-	54.9	57.5	32.1	32.0	64.0	29.1	38.4	80.2	39.7	47.5
	JREG (ours)	0.0	55.7	58.3	32.4	**32.5**	**65.9**	29.6	38.6	81.3	**40.2**	48.3
		0.1	56.1	60.5	32.2	29.8	65.6	29.6	39.2	80.9	39.9	48.2
		0.3	56.0	59.2	32.2	31.7	65.0	29.1	38.1	**82.2**	40.0	48.0
		0.5	57.0	57.1	**32.5**	30.7	65.0	29.3	38.6	**82.2**	40.0	48.0
		1.0	**57.2**	60.0	32.1	31.7	65.2	**29.9**	38.8	81.1	**40.2**	**48.5**
		3.0	**57.2**	60.0	32.4	31.6	65.1	28.9	**40.0**	81.7	39.9	**48.5**
1B	Baseline	-	68.5	**61.4**	42.9	46.6	**72.4**	32.9	41.0	89.5	47.3	55.8
	JREG (ours)	1.0	69.4	**61.4**	**43.1**	**47.2**	71.8	35.3	**42.1**	88.6	**47.6**	**56.2**
		3.0	**70.6**	59.3	42.6	45.6	71.8	34.9	41.4	**91.2**	47.3	56.1
3.4B	Baseline	-	75.6	59.4	49.7	55.9	**76.5**	36.3	**43.9**	**93.4**	51.3	60.2
	JREG (ours)	1.0	**77.9**	**61.6**	**50.3**	**57.2**	75.2	**37.8**	42.5	93.0	**51.5**	**60.8**

Table 3: Results of downstream task performance. α indicates the hyperparameter of JREG. Each row shows the score on each benchmark dataset, with its average in the rightmost column. The upper table shows the results of models trained with $\mathcal{L}_{\text{CE}}$ (baseline) and $\mathcal{L}_{\text{JREG}}$ (ours) using a 170M model, while the lower table shows those trained with the 1B and 3.4B model.

	Model	α	ζ_L	ζ_{L-1}	ζ_{L-2}
	Baseline	-	7.24	7.63	7.63
170M	JREG (ours)	0.0	0.84	1.28	1.76
		0.1	0.48	0.72	1.05
		0.3	**0.00**	0.05	0.15
		0.5	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	1.55
1B	Baseline	-	5.42	5.66	5.66
	JREG (ours)	0.5	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	**0.00**
3.4B	Baseline	-	5.21	6.15	6.25
	JREG (ours)	1.0	**0.00**	**0.00**	**0.00**

Table 4: Comparison of jump rates ($\zeta_L, \zeta_{L-1}, \zeta_{L-2}$) as defined in Equation (3) for baseline models and models trained with JREG. Results are shown for different hyperparameter α settings on 170M, 1B and 3.4B models. JREG effectively suppresses jump rates near the final layers.

1B models with top performing settings found on the 170M model, $\alpha \in \{1.0, 3.0\}$, and evaluate its effectiveness with jump rate ζ_L, downstream task performance.

Table 3 shows downstream task performance for a 1B model after pre-training. For all settings of α, the average downstream task performance exceeds the baseline. However, while the 170M model achieved identical performance for $\alpha \in \{1.0, 3.0\}$, the 1B model attained its highest performance at $\alpha = 1.0$. We attribute this difference to the increased number of layers in the larger model, which shifts the optimal hyperparameter α relative to the 170M model.

Table 4 compares the jump rate ζ_L of the 1B models between the baseline and JREG methods. The jump rates ζ_L are 6.42 with the 1B baseline models, whereas in JREG, they are 0.0. Thus, similar to the 170M model, these results confirm that the proposed method suppresses the final layer hidden state jump.

Furthermore, we extended our experiment to the 3.4B model, comparing the baseline and the JREG method with $\alpha = 1.0$, the best setting from 1B. Similar to the 1B model, JREG improves downstream performance and suppresses the jump rate over the baseline.

6.3 Supervised fine-tuning on 3.4B model

To further assess the downstream task performance of our pre-trained models, we performed SFT.

Table 6 shows the results on three benchmarks for the 3.4B baseline and JREG ($\alpha = 1.0$) models after SFT. The proposed JREG method outperformed the baseline on all benchmarks, with average scores of 4.28 and 4.67 for the baseline and JREG method, respectively. These improvements are statistically significant[12]. Table 7 compares the jump rates ζ_L, ζ_{L-1} and ζ_{L-2} of the 3.4B models between the baseline and JREG method after SFT. The baseline model exhibits $\zeta_L = 5.14$, which indicates that the jump behavior at the final layer persisted even after SFT. In contrast, the models pre-trained with JREG exhibit ζ_L, ζ_{L-1}, and ζ_{L-2} equal to 0.0, which indicates that the jump behavior was eliminated entirely. Overall, these results

---

[12]Details are discussed in Section G.2

Layer index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Baseline	27.30	4.78	4.08	3.14	3.25	4.10	4.70	3.71	4.91	3.85	4.28	3.43	3.93	3.62	3.42
JREG (ours)	27.58	4.89	5.13	3.54	3.41	3.48	4.31	4.73	4.92	4.39	3.88	3.57	3.92	3.97	3.56
Δ_ℓ	**0.28**	**0.11**	**1.05**	**0.40**	**0.16**	-0.62	-0.39	**1.02**	**0.01**	**0.54**	-0.40	**0.14**	-0.01	**0.35**	**0.14**

(a) 3.4B Model (Layers 1–15)

Layer index	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Baseline	3.96	3.4	3.01	2.57	2.22	2.27	2.05	1.71	1.60	1.48	1.25	1.33	1.43	2.37	7.58
JREG (ours)	3.74	3.39	3.12	2.57	2.47	2.07	2.13	1.86	1.76	1.86	1.37	1.11	0.71	0.40	0.18
Δ_ℓ	-0.22	-0.01	**0.11**	0.00	**0.25**	-0.20	**0.08**	**0.15**	**0.16**	**0.38**	**0.12**	-0.22	-0.72	-1.97	-7.40

(b) 3.4B Model (Layers 16–30)

Table 5: Displacement Ψ_ℓ for the baseline and JREG ($\alpha = 1.0$) on the 3.4B model, and the displacement difference Δ_ℓ ($\times 10^{-2}$). $\Delta_\ell > 0$ indicates that JREG reduces redundancy at layer ℓ.

Model	Vicuna Bench	WizardLM testset	MT Bench	avg
Baseline	5.89	4.10	2.84	4.28
JREG ($\alpha = 1.0$)	**6.36**	**4.23**	**3.40**	**4.67**

Table 6: Results of different benchmark performances on 3.4B baseline and JREG ($\alpha = 1.0$). Each row shows the score on each benchmark dataset, with its average in the rightmost column.

Model	ζ_L	ζ_{L-1}	ζ_{L-2}
Baseline	5.14	6.11	6.20
JREG ($\alpha = 1.0$)	**0.00**	**0.00**	**0.00**

Table 7: Comparison of jump rates between 3.4B baseline and JREG with $\alpha = 1.0$ model after SFT. Consistent with the pre-training results, the JREG maintains a jump rate of 0.0 after SFT.

suggest that the impact of the JREG method during pre-training is preserved after SFT, resulting in models that do not exhibit jump behavior and improved downstream performance.

7 Analysis and Discussion

In the following, we quantitatively evaluate the degree of redundancy reduction in the middle layers achieved by JREG. We define the improvement in redundancy at layer ℓ, denoted as Δ_ℓ, as the difference in displacement between the JREG and baseline models.

$$\Delta_\ell = \Psi_\ell^{\text{JREG}} - \Psi_\ell^{\text{Baseline}}, \qquad (7)$$

Here, $\Delta_\ell > 0$ indicates that the proposed JREG method yields a larger displacement than the baseline, thereby signifying an improvement in redundancy at that layer. We compute Δ_ℓ for all layers ℓ to assess the degree of redundancy improvement across the network.

Table 5 shows the displacement Δ_ℓ between the JREG and the baseline on the 3.4B model [13]. The JREG models exhibit a tendency for smaller displacement near the final layers compared to the baseline ($\Delta_\ell < 0$), while the displacement in the middle layers tends to be larger ($\Delta_\ell > 0$). Specifically, from layers 1 through 26, excluding the 8th layer of the 26 layers, JREG shows greater displacement. This suggests that, while JREG reduced the overcontribution of the layers near the final layer, it increased the effectiveness of the middle layers, thereby enhancing their contribution relative to the baseline.

8 Conclusion

This study investigated the jump behavior, defined as the pronounced large change in angular distance between the input and output hidden state vectors, occurring in or around the final layer(s) in Transformer-LMs, and examined its impact on model capacity. We first revealed that this phenomenon is ubiquitous across recent Transformer-LMs, regardless of open-weight pre-trained models and models trained from scratch. We then confirmed that our proposal of JREG, which penalises large hidden state displacements near the final layer during pre-training, consistently (i) reduced the hidden state jumps in the final layer, (ii) increased the relative contribution of middle layers, and (iii) improved downstream task performance without altering the model architecture, in our experiments conducted on three model sizes of Llama-based architecture. These findings suggest that mitigating hidden state jumps offers a simple yet effective means to unlock the latent capacity of middle layers in Transformer-LMs.

---

[13] 170M and 1B displacement are discussed in the Section I

Limitations

Despite the positive findings, several limitations of this study should be addressed in future research. First, our experiments were conducted exclusively on Llama-based architectures. While Llama represents a prominent family of models in the field, it remains unclear whether the proposed method JREG would be equally effective for other architectural designs, such as Transformer variants with different normalization schemes, attention mechanisms, or entirely different model architectures. Second, JREG is depth-sensitive. Therefore, as the number of layers changes, the optimal hyperparameter values also vary, requiring re-tuning to achieve the best performance.

Ethical Considerations

In this study, we exclusively used publicly available datasets for pre-training, fine-tuning, and evaluation. In addition, we developed the language models entirely from scratch, avoiding the use of publicly available models. For the publicly available datasets and existing pre-trained models analyzed in this study (including those under terms such as the META LLAMA 3 COMMUNITY LICENSE), we strictly adhered to all applicable licenses. Given that the proposed JREG method is a framework for pre-training language models, the risk of ethical concerns is expected to be minimal.

During code development and writing, we used AI assistants, including language models. All the generated code snippets and texts are checked and modified by the authors for scientific integrity and accuracy.

Acknowledgments

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report.

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

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

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 2397–2430. PMLR.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2924–2936. Association for Computational Linguistics.

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

Souvik Das, Lifeng Jin, Linfeng Song, Haitao Mi, Baolin Peng, and Dong Yu. 2025. Entropy guided extrapolative decoding to improve factuality in large

language models. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 6589–6600. Association for Computational Linguistics.

DeepSeek-AI, :, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, and 69 others. 2024. Deepseek llm: Scaling open-source language models with longtermism. *Preprint*, arXiv:2401.02954.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 12622–12642. Association for Computational Linguistics.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Nathan Godey, Éric Villemonte de la Clergerie, and Benoît Sagot. 2024. Anisotropy is inherent to self-attention in transformers. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - Volume 1: Long Papers, St. Julian's, Malta, March 17-22, 2024*, pages 35–48. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024a. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024b. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Dirk Groeneveld, Iz Beltagy, Evan Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, and 24 others. 2024. Olmo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15789–15809. Association for Computational Linguistics.

Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, and Sivakanth Gopi. 2023. Phi-2: The surprising power of small language models. Microsoft Research Blog.

Jiachen Jiang, Jinxin Zhou, and Zhihui Zhu. 2025. Tracing representation progression: Analyzing and enhancing layer-wise similarity. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Akhil Kedia, Mohd Abbas Zaidi, Sushil Khyalia, Jungho Jung, Harshith Goka, and Haejun Lee. 2024. Transformers get stable: An end-to-end signal propagation theory for language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. 2024. Analyzing feed-forward blocks in transformers through the lens of attention maps. In *The Twelfth International Conference on Learning Representations*.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. 2025. The remarkable robustness of llms: Stages of inference? *Preprint*, arXiv:2406.19384.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard H. Hovy. 2017. RACE: large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 785–794. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut.

2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Pengxiang Li, Lu Yin, and Shiwei Liu. 2025. Mix-LN: Unleashing the power of deeper layers by combining pre-LN and post-LN. In *The Thirteenth International Conference on Learning Representations*.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6035–6044. Association for Computational Linguistics.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2025. ShortGPT: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20192–20204, Vienna, Austria. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Nay Myat Min, Long H. Pham, Yige Li, and Jun Sun. 2025. CROW: Eliminating backdoors from large language models via internal consistency regularization. In *Forty-second International Conference on Machine Learning*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin A. Raffel, Leandro von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Anton Razzhigaev, Matvey Mikhalchuk, Elizaveta Goncharova, Nikolai Gerasimenko, Ivan V. Oseledets, Denis Dimitrov, and Andrey Kuznetsov. 2024. Your transformer is secretly linear. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 5376–5384. Association for Computational Linguistics.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Comput. Speech Lang.*, 77:101429.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social iqa: Commonsense reasoning about social interactions. 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 2019, Hong Kong, China, November 3-7, 2019*, pages 4462–4472. Association for Computational Linguistics.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Antoine Simoulin and Benoît Crabbé. 2021. How many layers and why? an analysis of the model depth in transformers. In *Proceedings of the ACL-IJCNLP 2021 Student Research Workshop, ACL 2021, Online, JUli 5-10, 2021*, pages 221–228. Association for Computational Linguistics.

Bingqing Song, Boran Han, Shuai Zhang, Jie Ding, and Mingyi Hong. 2024. Unraveling the gradient descent dynamics of transformers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Alessandro Stolfo, Ben Peng Wu, Wes Gurnee, Yonatan Belinkov, Xingyi Song, Mrinmaya Sachan, and Neel Nanda. 2024. Confidence regulation neurons in language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2025a. Transformer layers as painters. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 25219–25227. AAAI Press.

Wenfang Sun, Xinyuan Song, Pengxiang Li, Lu Yin, Yefeng Zheng, and Shiwei Liu. 2025b. The curse of depth in large language models. *Preprint*, arXiv:2502.05795.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros,

Ambrose Slone, and 89 others. 2024. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*, pages 2464–2469. IEEE.

Curt Tigges, Michael Hanna, Qinan Yu, and Stella Biderman. 2024. LLM circuit analyses are consistent across training and scale. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Georgy Tyukin, Gbetondji J-S Dovonon, Jean Kaddour, and Pasquale Minervini. 2024. Attention is all you need but you don't need all of it for inference of large language models. *Preprint*, arXiv:2407.15516.

Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. 2024. Meta Lingua: A minimal PyTorch LLM training library.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023. How far can camels go? exploring the state of instruction tuning on open resources. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text, NUT@EMNLP 2017, Copenhagen, Denmark, September 7, 2017*, pages 94–106. Association for Computational Linguistics.

Alexander Wettig, Kyle Lo, Sewon Min, Hannaneh Hajishirzi, Danqi Chen, and Luca Soldaini. 2025. Organize the web: Constructing domains enhances pretraining data curation. In *Forty-second International Conference on Machine Learning*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 38–45. Association for Computational Linguistics.

Shufang Xie, Huishuai Zhang, Junliang Guo, Xu Tan, Jiang Bian, Hany Hassan Awadalla, Arul Menezes, Tao Qin, and Rui Yan. 2023. Residual: Transformer with dual residual connections. *CoRR*, abs/2304.14802.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2246–2251. Association for Computational Linguistics.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 93–104. Association for Computational Linguistics.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*

36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023.

Hang Zhou, Yehui Tang, Haochen Qin, Yujie Yang, Renren Jin, Deyi Xiong, Kai Han, and Yunhe Wang. 2024. Star-agents: Automatic data optimization with LLM agents for instruction tuning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.*

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian J. McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

A   Details of experimental settings

A.1   Model settings

	170M	1B	3.4B
Layers	12	16	30
Model Dim	768	2048	3072
FFN Dim	2048	5376	8192
Attention Heads	12	16	24
Key / Value Heads	12	16	24
Activation		SwiGLU	
Vocabulary Size		32000	

Table 8: Model settings.

A.2   Pre-training settings

Configuration	170M	1B	3.4B
lr	9×10^{-4}	7×10^{-4}	5×10^{-4}
local batch	128	64	32
global batch		512	
sequence len		1024	
weight decay		0.1	
epsilon		1×10^{-8}	
Optimizer	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$)		
clip		1.0	
scheduler		cosine	
warmup		1000	
lr_min_ratio		0.1	
cycle_length		1.0	

Table 9: Pre-training settings.

Table 9 details the hyperparameter configurations employed during training. The model was trained on a corpus of 100 billion tokens, and its performance was validated on a dataset of 11 million tokens. For our implementation, we used the Meta Lingua library (Videau et al., 2024) as the codebase for our pre-training experiments. The total computational budget varied by model size:

- 170M model: 16 hours on $4\times$ NVIDIA H100 (94GB) GPUs

- 1B model: 55 hours on $8\times$ NVIDIA H200 (141GB) GPUs

- 3.4B model: 159 hours on $8\times$ NVIDIA H200 (141GB) GPUs

A.3   Supervised fine-tuning (SFT) settings

The maximum learning rate is set to 2×10^{-5} with a linear scheduler, and the optimizer is AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$). We use a global batch size of 512 and train for a total of 957 steps. The total computational budget is 2 hours on $8\times$ NVIDIA

H200 GPUs. For training, we employ huggingface transformers (Wolf et al., 2020), for evaluation, we employ FastChat (Zheng et al., 2023).

A.4   Evaluation settings for Pre-trained Models

For evaluating performance on downstream tasks described in Section 5.2, we utilized the `eval_harness` framework (Gao et al., 2024), which provides standardized implementations for a wide range of NLP benchmark tasks. All evaluations were conducted following the default hyperparameter configurations specified in the `eval_harness` framework, ensuring consistency and comparability with prior work.

B   Displacement on other dataset

In order to examine the displacement properties of open-weight models on datasets with distributions different from LAMBADA, we conducted experiments analogous to those in Section 3.2 on the test splits of Wikitext (Merity et al., 2017) and ARC-Easy (Clark et al., 2018).

B.1   Wikitext

Figure 6 shows the results of computing displacement using Wikitext.

B.2   ARC-Easy

Figure 7 shows the results of computing displacement using ARC-Easy.

## C   Non-Final Hidden State Jump Models

In this section, we examine two models, Gemma-7B and Phi2 (Javaheripi et al., 2023), that exhibit hidden states jumps in layers other than the final one (see Figure 8). In Gemma-7B, the largest jump occurs at the penultimate layer, while Phi2 shows a subtler jump at the third-to-last layer. We attribute these behaviors to each model's atypical training strategy. Gemma-7B gradually increases the proportion of high-quality, internally filtered data as training nears completion, whereas Phi2 initializes from the weights of a smaller, pretrained model. Thus, although these alternative strategies eliminate the pronounced final layer hidden state jump, they do not prevent all near final layer hidden state jumps.
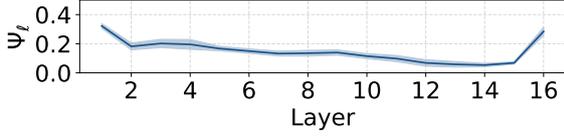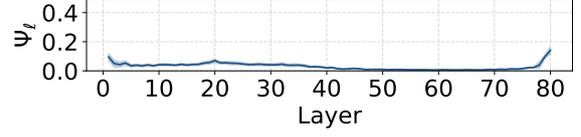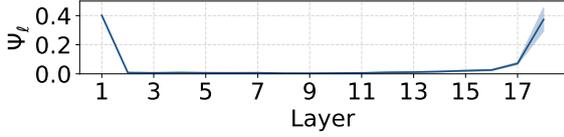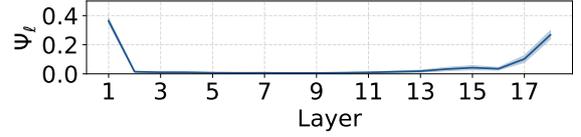
(a) Llama3.2-1B
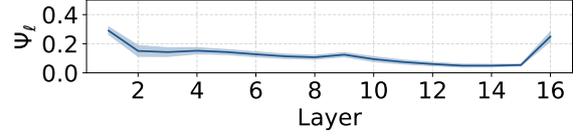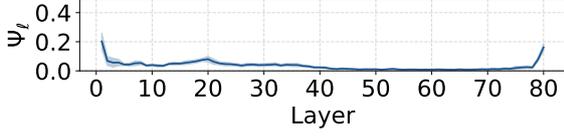
(b) Llama3.2-70B

(c) Gemma-2B
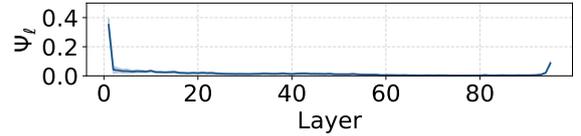
(d) DeepSeek-67B
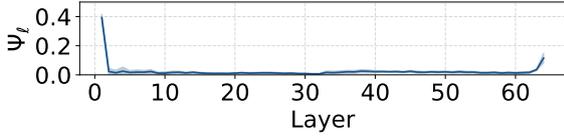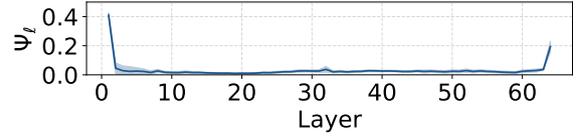
(e) Qwen3-32B
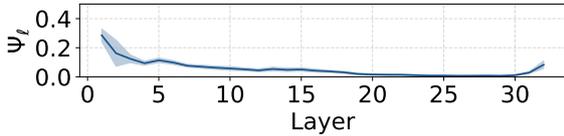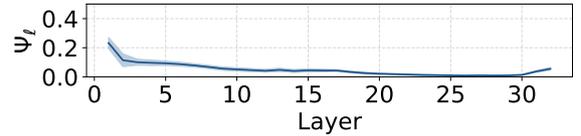
(f) layerskip-llama3.2-8B

Figure 6: Displacement on Wikitext.

(a) Llama3.2-1B
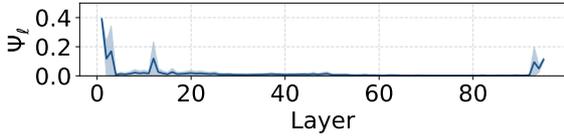

(b) Llama3.2-70B

(c) Gemma-2B

(d) DeepSeek-67B

(e) Qwen3-32B

(f) layerskip-llama3.2-8B

Figure 7: Displacement on ARC-easy.

D   Regularizing Only The Final Layer Displacement

While our main experiments use a weighted loss function that applies penalties across all layers with exponentially increasing weights (Equation (5)-Equation (6)), we also explored a simplified variant that focuses exclusively on the final layer displacement. In this variant, we replace the $\mathcal{L}_{\text{disp}}$ term in Equation (6) with:

$$\mathcal{L}_{\text{disp}} = \Psi_L \qquad (8)$$

This approach directly penalizes only the final layer displacement, ignoring middle layers. Equation (8)

are equivalent to $\alpha \to \infty$ in w_ℓ (Equation (5)). As shown in Figure 9, while this approach successfully suppresses the displacement at the final layer (Ψ_L), the displacement "jump" shifts to the penultimate layer, resulting in an increased Ψ_{L-1} value. Our hypothesis is that overreliance on a specific layer (the final layer in open-weight models) gives rise to redundancy in the middle layers. However, in Equation (8), the point of overreliance merely shifts from the final layer to the penultimate one, making it an inappropriate setting for testing this hypothesis.
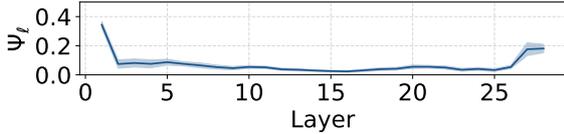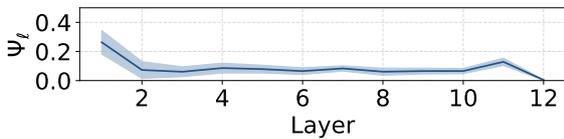
(a) Gemma-7B
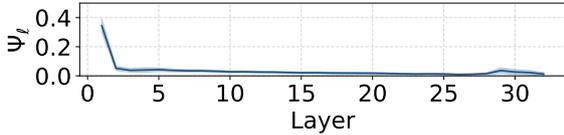
(b) Phi2

Figure 8: Layer-wise hidden state displacement Ψ_ℓ.

Figure 9: Hidden state displacement trajectory when applying regularization only to the final layer.

E Impact on Training Time and Memory

We analyze the change in training complexity introduced by JREG from the perspectives of training time and memory consumption during training. Table 10 shows a comparison of training time between the baseline and JREG ($\lambda = 1.0$, $\alpha = 1.0$) across different model sizes. As shown in the table, no significant change in training time is observed.

Next, Table 11 shows the increase ratio of memory consumption per step relative to the baseline. Memory consumption increases with larger model sizes. This is because, when computing Equation (5), the hidden states of each layer must be temporarily stored, and models with more layers require storing a greater number of hidden states.

In summary, while the introduction of JREG leads to an increase in memory consumption during training, it does not incur a substantial change in training time.

F Hyperparameter Tuning for JREG

The JREG loss function involves two hyperparameters, λ and α (see Eq. Equation (5), Equation (6)). While Section Section 6 of the main paper reported results only for $\lambda = 1.0$, this appendix evaluates 12 combinations of $\lambda \in \{1.0, 2.0, 3.0\}$ and $\alpha \in \{0.0, 0.1, 0.5, 1.0\}$ using a 170M model to identify the optimal λ. The results are summarized

Model		Training time (hours)
170M	Baseline	16
	JREG	16
1B	Baseline	55
	JREG	53
3.4B	Baseline	159
	JREG	159

Table 10: Comparison of training time between the baseline and JREG ($\lambda = 1.0, \alpha = 1.0$)

Model Size	Memory Increase Amount
170M	8.5%
1B	15.5%
3.4B	35.6%

Table 11: Increase in memory consumption with JREG ($\lambda = 1.0, \alpha = 1.0$) compared to the baseline

in Table 12.

The highest average performance was achieved with $(\lambda, \alpha) = (1.0, 1.0)$ and $(2.0, 1.0)$, and when averaging over α for each λ, the best overall performance occurs at $\lambda = 1.0$. Consequently, the main paper fixes $\lambda = 1.0$ and conducts additional experiments with α.

G Statistical Evaluation of Performance under Random Seed Perturbations

Section 6 shows the performance differences on the downstream task, and we further investigated their statistical significance by repeating the experiments with different random seeds.

G.1 Pre-trained Model Evaluation

In this subsection, we focus on the 170M pre-trained model, corresponding to the experiments presented in Section 6.1. The model architecture is shown in Table 8, and the training configuration was identical to that in Table 9, except for the random seed, which was varied across 10 settings. We used the JREG hyperparameters $\lambda = 1.0$ and $\alpha = 1.0$, which yielded the highest downstream task performance for the 170M models (Table 3).

Table 13 shows the downstream task performance of the models trained with each seed. To test whether the mean difference ($\mu_{\text{JREG}} - \mu$) is positive, we conducted a one-sample, one-side t-test. The mean performance difference was 0.41, and the p-value was 0.034, indicating statistical significance at the 5% level. The lower bound of the 95% one-sided confidence interval was 0.047. There-

λ	α	ARC-e	BoolQ	HellaSwag	LAMBADA	PIQA	RACE	SocialIQA	SciQ	SWAG	**avg**
1.0	0.0	55.7	58.3	32.4	32.5	65.9	29.6	38.6	81.3	40.2	48.3
	0.1	56.1	60.5	32.2	29.8	65.6	29.6	39.2	80.9	39.9	48.2
	0.5	57.0	57.1	32.5	30.7	65.0	29.3	38.6	82.2	40.0	48.0
	1.0	57.2	60.0	32.1	31.7	65.2	29.9	38.8	81.1	40.2	**48.5**
2.0	0.0	54.7	60.9	32.2	31.7	65.1	29.1	37.9	81.3	40.1	48.1
	0.1	56.6	61.1	32.0	31.9	65.7	28.2	38.3	80.9	40.0	48.3
	0.5	55.7	56.8	31.8	31.0	65.6	29.3	37.8	82.7	39.7	47.8
	1.0	57.1	56.4	32.3	32.8	65.9	30.1	39.5	82.5	40.1	**48.5**
3.0	0.0	56.4	58.7	31.8	32.2	64.8	30.1	38.2	81.5	39.9	48.2
	0.1	56.8	58.8	31.9	30.8	64.9	29.2	38.3	81.3	39.8	48.0
	0.5	57.3	57.0	32.1	30.7	65.5	30.2	39.2	82.1	39.8	48.2
	1.0	57.2	58.9	32.0	31.0	65.3	29.1	38.8	81.7	40.0	48.2

Table 12: Downstream task performance of the 170M model evaluated across 12 combinations of JREG hyperparameters $\lambda \in \{1.0, 2.0, 3.0\}$ and $\alpha \in \{0.0, 0.1, 0.5, 1.0\}$.

seed	Method	ARC-e	BoolQ	HellaSwag	LAMBADA	PIQA	RACE	SocialIQA	SciQ	SWAG	**avg**
42	Baseline	56.5	61.4	31.9	32.5	65.1	28.9	38.0	80.6	40.1	**48.3**
	JREG	57.0	56.2	31.9	32.6	65.5	28.6	38.6	80.9	40.0	47.9
123	Baseline	55.6	59.1	31.9	32.9	64.0	31.6	39.3	82.1	40	**48.5**
	JREG	56.4	56.0	32.2	31.3	65.1	29.8	37.5	82.1	40.0	47.8
777	Baseline	54.9	57.5	32.1	32.0	64.0	29.1	38.4	80.2	39.7	47.5
	JREG	57.2	60.0	32.1	31.7	65.2	29.9	38.8	81.1	40.2	**48.5**
888	Baseline	55.4	51.1	32.2	33.0	65.7	28.6	39.0	81.2	39.8	47.3
	JREG	57.3	60.8	32.1	32.1	64.4	28.0	39.4	81.6	40.1	**48.4**
2025	Baseline	55.4	57.9	32.3	32.1	65.0	28.5	39.7	80.9	40.0	48.0
	JREG	56.5	59.7	32.0	32.4	65.3	30.1	37.5	82.1	40.0	**48.4**
10000	Baseline	56.6	51.6	31.9	32.2	65.1	28.9	38.1	81.9	40.0	47.4
	JREG	55.9	61.7	31.9	31.7	64.5	29.2	39.2	82.5	39.7	**48.5**
65537	Baseline	55.9	60.4	32.2	30.0	65.2	29.9	38.7	79.8	39.9	48.0
	JREG	57.2	61.3	32.2	31.3	65.5	30.0	38.1	82.4	40.1	**48.7**
141421	Baseline	56.8	55.0	32.4	32.4	64.2	30.9	38.6	81.8	40.0	48.0
	JREG	58.2	59.4	32.3	31.7	64.3	30.0	38.9	82.1	39.8	**48.5**
271828	Baseline	55.2	58.7	32.4	32.4	64.9	29.5	38.4	80.9	40.3	48.1
	JREG	55.7	61.8	32.1	32.2	65.6	29.5	37.7	82.4	40.1	**48.6**
314159	Baseline	54.4	56.7	32.0	32.1	65.8	28.3	39.3	81.4	40.0	**47.8**
	JREG	55.9	53.9	32.1	33.1	64.1	29.7	38.4	82.3	39.9	47.7

Table 13: Results of evaluating downstream task performance for a 170M parameter model pre-trained on the Fineweb-edu corpus and trained with ten different random seeds.

fore, although the model's performance improvement is modest, it was shown to be statistically significant.

G.2 SFT Model Evaluation

In this subsection, we focus on the 3.4B model after SFT, corresponding to the experiments presented in Section 6.3. SFT configuration was identical to Section A.3 except for the random seed, which was varied across 5 settings.

Table 14 shows the downstream task performance of the models trained with each seed. The statistical testing procedure is the same as in Sec-

tion G.1. The mean performance difference was 0.46, and the p-value was 9.89×10^{-6}, indicating statistical significance at the 5% level. The lower bound of the 95% one-sided confidence interval was 0.42. Therefore, JREG consistently yields statistically significant improvements in the 3.4B model after SFT.

H Transition of Jump Rate During Training

In this section, we analyze the Jump rate (ζ_L, ζ_{L-1}, ζ_{L-2}) and displacement Ψ_ℓ during training

seed	Method	Vicuna-Bench	WizardLM testset	MT-bench	avg
123	Baseline	5.80	4.00	2.91	4.24
	JREG	**6.39**	**4.32**	**3.42**	**4.71**
777	Baseline	5.89	4.10	2.84	4.28
	JREG	**6.36**	**4.24**	**3.40**	**4.67**
888	Baseline	5.81	3.96	2.82	4.20
	JREG	**6.34**	**4.35**	**3.39**	**4.70**
2025	Baseline	5.89	4.03	2.88	4.27
	JREG	**6.36**	**4.36**	**3.42**	**4.71**
10000	Baseline	5.78	3.94	2.92	4.21
	JREG	**6.31**	**4.37**	**3.42**	**4.70**

Table 14: Results of evaluating different benchmarks for the 3.4B-parameter model after SFT across five random seeds.

at checkpoints corresponding to 25%, 50%, 75%, and 100% of training, i.e., at 50K, 100K, 150K, and 200K steps.

Table 15 shows the values of ζ_L, ζ_{L-1}, and ζ_{L-2} computed at each checkpoint, while Figure 10, Figure 11, and Figure 12 show the displacement Ψ_ℓ for the 170M, 1B, and 3.4B models. Similar to the characteristics observed in the open-weight models analyzed in 3, the baseline exhibits a monotonically increasing Jump rate as training progresses across checkpoints. In contrast, for models with JREG applied (with $\alpha \geq 0.5$ for the 170M model), we observe $\zeta_L = 0.00$, indicating that the training dynamics are altered.

I   Displacement across Different Weight Settings

In this section, we evaluate the degree of redundancy reduction in the middle layers on 170M and 1B models across different weight settings using Δ_ℓ metrics Equation (7).

Figure 13 shows Ψ_ℓ in graph and Table 16 shows Ψ_ℓ and Δ_ℓ for all α settings on 170M and 1B models. In the 1B model, both $\alpha = 1.0$ and $\alpha = 3.0$ settings, the displacement in the middle layers tends to be larger ($\Delta_\ell > 0$), a similar trend is observed in the 3.4B model (Section 7). For $\alpha = 1.0$, excluding layer 3 and 7, from layers 1 through 11 exhibit greater displacement under JREG, For $\alpha = 3.0$, excluding layer 7, from layers 1 through 14 exhibit greater displacement under JREG. However, in the 170M model, the effect of JREG is less pronounced, and the displacement increase is observed in fewer layers. Since the 170M model has only 12 layers, fewer than the 1B (16 layers) and 3.4B models (30 layers), the proportion of layers where $\Delta_\ell > 0$ is smaller compared to those larger models.

J   Result on Other Datasets

To examine the dataset dependency of JREG, we train models using WebOrganizer (Wettig et al., 2025). The base settings of the experiment were identical to those in Section 5 while the training was conducted on the FineWeb-Edu dataset.

Table 17 shows the downstream task performance. The average performance gap between the baseline and JREG was smaller when using the WebOrganizer dataset. When trained on the FineWeb-Edu dataset, the performance gap reached up to 1.0 for $\alpha = 1.0$ and $\alpha = 3.0$. In contrast, when trained on the WebOrganizer dataset, the performance gap were 0.6 for $\alpha = 1.0$ and 0.3 for $\alpha = 3.0$. As in Section G.1, we conducted a statistical test on the performance differences to examine their significance for $\alpha = 1.0$. The testing procedure followed the same protocol as described in Section G.1. The mean performance difference was 0.41, and the p-value was 0.0052, indicating statistical significance at the 5% level. The lower bound of the 95% one-sided confidence interval was 0.271. Therefore, the performance improvement achieved by applying JREG is statistically significant regardless of the choice of training dataset.

Table 19 shows displacement Ψ_ℓ and Δ_ℓ, Table 20 shows checkpoint-wise jump rates ζ_L, ζ_{L-1}, ζ_{L-2}, and Figure 10 shows checkpoint-wise displacement. Similar to the results using the FineWeb-Edu dataset, ζ_L remains at 0.00 for $\alpha \geq 0.3$, and the proportion of layers with $\Delta_\ell > 0$ increases as α increases.

K   Layer Skip Evaluation

In Table 5, we measured the reduction of layer redundancy by the displacement difference $\Delta_\ell =$

	Model	α	50k	100k	150k	200k
	Baseline	-	4.13	5.07	5.91	7.24
		0.0	0.48	0.38	0.55	0.84
		0.1	0.10	0.25	0.29	0.48
		0.3	**0.00**	**0.00**	**0.00**	**0.00**
170M	JREG	0.5	**0.00**	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	**0.00**	**0.00**
	Baseline	-	2.07	2.96	4.13	5.42
1B	JREG	1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	**0.00**	**0.00**
3.4B	Baseline	-	1.62	2.57	3.98	5.21
	JREG	1.0	**0.00**	**0.00**	**0.00**	**0.00**

(a) ζ_L

	Model	α	50k	100k	150k	200k
	Baseline	-	4.25	5.64	6.34	7.63
		0.0	0.56	0.56	0.87	1.28
		0.1	0.11	0.33	0.43	0.72
		0.3	**0.00**	0.03	0.00	0.05
170M	JREG	0.5	**0.00**	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	**0.00**	**0.00**
	Baseline	-	2.16	3.16	4.27	5.66
1B	JREG	1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	**0.00**	**0.00**
3.4B	Baseline	-	2.05	3.12	4.74	6.15
	JREG	1.0	**0.00**	**0.00**	**0.00**	**0.00**

(b) ζ_{L-1}

	Model	α	50k	100k	150k	200k
	Baseline	-	4.25	5.64	6.34	7.63
		0.0	1.13	1.05	1.30	1.76
		0.1	0.53	0.67	0.72	1.05
		0.3	0.08	0.09	0.07	0.15
170M	JREG	0.5	**0.00**	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	1.48	1.55	1.41	1.55
	Baseline	-	2.16	3.16	4.27	5.66
1B	JREG	1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	0.22	0.31	0.43	0.50
3.4B	Baseline	-	2.11	3.22	4.85	6.25
	JREG	1.0	**0.00**	**0.00**	**0.00**	**0.00**

(c) ζ_{L-2}

Table 15: Checkpoint-wise jump rate of models pre-trained on Fineweb-edu

$\Psi_\ell^{\text{JREG}} - \Psi_\ell^{\text{Baseline}}$. In this section, however, we assess redundancy from a different perspective by comparing the extent of performance degradation under layer skip between the baseline and the JREG-trained models. Layer skip is a technique that accelerates inference by terminating the forward pass at a middle layer and using that layer's output as the model prediction, instead of propagating to the final layer. Because JREG is designed to reduce redundancy in the middle layers, we expect it to achieve higher performance than the baseline

when inference is stopped at a middle layer.

To test this, we perform layer skip evaluation. Using the same 170M, 1B 3.4B parameter Llama-based architecture as in, we compare baseline and JREG models across different α settings. At inference time, instead of taking the final hidden state $\boldsymbol{h}_L$, we output the hidden state $\boldsymbol{h}_\ell$ ($0 \leq \ell \leq L$) from a middle layer and terminate decoding early. Performance is measured by validation loss on the FineWeb-Edu split.

Figure 15 shows the validation loss when infer-

Layer index	1	2	3	4	5	6	7	8	9	10	11	12
Baseline	25.63	7.02	5.72	9.48	7.52	6.31	7.39	5.53	6.00	5.59	5.98	13.22
JREG ($\alpha = 0.0$)	3.02	1.89	1.41	1.58	1.54	1.73	1.45	1.93	1.92	2.4	2.84	3.68
Δ_ℓ	-22.61	-5.13	-4.31	-7.9	-5.98	-4.58	-5.94	-3.6	-4.08	-3.19	-3.14	-9.54
JREG ($\alpha = 0.1$)	4.86	2.54	1.86	2.03	1.71	1.89	1.53	1.91	1.72	2.05	2.29	2.77
Δ_ℓ	-20.77	-4.48	-3.86	-7.45	-5.81	-4.42	-5.86	-3.62	-4.28	-3.54	-3.69	-10.45
JREG ($\alpha = 0.3$)	13.24	5.25	3.47	3.83	2.91	2.59	2.36	1.91	1.80	1.90	1.95	1.55
Δ_ℓ	-12.39	-1.77	-2.25	-5.65	-4.61	-3.72	-5.03	-3.62	-4.20	-3.69	-4.03	-11.67
JREG ($\alpha = 0.5$)	20.49	6.62	5.06	6.08	4.85	4.53	3.60	2.49	2.08	2.01	1.64	1.09
Δ_ℓ	-5.14	-0.40	-0.66	-3.40	-2.67	-1.78	-3.79	-3.04	-3.92	-3.58	-4.34	-12.13
JREG ($\alpha = 1.0$)	26.36	7.12	5.48	7.45	6.76	7.63	7.02	5.71	4.00	3.00	1.70	0.67
Δ_ℓ	0.73	**0.10**	-0.24	-2.03	-0.76	**1.32**	-0.37	**0.18**	-2.00	-2.59	-4.28	-12.55
JREG ($\alpha = 3.0$)	25.77	7.47	5.32	7.28	7.09	6.96	6.34	7.48	7.10	8.65	4.46	0.43
Δ_ℓ	**0.14**	**0.45**	-0.40	-2.20	-0.43	**0.65**	-1.05	**1.95**	**1.10**	**3.06**	-1.52	-12.79

(a) 170M Model

Layer index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Baseline	25.00	6.37	5.90	5.44	6.82	5.49	5.63	4.77	5.26	5.33	5.45	4.57	4.26	3.40	3.64	9.06
JREG ($\alpha = 1.0$)	28.04	6.68	5.68	6.05	7.53	5.79	5.57	5.29	5.65	6.41	5.70	4.14	2.83	1.67	0.88	0.39
Δ_ℓ	**3.04**	**0.31**	-0.22	**0.61**	**0.71**	**0.30**	-0.06	**0.52**	**0.39**	**1.08**	**0.25**	-0.43	-1.43	-1.73	-2.76	-8.67
JREG ($\alpha = 3.0$)	27.42	6.46	6.44	5.72	7.39	5.60	5.60	5.01	5.36	6.04	5.47	4.86	4.93	5.43	2.46	0.26
Δ_ℓ	**2.42**	**0.09**	**0.54**	**0.28**	**0.57**	**0.11**	-0.03	**0.24**	**0.10**	**0.71**	**0.02**	**0.29**	**0.67**	**2.03**	-1.18	-8.8

(b) 1B Model

Table 16: Displacement Ψ_ℓ for the baseline and JREG, and the displacement difference Δ_ℓ ($\times 10^{-2}$) on the 170M and 1B models.

Method	α	ARC-e	BoolQ	HellaSwag	LAMBADA	PIQA	RACE	SocialIQA	SciQ	SWAG	**avg**
170M Param. Model (Llama architecture)											
Baseline	-	43.5	55.4	31.3	37.9	66.1	28.6	37.7	75.8	41.4	46.4
JREG (ours)	0.0	44.2	56.9	31.4	35.8	65.5	28.5	37.8	74.6	41.3	46.2
	0.1	43.7	58.2	31.2	36.2	65.6	28.9	38.0	78.0	41.4	46.8
	0.3	43.7	59.3	31.5	36.1	66.7	29.4	39.4	78.8	41.5	47.4
	0.5	44.2	56.4	31.0	35.6	64.9	28.1	37.9	77.2	41.4	46.3
	1.0	42.7	61.0	31.4	37.6	64.6	30.0	37.9	75.7	41.8	47.0
	3.0	42.4	55.3	31.7	36.8	65.6	29.5	38.3	79.1	41.5	46.7

Table 17: Results of downstream task performance pre-trained on Weborganizer. α indicates the hyperparameter of JREG. Each row shows the score on each benchmark dataset, with its average in the rightmost column.

ence is stopped at each layer ℓ. Across all middle layers ($0 < \ell < L$), JREG achieves lower validation loss than the baseline. This indicates that JREG enhances the quality of middle-layer representations and utilizes parameters more effectively.

seed	Method	ARC-e	BoolQ	HellaSwag	LAMBADA	PIQA	RACE	SocialIQA	SciQ	SWAG	avg
123	Baseline	43.8	60.1	31.4	37.4	66.9	28.7	38.6	76.9	41.4	46.7
	JREG	43.6	61.0	31.1	37.8	65.8	75.8	41.4	79.3	41.8	**47.5**
777	Baseline	43.5	55.4	31.3	37.9	66.1	28.6	37.7	75.8	41.4	46.4
	JREG	42.7	61.0	31.4	37.6	64.6	30	37.9	75.7	41.8	**47.0**
888	Baseline	43	53.7	31.5	37.6	64.7	28.3	37.2	78.0	41.6	46.2
	JREG	44.3	57.7	31.3	38.3	65.6	27.9	37.9	77.5	41.7	**46.9**
2025	Baseline	43.4	55.3	31.2	36.3	65.5	28.5	37.2	80.4	41.6	46.6
	JREG	43.4	59.2	31.6	37.1	65.0	29.1	38.4	76.3	41.6	**46.9**
10000	Baseline	44.7	56.7	31.6	37.5	66.6	27.9	37.0	78.8	41.2	46.9
	JREG	44.9	59.4	31.3	36.6	65.1	28.7	38.6	78.3	41.3	**47.1**

Table 18: Results of evaluating downstream task performance for a 170M parameter model pre-trained on the WebOrganizer corpus and trained with five different random seeds.

Layer index	1	2	3	4	5	6	7	8	9	10	11	12
Baseline	27.19	6.72	6.43	6.90	6.28	8.59	5.24	5.76	6.14	4.96	5.09	12.41
JREG ($\alpha = 0.0$)	2.84	1.7	1.64	1.58	1.4	1.51	2.04	1.66	1.94	1.94	2.58	3.79
Δ_ℓ	-24.35	-5.02	-4.79	-5.32	-4.88	-7.08	-3.20	-4.10	-4.20	-3.02	-2.51	-8.62
JREG ($\alpha = 0.1$)	4.86	2.54	1.86	2.03	1.71	1.89	1.53	1.91	1.72	2.05	2.29	2.77
Δ_ℓ	-22.45	-4.38	-4.29	-4.97	-4.66	-7.04	-3.05	-4.26	-4.42	-3.29	-3.11	-9.71
JREG ($\alpha = 0.3$)	13.79	4.67	4.35	3.60	2.93	2.28	2.66	1.78	1.74	1.47	1.46	1.56
Δ_ℓ	-13.4	-2.05	-2.08	-3.30	-3.35	-6.31	-2.58	-3.98	-4.40	-3.49	-3.63	-10.85
JREG ($\alpha = 0.5$)	21.43	6.40	6.35	6.30	5.07	3.98	4.00	2.39	2.13	1.61	1.29	1.09
Δ_ℓ	-5.76	-0.32	-0.08	-0.60	-1.21	-4.61	-1.24	-3.37	-4.01	-3.35	-3.80	-11.32
JREG ($\alpha = 1.0$)	29.3	6.84	6.35	7.85	7.14	6.85	8.52	5.16	3.99	2.32	1.37	0.67
Δ_ℓ	**2.11**	**0.12**	-0.08	**0.95**	**0.86**	-1.74	**3.28**	-0.60	-2.15	-2.64	-3.72	-11.74
JREG ($\alpha = 3.0$)	27.14	7.21	5.99	8.01	7.23	6.12	8.01	6.16	7.54	7.68	3.47	0.46
Δ_ℓ	-0.05	**0.49**	-0.44	**1.11**	**0.95**	-2.47	**2.77**	**0.40**	**1.40**	**2.72**	-1.62	-11.95

Table 19: Displacement Ψ_ℓ for the baseline and JREG on 170M model trained on Weborganizer

Model		α	50k	100k	150k	200k
	Baseline	-	3.96	4.74	5.76	7.32
		0.0	0.49	0.53	0.9	1.21
		0.1	0.19	0.35	0.59	0.72
170M	JREG	0.3	**0.00**	0.03	0.11	0.10
		0.5	**0.00**	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	**0.00**	**0.00**

(a) ζ_L

Model		α	50k	100k	150k	200k
	Baseline	-	4.01	4.88	5.88	7.45
		0.0	0.77	0.91	1.41	1.85
		0.1	0.31	0.50	0.81	1.03
170M	JREG	0.3	**0.00**	0.03	0.11	0.10
		0.5	**0.00**	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	**0.00**	**0.00**	**0.00**

(b) ζ_{L-1}

Model		α	50k	100k	150k	200k
	Baseline	-	4.01	4.88	5.88	7.45
		0.0	0.77	0.91	1.41	1.85
		0.1	0.31	0.50	0.81	1.03
170M	JREG	0.3	**0.00**	0.03	0.11	0.10
		0.5	**0.00**	**0.00**	**0.00**	**0.00**
		1.0	**0.00**	**0.00**	**0.00**	**0.00**
		3.0	**0.00**	0.25	0.13	0.14

(c) ζ_{L-2}

Table 20: Checkpoint-wise jump rate of models pre-trained on Weborganizer
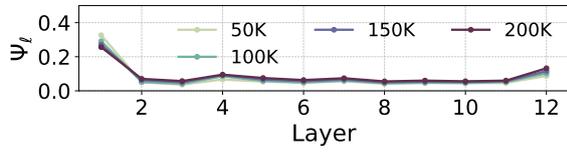
(a) Baseline
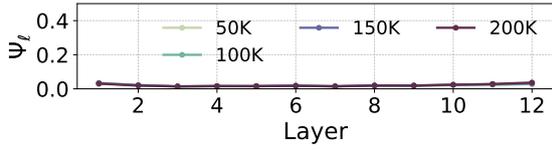
(b) JREG ($\alpha = 0.0$)
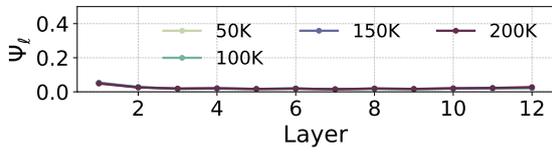
(c) JREG ($\alpha = 0.1$)
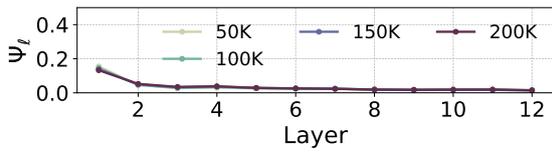
(d) JREG ($\alpha = 0.3$)
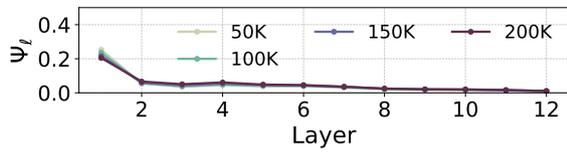
(e) JREG ($\alpha = 0.5$)
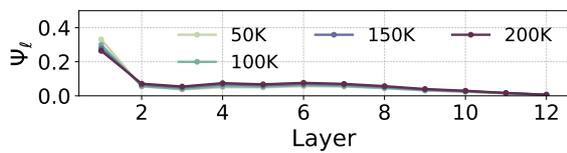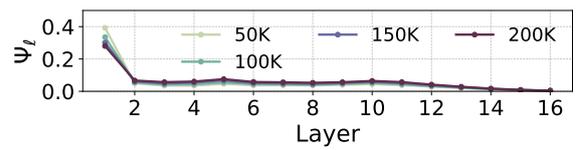
(f) JREG ($\alpha = 1.0$)
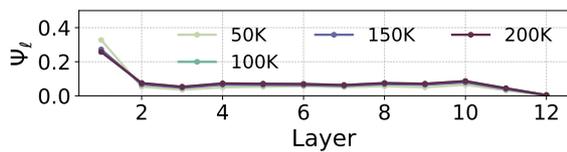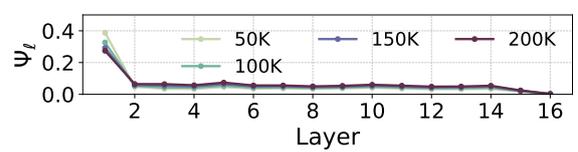
(g) JREG ($\alpha = 3.0$)

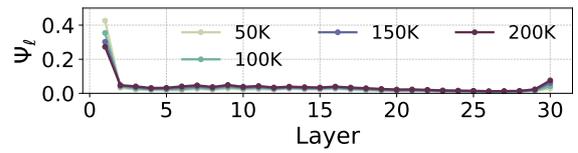Figure 10: Checkpoint-wise hidden state displacement on 170M model
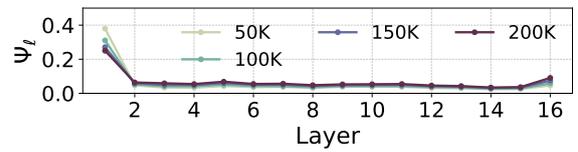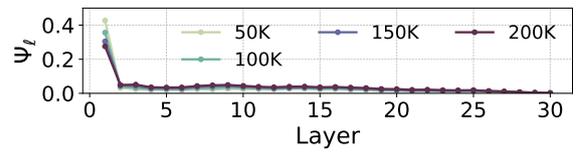

(a) Baseline

(b) JREG ($\alpha = 1.0$)

(c) JREG ($\alpha = 3.0$)

Figure 11: Checkpoint-wise hidden state displacement on 1B model

(a) Baseline

(b) JREG ($\alpha = 1.0$)
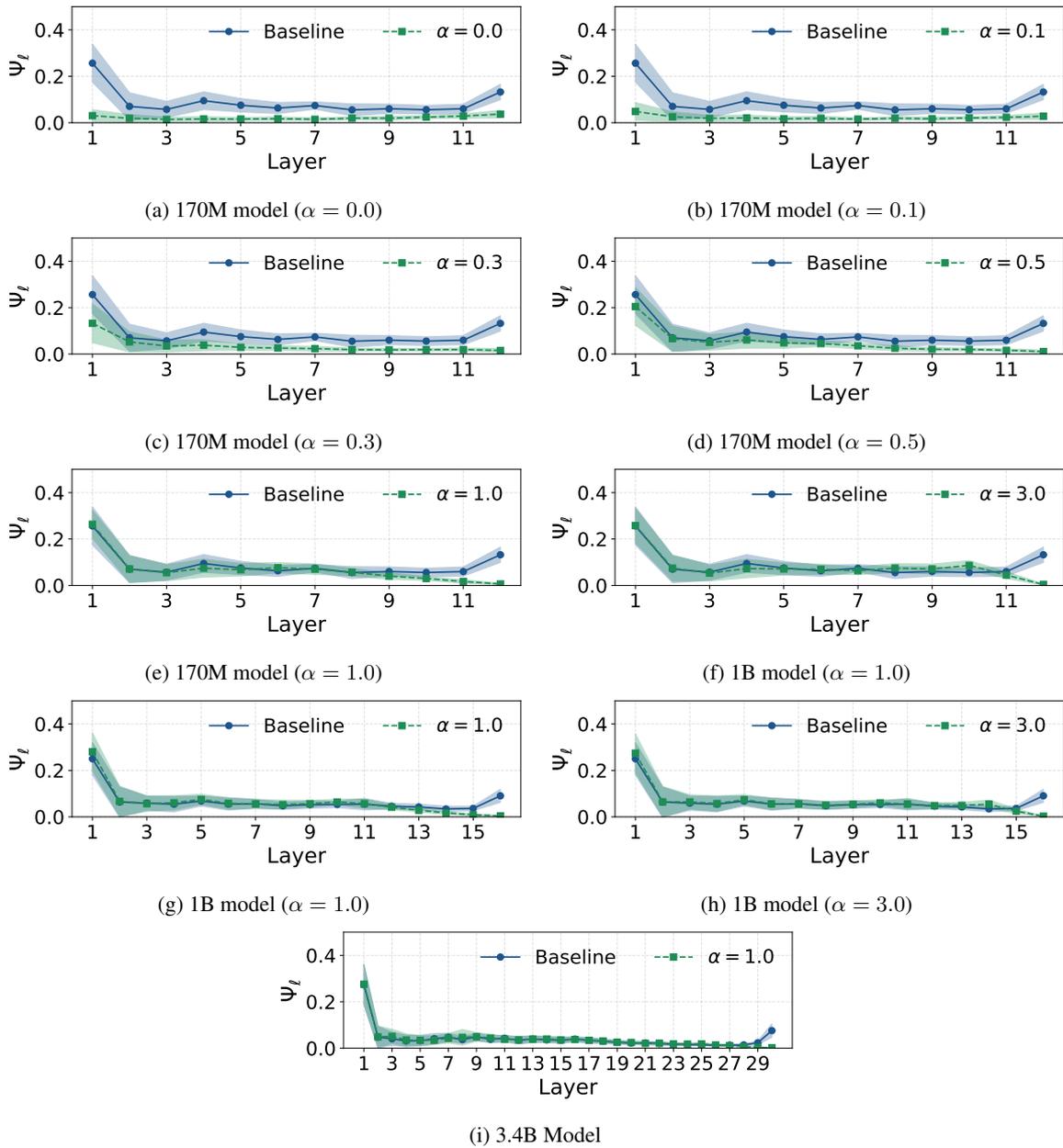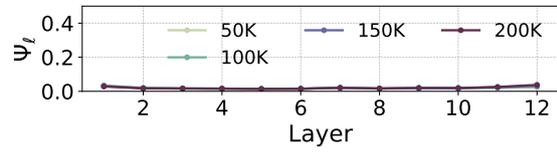
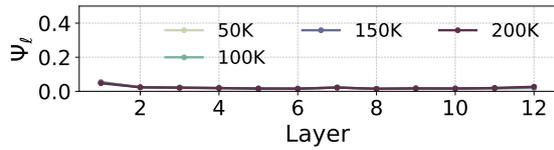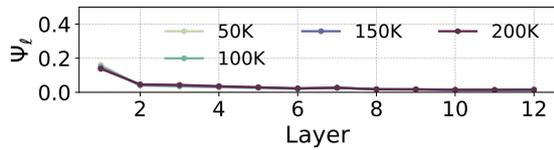Figure 12: Checkpoint-wise hidden state displacement on 3.4B model

(a) 170M model ($\alpha = 0.0$)

(b) 170M model ($\alpha = 0.1$)

(c) 170M model ($\alpha = 0.3$)

(d) 170M model ($\alpha = 0.5$)

(e) 170M model ($\alpha = 1.0$)

(f) 1B model ($\alpha = 1.0$)

(g) 1B model ($\alpha = 1.0$)

(h) 1B model ($\alpha = 3.0$)

(i) 3.4B Model

Figure 13: Layer-wise displacement Ψ_ℓ. Blue line and green line represent the results for baseline and JREG, respectively.
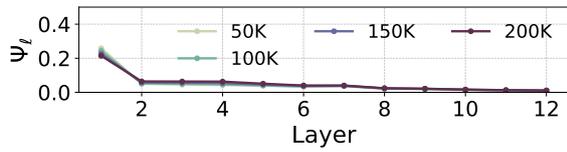
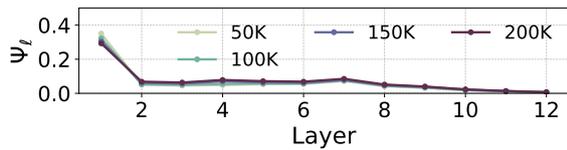(a) JREG ($\alpha = 0.0$)
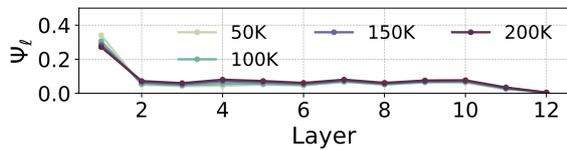

(b) JREG ($\alpha = 0.1$)

(c) JREG ($\alpha = 0.3$)

(d) JREG ($\alpha = 0.5$)

(e) JREG ($\alpha = 1.0$)

(f) JREG ($\alpha = 3.0$)

Figure 14: 170M checkpoint-wise hidden state displacement pretrained on Weborganizer.
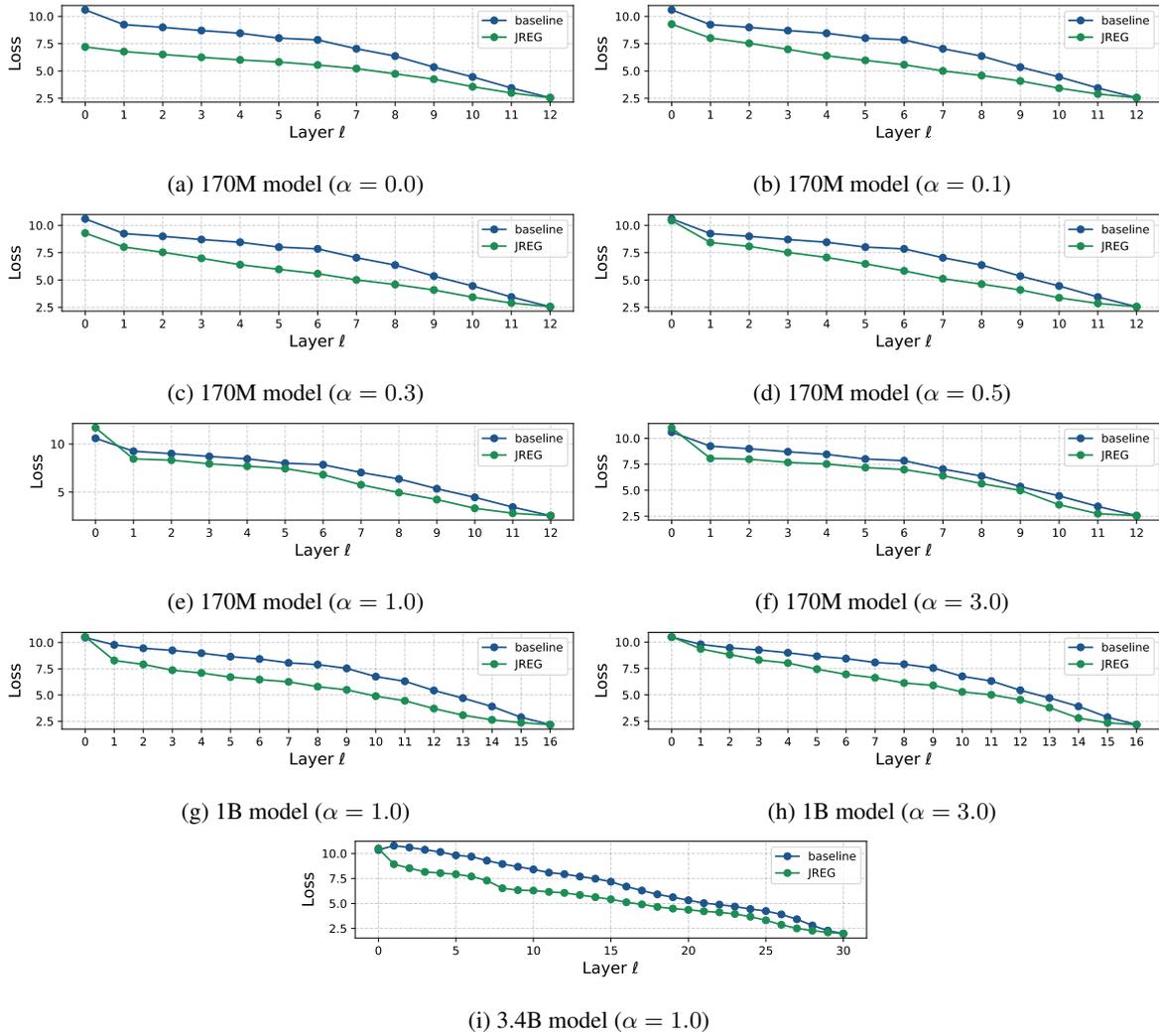
(a) 170M model ($\alpha = 0.0$)

(b) 170M model ($\alpha = 0.1$)

(c) 170M model ($\alpha = 0.3$)

(d) 170M model ($\alpha = 0.5$)

(e) 170M model ($\alpha = 1.0$)

(f) 170M model ($\alpha = 3.0$)

(g) 1B model ($\alpha = 1.0$)

(h) 1B model ($\alpha = 3.0$)

(i) 3.4B model ($\alpha = 1.0$)

Figure 15: Validation loss at each layer when inference is stopped at h_ℓ. Lower values indicate better performance.