UNLOCKING THE POWER OF LAYER BY LAYER TRAINING FOR LLM

Anonymous authors
Paper under double-blind review

ABSTRACT

Layer-wise (LW) training of deep neural networks has long been associated with memory and parallelism advantages, yet it suffers from information degradation and poor convergence in deep architectures. Recent work attributes these issues to the loss of input information and the lack of layer-role differentiation, as measured by the Hilbert-Schmidt Independence Criterion (HSIC).

In this paper, we present a novel algorithm that enables full end-to-end training of Large Language Models (LLMs) using a LW approach, while minimizing performance degradation. Through a comprehensive set of new experimental results, we demonstrate that although prior work has shown LW training to be effective in shallow architectures such as ResNet, its direct application to GPT-style LLMs leads to significant information loss and severely impaired convergence. Our fundamental contribution lies in the discovery that strategically reintroducing the final layers during LW training not only mitigates the convergence degradation typically observed in GPT-style LLMs but can in fact surpass the performance of conventional end-to-end training. This breakthrough unlocks a new paradigm for scalable optimization of deep transformer architectures, offering a powerful framework for training large models with improved efficiency, stability, and resource utilization.

We introduce Segmented Propagation (SegProp), a novel training paradigm that seamlessly integrates the computational efficiency of LW optimization with the representational power of global supervision. SegProp also introduces early-exit opportunities during training, enabling model compression. Quantitative results demonstrate substantial improvements in convergence compared to standard LW training. Finally, we position SegProp within the broader literature on information bottleneck theory, LW training, and early-exit strategies, and discuss its implications for scalable, energy efficient AI training and inference.

1 INTRODUCTION

Training large language models (LLMs) has become synonymous with leveraging parallelism strategies such as data, model, tensor, and pipeline parallelism to overcome the memory and computational constraints posed by billions of parameters. For today's state of the art LLMs, parallelism is not optional but essential: these models cannot fit into a single GPU's memory, even with advanced memory optimization techniques (HuggingFace, 2023). This requirement has led to the development of sophisticated distributed training frameworks and best practices for balancing efficiency and scalability (Wang et al., 2025).

End-to-end (E2E) training via back-propagation has driven the success of deep learning, but it faces challenges related to memory consumption, limited parallelism, and biological plausibility (Baldi et al., 2017). Layer-wise training, which optimizes local losses for each layer, offers hardware and memory advantages, but it fails to match E2E performance especially in deep models (Sakamoto & Sato, 2024).

The prevailing explanation for this performance gap is information loss: greedy local optimization discards input information needed for classification, leading to poor convergence and limited accuracy improvements as depth increases (Sakamoto & Sato, 2024). Recent studies have quantified this degradation using the Hilbert–Schmidt Independence Criterion (HSIC), which serves as

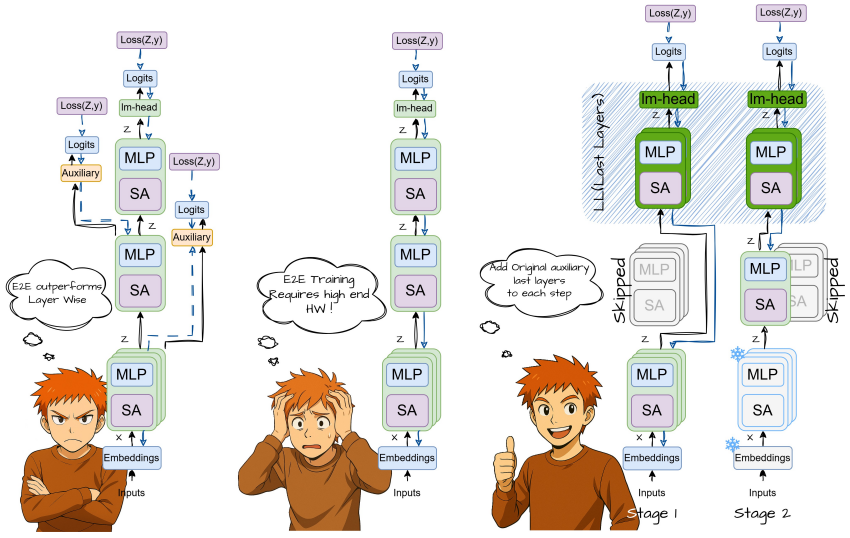


Figure 1: From left to right: Layer Wise (LW), End to End (E2E), Segmented Propagation (SegProp).

a proxy for mutual information, and have connected these dynamics to the information bottleneck principle (Tishby et al., 2000; Tishby & Zaslavsky, 2015).

In this paper, we target to restore information flow and achieve competitive accuracy while retaining the efficiency benefits of segmented training. We present *Segmented Propagation (SegProp)*, a novel training paradigm that combines the efficiency of LW optimization with the benefits of global supervision. SegProp also introduces early-exit opportunities during training, enabling model compression and dynamic depth selection. Building on the principles of the information bottleneck (Tishby et al., 2000; Tishby & Zaslavsky, 2015), SegProp restores information flow and achieves competitive accuracy while substantially reducing computational complexity.

2 Related Work

2.1 Layer-wise Training and Information Bottleneck

Layer-wise training was originally proposed to address the credit assignment problem and provide better initialization for deep networks (Bengio et al., 2006; Hinton et al., 2006). More recently, studies have shown that layer-wise training suffers from information degradation, as quantified by the Hilbert–Schmidt Independence Criterion (HSIC), which serves as a proxy for mutual information. This degradation leads to poor generalization and limited accuracy gains as depth increases (Sakamoto & Sato, 2024; Wang et al., 2021). The information bottleneck principle (Tishby et al., 2000; Tishby & Zaslavsky, 2015) has been used to analyze these dynamics, showing that end-to-end (E2E) training achieves compression in intermediate layers while preserving task-relevant information in the final layer.

However, a fundamental challenge remains: standard layer-wise training discards input information needed for classification, resulting in poor convergence and limited accuracy improvements as model depth increases. The lack of global supervision means that intermediate representations may not be sufficiently informative for the final prediction task.

In this work, we address these limitations by rethinking the role of final layers (see Figure 1) in segmented optimization. Specifically, we propose Segmented Propagation (SegProp), which reintroduces the final layers during LW training, restoring information flow and achieving competitive accuracy (see Figure 2) while retaining the efficiency benefits of segmented optimization.

2.2 Biologically Plausible and Modular Training

Alternatives to backpropagation, such as Hebbian learning (Hebb, 1949), reservoir computing (Bianchi et al., 2020), and signal propagation (Kohan et al., 2022), aim to reduce memory usage and computational cost by localizing learning. Modular and block-wise training strategies (Belilovsky et al., 2018; Gomez et al., 2022) seek to balance parallelism and accuracy, but they often rely on backpropagation within blocks. The Forward-Forward algorithm (Hinton, 2022) enforces distinct roles for each layer, but it still suffers from information loss as depth increases.

2.3 Activation Checkpointing and Memory-Efficient Training

To address the high memory footprint of E2E backpropagation, checkpointing strategies (Chen et al., 2016; He & Yu, 2023; Purandare et al., 2023; Korthikanti et al., 2022) have been widely explored. These methods reduce memory usage by selectively storing intermediate activations and recomputing others during the backward pass, trading additional computation for lower memory requirements. The work by Sakamoto & Sato (2024) highlights checkpointing as a key technique for enabling deeper models under constrained resources, particularly when comparing E2E and LW training paradigms.

Activation checkpointing (AC) (Chen et al., 2016; He & Yu, 2023; Purandare et al., 2023) and selective activation checkpointing (SAC) (Korthikanti et al., 2022) are standard techniques for reducing peak GPU memory consumption. SAC further optimizes this process by applying checkpointing only to critical layers, balancing computational overhead with memory efficiency. These approaches remain essential even when multi-dimensional parallelism is employed. While checkpointing strategies effectively address memory constraints, they introduce a trade-off: additional computation and potential bandwidth bottlenecks, as recomputation can increase data movement and slow down throughput, especially in distributed or multi-GPU settings.

We introduce a new mechanism, *snapshot checkpointing* (SnapCheck), described in detail later in the paper, which further improves efficiency by caching intermediate representations for reuse across training iterations.

2.4 Early Exit and Compression

Early-exit strategies introduce intermediate classifiers that allow models to terminate inference once a confidence threshold is met, reducing computation and latency (Marquez et al., 2018). These methods have also been explored for efficiency gains in model design and deployment, including scenarios that enable pruned or compressed architectures (Blalock et al., 2020).

Most early-exit and compression techniques focus on inference, not training, and typically require architectural modifications or additional parameters. They do not provide input-adaptive behavior during training.

In contrast, SegProp incorporates early-exit mechanisms directly into the training process, enabling static model compression and resource-aware optimization. By leveraging the original LM head and optionally final decoder layers for all segments, SegProp allows training to terminate once convergence criteria are met for a given segment effectively pruning deeper layers before they are trained and unlocking new opportunities for efficient, adaptive model design.

3 Segmented Propagation (SegProp)

3.1 Problem Setting

As previously noted, E2E training of large models demands significant GPU memory, as it must store all model components - weights, activations, optimizer states, gradients, and more - resulting in high peak memory usage and extended runtimes.

To mitigate the computational burden, LW training updates one layer or sub-block at a time rather than performing full E2E backpropagation, thereby significantly lowering memory and compute requirements (Bengio et al., 2006; Hinton et al., 2006). However, this approach often suffers from

information loss and suboptimal performance, as the absence of global supervision can lead to intermediate representations that are insufficiently informative for the final prediction task (Sakamoto & Sato, 2024). While prior work on convolutional architectures such as ResNet18, ResNet50, and VGG11 has shown that LW training can achieve acceptable accuracy with only minor degradation (Sakamoto & Sato, 2024), our experiments on a Transformer-based architecture reveal a much more severe performance drop under the MMLU and HumanEval+ (HE+) (Liu et al., 2023) benchmark.

We propose Segmented Propagation (SegProp) - a method grounded in LW training but enhanced by reintroducing the final layers during each training segment. This integration helps recover information loss and achieves competitive accuracy, while preserving the efficiency benefits of layer-wise training.

Focusing on Transformer based architectures, SegProp leverages the inherent dimensional consistency across decoder components, such as self-attention (SA), MLP layers, and the LM head input; thereby eliminating the need for auxiliary networks typically introduced to resolve dimensional mismatches between internal layers and the objective function. For models that lack this property, an auxiliary network may still be required; however, the core training principles of SegProp remain unchanged.

SegProp employs a two-stage training strategy designed to balance computational efficiency with strong supervision:

1. **Joint Training of Base and Final Layers:** A selected prefix of the model (the base layers) is trained jointly with a set of final layers to establish strong end-task supervision early in training. The non selected intermediate layers are skipped during this stage to reduce compute.
2. **Iterative Layer-Wise Training of Intermediate Layers:** Each intermediate layer is trained individually alongside the final layers, which provide consistent supervision. Previously trained layers are kept frozen, and only the current updated target layer is committed for the next iteration.

We formalize the system setup and introduce the notation used throughout this paper before diving into the finer details of each step. We consider a dataset $\{(x_i, y_i)\}_{i=1}^{m}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Specifically, $\mathcal{X} = \mathbb{N}_+^{v \times s}$ and $\mathcal{Y} \subset \mathbb{N}_+^{v}$, where s denotes the sequence length and v the vocabulary size.

Let

$$f(x) = f_{\text{LM}} \circ f_{n-1} \circ f_{n-2} \circ \cdots \circ f_0 \circ f_{\text{embed}}(x) \tag{3.1}$$

denote an LLM with Transformer architecture (see Eq. (3.1)) and n decoder layers, where $f_i(x) = f_{\text{MLP}} \circ f_{\text{SA}}(x)$. Define: $f_{\text{embed}} : \mathbb{R}^v \to \mathbb{R}^h$ as the embedding layer, $f_{\text{SA}} : \mathbb{R}^h \to \mathbb{R}^h$ as the SA layer, $f_{\text{MLP}} : \mathbb{R}^h \to \mathbb{R}^h$ as the Multi-Layer Perceptron (MLP) layer, $f_i : \mathbb{R}^h \to \mathbb{R}^h$ as the i-th decoder layer, $f_{\text{LM}} : \mathbb{R}^h \to \mathbb{R}^v$ as the LM head, where h denotes the hidden size and $i \in \{0, \ldots, n-1\}$.

For compactness, define the half-open composition operator:

$$F_{a:b}(x) := f_{b-1} \circ f_{b-2} \circ \cdots \circ f_a(x) \tag{3.2}$$

with the convention $F_{a:a}(x) = x$ (i.e., the identity map).

Let

$$f^{[0:p]}(x) := F_{0:p+1}(f_{\text{embed}}(x)), \quad p \in \{0, \ldots, n-2\} \tag{3.3}$$

represent the base model up to depth p (Eq. (3.3)).

Define the last-layers module as

$$\mathcal{LL}^{(r)}(x) := f_{\text{LM}} \circ F_{n-r:n}(x), \quad r \in \{0, \ldots, n-p-2\} \tag{3.4}$$

where r is the number of decoder layers included in $\mathcal{LL}^{(r)}$ (not counting the LM head). This subset always includes the LM head (f_{LM}) and may optionally include one or more of the final decoder layers (e.g., f_{n-2}, f_{n-1}). For $r = 0$, $\mathcal{LL}^{(0)}(x) = f_{\text{LM}}(x)$ (LM head only).

Intermediate layers are those not included in $f^{[0:p]}$ or $\mathcal{LL}^{(r)}$. Finally, let `StopGrad` denote an operator that prevents gradient propagation (e.g., `Tensor.detach()` in PyTorch), ensuring that gradients do not flow through preceding layers (Sakamoto & Sato, 2024).

For clarity, we present the algorithm at the granularity of individual decoder layers. While SegProp can operate on segments comprising multiple layers, in this work we assume a simplified setting where each segment consists of a single layer trained sequentially.

3.2 Stage 1: Joint Fine-Tuning of Base and Last Layers

We begin by selecting a base model depth p (see Eq. (3.3)) from the network f (see Eq. (3.1)). In Stage 1, we train the base jointly with $\mathcal{LL}$, while skipping intermediate layers.

Given a minibatch $\mathcal{B} = \{(x^{(b)}, y^{(b)})\}_{b=1}^{|\mathcal{B}|}$, the forward pass is Eq. (3.5):

$$z^{(b)} = f^{[0:p]}(x^{(b)}), \quad \hat{y}^{(b)} = \mathcal{LL}\big(z^{(b)}\big). \tag{3.5}$$

and the stage loss is computed as:

$$\mathcal{L}_1 = \frac{1}{|\mathcal{B}|} \sum_{b=1}^{|\mathcal{B}|} \ell\big(\hat{y}^{(b)}, y^{(b)}\big), \tag{3.6}$$

where ℓ is the task-specific objective (e.g., cross-entropy). Importantly, backpropagation is restricted to $\{f^{[0:p]}, \mathcal{LL}\}$, ensuring that early representations align with the final prediction objective without constraining intermediate layers. Upon completing Stage 1, the base prefix is committed and remains frozen for subsequent stages.

3.3 Stage 2: Iterative Training of Middle Layers with $\mathcal{LL}$

In Stage 2, we train the intermediate layers which were not part of the base model nor the $\mathcal{LL}$ layers through segmented training. Each layer $f_{\hat{p}}$ is trained individually while reintroducing $\mathcal{LL}$ to maintain global context. The committed base prefix $\{f^{[0:p]}\}$ and the trained layers from previous iterations remain frozen. We denote that by setting $\hat{p} = p + 1$ before each iteration.

For a given layer $\hat{p}$ and minibatch $\mathcal{B}$, the forward pass is (see Eq. (3.7)):

$$z^{(b)} = f^{[0:p]}(x^{(b)}), \quad \hat{y}^{(b)} = \mathcal{LL}(f_{\hat{p}}(\text{StopGrad}\big(z^{(b)}\big))\big). \tag{3.7}$$

and the local loss is:

$$\mathcal{L}_2 = \frac{1}{|\mathcal{B}|} \sum_{b=1}^{|\mathcal{B}|} \ell\big(\hat{y}^{(b)}, y^{(b)}\big), \quad p \leftarrow \hat{p}. \tag{3.8}$$

Here, the forward pass flows through the frozen layers, followed by the current training layer $f_{\hat{p}}$ and $\mathcal{LL}$. Gradients are propagated only through $\{f_{\hat{p}}, \mathcal{LL}\}$ and updates to $\mathcal{LL} \setminus \{f_{LM}\}$ are temporary, where only $f_{\hat{p}}, f_{LM}$ are committed after convergence.

3.4 Snapshot Checkpointing (SnapCheck)

To mitigate redundant computation during Stage 2, SegProp introduces *Snapshot Checkpointing* (SnapCheck), a mechanism that caches intermediate representations for reuse across iterations. Specifically, after computing the committed prefix output (Eq. (3.9)):

$$z^{(b)} = f^{[0:p]}(x^{(b)}), \quad \forall b \in \{1, \ldots, |\mathcal{B}|\}, \tag{3.9}$$

SnapCheck stores $z^{(b)}$ activations (Eq. (3.9)) in a memory-efficient structure indexed by both the prefix depth p and the minibatch identifier. When fine-tuning a subsequent layer $f_{\hat{p}}$, instead of recomputing $f^{[0:p]}(x^{(b)})$ for every forward pass, the algorithm retrieves the cached $z^{(b)}$ (Eq. (3.10)):

$$z^{(b)} \leftarrow \mathcal{S}[p, \text{batch\_id}], \tag{3.10}$$

where $\mathcal{S}$ denotes the snapshot buffer. If the snapshot is unavailable (e.g., first access), the forward computation is performed and the result can be stored for future reuse.

This approach offers a key advantage: it eliminates repeated evaluation of the committed prefix, significantly reducing computational overhead for deep architectures and improving overall training efficiency. SnapCheck is particularly effective when the committed prefix is large (e.g., $p \gg 0$), as the cost of recomputation grows linearly with p. By leveraging SnapCheck, SegProp improves training throughput and reduces energy consumption without compromising convergence.

3.5 Comparison with Existing Early Exit and Pruning Methods

Early-exit strategies aim to reduce inference costs by introducing auxiliary classifiers at intermediate layers and using confidence-based halting criteria, such as maximum softmax probability or entropy, to terminate computation once a threshold is met (Xin et al., 2020; Tang et al., 2023). While effective for adaptive inference, these methods add overhead from auxiliary heads and repeated softmax operations, prompting optimizations like dynamic vocabulary pruning (Vincenti et al., 2024) and adaptive-depth models (e.g., Universal Transformers) that use halting probabilities (Dehghani et al., 2019). However, these remain inference-centric and require architectural changes. In contrast, pruning-based approaches focus on static compression by permanently reducing model size through techniques such as structural pruning (LLM-Pruner) (Ma et al., 2023), one-shot weight pruning (SparseGPT) (Frantar & Alistarh, 2023), embedding slicing (SliceGPT) (Ashkboos et al., 2024), and block or GLU-aware pruning (Lagunas et al., 2021; Girija et al., 2025). Recent frameworks, including DASH, LLM-BIP, LaCo, ShortGPT, and FinerCut (Liu et al., 2021; Wu, 2024; Liu et al., 2025; Miao et al., 2024; Zhang et al., 2024), extend these ideas with advanced layer dropping and fine-grained pruning, but they lack input-adaptive behavior.

SegProp differs fundamentally from these paradigms. Our early-exit mechanism operates *during training*, not inference, and is integrated into the segmented optimization process. Rather than relying on auxiliary classifiers, SegProp leverages the original LM head with optionally one or more of the final decoder layers for all segments, enabled by the dimensional consistency of Transformer blocks. This design allows SegProp to terminate training once convergence criteria are met for a given segment, effectively pruning deeper layers before they are trained. Consequently, SegProp achieves dynamic depth selection and computational savings without introducing additional parameters or inference-time complexity.

4 Results and Analysis

4.1 Restoring Convergence with Final Layers

Figure 2 illustrates the impact of reintroducing the last 2 decoder layers during SegProp training on MMLU and HE+ performance. Conventional LW training demonstrates slow convergence and limited final accuracy, consistent with previously reported effects of information degradation (Sakamoto & Sato, 2024). In contrast, integrating the final layers significantly accelerates convergence and improves accuracy (See Figure 2d), enabling SegProp to surpass traditional end-to-end training approaches across language understanding (MMLU) and code generation (HE+) benchmarks.

4.2 Empirical Evaluation and Performance Metrics

Appendix Table 1, 2 presents the final MMLU and HE+ scores obtained for each middle layer following fine-tuning with SegProp at the end of its respective training step. The results indicate that the choice of base model defined by the number of layers included in Stage 1 of the SegProp algorithm has a significant impact on performance. Notably, when comparing the Base 1 and Base 18 configurations, it becomes evident that the model retains essential task-relevant knowledge for the MMLU benchmark within the first 18 layers. At this depth, the model surpasses random chance (25%) and exhibits clear signs of restored convergence.

Algorithm 1 Segmented Propagation (Stochastic) Gradient Descent (SegProp-SGD)

Inputs: Network $f(x) = f_{\text{LM}} \circ f_{n-1} \circ f_{n-1} \circ \cdots \circ f_0 \circ f_{\text{embed}}(x)$ with n layers; base depth p; loss ℓ; optimizer $\mathcal{O}$; exit criterion for Stage 1; exit criterion for layer $\hat{p}$; global exit criterion;

Notation: $F_{a:b}(x) := f_{b-1} \circ f_{b-2} \circ \cdots \circ f_a(x)$; $f^{[0:p]}(x) := F_{0:p+1}(f_{\text{embed}}(x))$, $p \in \{0, \ldots, n-2\}$; $\hat{p} = p + 1$; $\mathcal{LL}^{(r)} := f_{\text{LM}} \circ F_{n-r:n}(x)$, $r \in \{0, \ldots, n-p-2\}$;

Partition: Decompose f into non-overlapping segments (SA/MLP/Decoder). Always include $\mathcal{LL}$ during training to mitigate information loss.

Stage 1: Joint fine-tuning of base layers with $\mathcal{LL}$

while exit criterion for Stage 1 not met **do**
  for each minibatch (x, y) **do**
    $z \leftarrow f^{[0:p]}(x)$
    $\hat{y} \leftarrow \mathcal{LL}^{(r)}(z)$
    $\mathcal{L}_1 \leftarrow \ell(\hat{y}, y)$
    Backward/Update: backprop through $\{f^{[0:p]}, \mathcal{LL}^{(r)}\}$ and update.
Freeze $f^{[0:p]}$; **commit** base prefix $f^{[0:p]}, f_{LM}$

Stage 2: Iterative layer-by-layer training with $\mathcal{LL}$ (cumulative base)

for $\hat{p} \leftarrow p{+}1$ **to** $n{-}2$ **do**
  while exit criterion for layer $\hat{p}$ not met $\forall f_i \in \mathcal{LL} \backslash \{f_{LM}\}$ **do Reset** to their baseline weights.
    for each minibatch (x, y) **do**
      if snapshot $\mathcal{S}[p, \text{batch}]$ exists **then**
        $z \leftarrow \mathcal{S}[p, \text{batch}]$
      else
        $z \leftarrow \text{StopGrad}\big(f^{[0:p]}(x)\big)$
        Optionally store $\mathcal{S}[p, \text{batch}] \leftarrow z$
      $\hat{y} \leftarrow \mathcal{LL}^{(r)}\big(f_{\hat{p}}(z)\big)$
      $\mathcal{L}_2 \leftarrow \ell(\hat{y}, y)$
      Backward/Update: backprop through $\{f_{\hat{p}}, \mathcal{LL}^{(r)}\}$; update $\{f_{\hat{p}}, \mathcal{LL}^{(r)}\}$
    Commit and freeze $f_{\hat{p}}, f_{LM}$; **extend base** $p \leftarrow \hat{p}$
    if global exit criterion satisfied **then break**

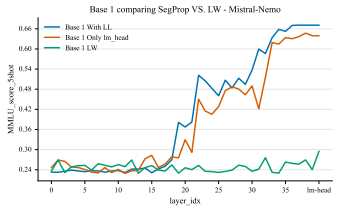
5 Discussion And Further Work

Our results provide direct evidence that *reintroducing the final (LL, see Eq.* (3.4)*) component* i.e., training with the model's last two layers present and participating in gradient flow **mitigates the information loss characteristic of purely LW training**. In at least one of our simulations, this approach **outperforms the conventional LW baseline and matches or exceeds E2E training** on MMLU (see Fig. 2c, 2d and Appendix Table 1). This observation aligns with the view that global targets (or "top-layer supervision") help preserve task-relevant information across depth, a role historically played by deep supervision and auxiliary heads (Belilovsky et al., 2018; Marquez et al., 2018).

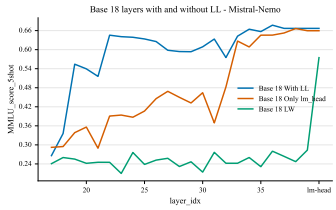
5.1 Comparison with Prior Work

A large body of work has highlighted why standard LW or locally supervised training often underperforms compared to E2E methods: local objectives can induce representations that are insufficiently informative for the final prediction task. This deficit can be revealed by dependence measures such as the Hilbert–Schmidt Independence Criterion (HSIC) along the depth of the network (Sakamoto & Sato, 2024). Recent analyses show that E2E training both propagates input information more effectively and induces differentiated roles across layers properties that degrade when layers are optimized greedily and in isolation (Sakamoto & Sato, 2024).

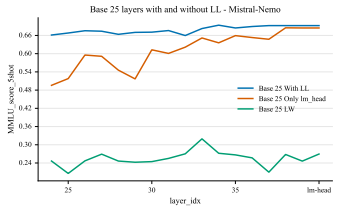
SegProp's design explicitly addresses this gap by *reintroducing the $\mathcal{LL}$ (see Eq.* (3.4)*) component during each segment's optimization*, thereby providing a global, task-aligned signal to otherwise



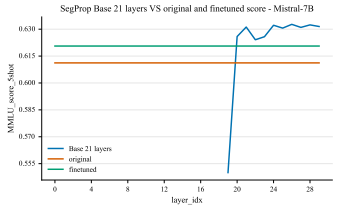
(a) Mistral Nemo with 1 layer on mmlu benchmark



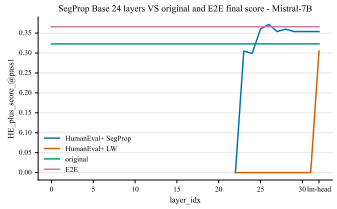
(b) Mistral Nemo with 18 layers (0–17) on mmlu benchmark



(c) Mistral Nemo with 25 layers (0–24) on mmlu benchmark



(d) Mistral 7B with 21 layers on mmlu benchmark



(e) Mistral 7B instruct with 24 layers on coding benchmark

Figure 2: SegProp convergence across different model prefixes.

locally optimized blocks. This mechanism is complementary to prior approaches that mitigate local-learning limitations via deep supervision or synthetic gradients (Belilovsky et al., 2018; Marquez et al., 2018). Unlike those methods, SegProp leverages the *actual* final layers as a universal target pathway, avoiding auxiliary classifiers or gradient predictors while retaining the parallelism and memory benefits of segmented training.

5.2 Information Bottleneck and Layer-role Differentiation

The Information Bottleneck (IB) principle frames supervised learning as finding minimal sufficient representations: compress X while preserving information about Y (Tishby et al., 2000). In deep networks, this translates to a trade-off across layers between capturing task-relevant information and discarding nuisances. Classical IB work formalizes this as an optimization over mutual information, while subsequent interpretations argue that deep networks traverse "information plane" trajectories reflecting fitting and compression phases (Tishby & Zaslavsky, 2015).

Critiques have clarified that *how* one measures information in deterministic networks matters; observed "compression phases" can depend on activation saturation and estimation details, cautioning against universal claims. Nevertheless, even these critiques concede that when data contain distinct task-relevant and irrelevant components, hidden representations tend to compress the latter while refining the former (Saxe et al., 2018). Our findings are consistent with this view: by restoring the $\mathcal{LL}$ (see Eq. (3.4)) component during segmented optimization, SegProp reinstates cooperative interactions across layers, encouraging the emergence of layer-role differentiation and task-aligned compression patterns without requiring full E2E backpropagation for all parameters at once.

From a dependence-measure perspective, HSIC has emerged as a practical, kernel-based proxy for tracking information propagation and layer-role specialization. Prior analyses explicitly compared E2E vs. layer-wise training using HSIC-normalized planes, showing the superiority of E2E at sustaining dependence on inputs while shaping later-layer dynamics consistent with IB (Sakamoto & Sato, 2024). SegProp's empirical behavior mirrors these dynamics: adding $\mathcal{LL}$ (see Eq. (3.4)) to each segment supplies a global constraint that counteracts the independence drift of middle layers.

5.3 Hardware and Energy Efficiency

A key practical advantage of SegProp lies in its ability to translate the theoretical benefits of localized training into *system-level efficiency*. This is achieved through three core design choices: (i) training one segment at a time, (ii) reusing the $\mathcal{LL}$ (see Eq. (3.4)) component as a universal head across all

segments, and (iii) leveraging the dimensional consistency among SA, MLP, and Decoder blocks in transformer architectures. These features reduce peak activation memory and support flexible scheduling on constrained accelerators.

SegProp integrates with activation checkpointing (AC) and selective activation recomputation (SAC), minimizing stored/recomputed activations since only one segment backpropagates at a time. AC/SAC manage residual hotspots with modest overhead (Chen et al., 2016; Korthikanti et al., 2022; Purandare et al., 2023).

SnapCheck eliminates redundant computation, accelerates training, and reduces energy use.

SegProp's *training-time early exits* complement *inference-time* halting. Intermediate layers calibrated against the $\mathcal{LL}$ (see Eq. (3.4)) component enable confidence-based exits without auxiliary heads, aligning with deeply supervised and early-exit Transformer methods (Marquez et al., 2018; Miao et al., 2024).

Segmentation enables *structured compression*: segments meeting targets early allow pruning or skipping. Frameworks like SparseGPT, LLM-Pruner, LaCo, FinerCut, and SliceGPT consolidate compute/memory savings while preserving accuracy (Frantar & Alistarh, 2023; Ma et al., 2023; Ashkboos et al., 2024; Liu et al., 2025; Zhang et al., 2024).

5.4 Future Directions

While SegProp demonstrates strong performance when applied as a fine-tuning strategy, an important avenue for future research is training from scratch under the SegProp regime. This approach could enable more granular control over representational capacity across segments, allowing practitioners to selectively allocate or compress knowledge within specific blocks. Such flexibility would not only facilitate structured model compression but also make task-specific compression more practical, enabling the deployment of specialized sub-networks without retraining the entire model. These capabilities would extend SegProp's utility beyond efficiency gains, positioning it as a framework for adaptive and domain-aware model design.

6 Conclusion

We presented **Segmented Propagation (SegProp)**, a two-stage algorithm that restores global task information during otherwise local, memory-efficient training. By *reintroducing the final ($\mathcal{LL}$, (see Eq. (3.4))) layers* in every segment's optimization, SegProp **overcomes the information loss endemic to purely LW training** and **matches or exceeds** E2E performance on both MMLU and HE+ benchmarks while retaining the scalability advantages of segmentation. Our analysis connects these effects to the Information Bottleneck perspective (Tishby et al., 2000; Tishby & Zaslavsky, 2015) and to HSIC-based evidence that E2E success hinges on effective information propagation and layer-role differentiation (Sakamoto & Sato, 2024); SegProp reproduces these beneficial dynamics without requiring full-network backprop at all times.

Practically, SegProp's training-time structure translates into *hardware and energy benefits*: lower peak memory via segmented backprop that plays well with AC/SAC (Chen et al., 2016; Korthikanti et al., 2022), readiness for early-exit policies at inference (Marquez et al., 2018; Miao et al., 2024), and compatibility with modern LLM pruning pipelines for permanent compute reductions (Frantar & Alistarh, 2023; Ma et al., 2023; Ashkboos et al., 2024; Liu et al., 2025; Zhang et al., 2024). Together, these features make SegProp a *scalable, resource-aware* alternative to monolithic E2E training well suited to future systems where training efficiency, adaptability, and deployment cost are as critical as final accuracy.

References

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*, 2024. doi: 10.48550/arXiv.2401.15024. URL https://arxiv.org/abs/2401.15024. Accepted at ICLR 2024.

Pierre Baldi, Peter Sadowski, and Zhiqin Lu. Learning in the machine: the symmetries of the deep learning channel. *arXiv preprint arXiv:1712.08608*, 2017. URL https://arxiv.org/abs/1712.08608.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. *arXiv preprint arXiv:1812.11446*, 2018. doi: 10.48550/arXiv.1812.11446. URL https://arxiv.org/abs/1812.11446.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. 2006. URL https://proceedings.neurips.cc/paper_files/paper/2006/file/5da713a690c067105aeb2fae32403405-Paper.pdf. Advances in Neural Information Processing Systems 19 (NeurIPS 2006).

Filippo Maria Bianchi, Simone Scardapane, Sigurd Løkse, and Robert Jenssen. Reservoir computing approaches for representation and classification of multivariate time series. *arXiv preprint arXiv:1803.07870*, 2020. doi: 10.48550/arXiv.1803.07870. URL https://arxiv.org/abs/1803.07870.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020. doi: 10.48550/arXiv.2003.03033. URL https://arxiv.org/abs/2003.03033.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint*, 2016. URL https://arxiv.org/abs/1604.06174.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. 2019. URL https://arxiv.org/abs/1807.03819.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023. doi: 10.48550/arXiv.2301.00774. URL https://arxiv.org/abs/2301.00774.

Sanjay Surendranath Girija, Shashank Kapoor, Lakshit Arora, Dipen Pradhan, Aman Raj, and Ankit Shetgaonkar. Optimizing llms for resource-constrained environments: A survey of model compression techniques. *arXiv preprint arXiv:2505.02309*, 2025. doi: 10.48550/arXiv.2505.02309. URL https://arxiv.org/abs/2505.02309.

Aidan N. Gomez, Oscar Key, Kuba Perlin, Stephen Gou, Nicholas Frosst, Jeff Dean, and Yarin Gal. Interlocking backpropagation: Improving depthwise model-parallelism. *Journal of Machine Learning Research*, 23(1):7714–7741, 2022.

Horace He and Shangdi Yu. Transcending runtime-memory tradeoffs in checkpointing by being fusion aware. In *Proceedings of the 6th MLSys Conference*, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/8a27bb69950c0b46cdb36d10e5514cc8-Paper-mlsys2023.pdf.

Donald O. Hebb. *The Organization of Behavior*. Wiley, 1949.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022. doi: 10.48550/arXiv.2212.13345. URL https://arxiv.org/abs/2212.13345.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. doi: 10.1162/neco.2006.18.7.1527. URL https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf.

HuggingFace. The large language model training handbook. https://github.com/huggingface/llm_training_handbook, 2023. Accessed: 2025-09-14.

Adam Kohan, Edward A. Rietman, and Hava T. Siegelmann. Signal propagation: A framework for learning and inference in a forward pass. *arXiv preprint arXiv:2204.01723*, 2022. doi: 10.48550/arXiv.2204.01723. URL https://arxiv.org/abs/2204.01723.

Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *arXiv preprint arXiv:2205.05198*, 2022. doi: 10.48550/arXiv.2205.05198. URL https://arxiv.org/abs/2205.05198.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M. Rush. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021. doi: 10.48550/arXiv.2109.04838. URL https://arxiv.org/abs/2109.04838. Presented at EMNLP 2021.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. *arXiv preprint arXiv:2305.01210*, 2023. doi: https://doi.org/10.48550/arXiv.2305.01210. URL https://arxiv.org/abs/2305.01210.

Juntao Liu, Liqiang Niu, Wenchao Chen, Jie Zhou, and Fandong Meng. Laco: Efficient layer-wise compression of visual tokens for multimodal large language models. *arXiv preprint arXiv:2507.02279*, 2025. doi: 10.48550/arXiv.2507.02279. URL https://arxiv.org/abs/2507.02279.

Liu Liu, Zheng Qu, Zhaodong Chen, Yufei Ding, and Yuan Xie. Transformer acceleration with dynamic sparse attention. *arXiv preprint arXiv:2110.11299*, 2021. doi: 10.48550/arXiv.2110.11299. URL https://arxiv.org/abs/2110.11299.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023. doi: 10.48550/arXiv.2305.11627. URL https://arxiv.org/abs/2305.11627. Accepted at NeurIPS 2023.

Enrique S. Marquez, Jonathon S. Hare, and Mahesan Niranjan. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5475–5485, 2018. doi: 10.1109/TNNLS.2018.2791403. URL https://ieeexplore.ieee.org/document/8307262.

Ruijie Miao, Yihan Yan, Xinshuo Yao, and Tong Yang. An efficient inference framework for early-exit large language models. *arXiv preprint arXiv:2407.20272*, 2024. doi: 10.48550/arXiv.2407.20272. URL https://arxiv.org/abs/2407.20272.

Pradnya Purandare et al. Universal checkpointing: Efficient and flexible checkpointing for large models. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2406.18820.

Keitaro Sakamoto and Issei Sato. End-to-end training induces information bottleneck through layer-role differentiation: A comparative analysis with layer-wise training. *arXiv preprint arXiv:2402.09050*, 2024. doi: 10.48550/arXiv.2402.09050. URL https://arxiv.org/abs/2402.09050.

Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. 2018. URL https://openreview.net/forum?id=ry_WPG-A-.

Shengkun Tang, Yaqing Wang, Zhenglun Kong, Tianchi Zhang, Yao Li, Caiwen Ding, Yanzhi Wang, Yi Liang, and Dongkuan Xu. You need multiple exiting: Dynamic early exiting for accelerating unified vision language model. 2023. URL https://arxiv.org/abs/2211.11152.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *arXiv preprint arXiv:1503.02406*, 2015. doi: 10.48550/arXiv.1503.02406. URL https://arxiv.org/abs/1503.02406. Invited paper to IEEE Information Theory Workshop (ITW) 2015.

Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. *arXiv preprint arXiv:physics/0004057*, 2000. URL https://arxiv.org/abs/physics/0004057.

Jort Vincenti, Karim Abdel Sadek, Joan Velja, Matteo Nulli, and Metod Jazbec. Dynamic vocabulary pruning in early-exit llms. 2024. URL https://arxiv.org/abs/2410.18952.

Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. Revisiting locally supervised learning: An alternative to end-to-end training. 2021. URL https://arxiv.org/abs/2101.10832. International Conference on Learning Representations (ICLR).

Zheng Wang, Anna Cai, Xinfeng Xie, Zaifeng Pan, Yue Guan, Weiwei Chu, Jie Wang, Shikai Li, Jianyu Huang, Chris Cai, Yuchen Hao, and Yufei Ding. Wlb-llm: Workload-balanced 4d parallelism for large language model training. *arXiv preprint arXiv:2503.17924*, 2025. URL https://arxiv.org/abs/2503.17924.

Haihang Wu. Llm-bip: Structured pruning for large language models with block-wise forward importance propagation. *arXiv preprint arXiv:2412.06419*, 2024. doi: 10.48550/arXiv.2412.06419. URL https://arxiv.org/abs/2412.06419.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. 2020. URL https://arxiv.org/abs/2004.12993.

Yang Zhang, Yawei Li, Xinpeng Wang, Qianli Shen, Barbara Plank, Bernd Bischl, Mina Rezaei, and Kenji Kawaguchi. Finercut: Finer-grained interpretable layer pruning for large language models. *arXiv preprint arXiv:2405.18218*, 2024. doi: 10.48550/arXiv.2405.18218. URL https://arxiv.org/abs/2405.18218.

Table 1: Mistral-Nemo: MMLU Scores for different base models with & without LL (see Eq. (3.4))

Layers	Base 1 w/ LL	Base 1 w/o LL	Base 18 w/ LL	Base 18 w/o LL	Base 25 w/ LL	Base 25 w/o LL	Original	Finetuned
0	0.2329	0.2465	–	–	–	–		
1	0.2324	0.2689	–	–	–	–		
2	0.2355	0.2648	–	–	–	–		
3	0.2389	0.2478	–	–	–	–		
4	0.2363	0.2465	–	–	–	–		
5	0.2342	0.2408	–	–	–	–		
6	0.2369	0.2329	–	–	–	–		
7	0.2363	0.2305	–	–	–	–		
8	0.2327	0.2464	–	–	–	–		
9	0.2361	0.2314	–	–	–	–		
10	0.2377	0.2408	–	–	–	–		
11	0.2314	0.2289	–	–	–	–		
12	0.2419	0.2366	–	–	–	–		
13	0.2419	0.2381	–	–	–	–		
14	0.2451	0.2725	–	–	–	–		
15	0.2308	0.2829	–	–	–	–		
16	0.2424	0.2465	–	–	–	–		
17	0.2502	0.2567	0.2661	0.2922	–	–		
18	0.2697	0.2783	0.3358	0.2947	–	–		
19	0.3809	0.2752	0.5542	0.3385	–	–		
20	0.3673	0.3295	0.5395	0.3561	–	–	0.6770	0.6931
21	0.3821	0.2918	0.5158	0.2894	–	–		
22	0.5214	0.4502	0.6456	0.3908	–	–		
23	0.5050	0.4149	0.6409	0.3939	–	–		
24	0.4824	0.4054	0.6390	0.3871	0.6614	0.4957		
25	0.4605	0.4283	0.6337	0.4065	0.6680	0.5179		
26	0.5069	0.4757	0.6255	0.4457	0.6755	0.5955		
27	0.4836	0.4874	0.5981	0.4687	0.6739	0.5913		
28	0.5127	0.4804	0.5940	0.4498	0.6637	0.5457		
29	0.4945	0.4634	0.5935	0.432	0.6701	0.5169		
30	0.537	0.4899	0.6089	0.4643	0.6708	0.6127		
31	0.5998	0.4216	0.6334	0.3692	0.6761	0.6008		
32	0.5862	0.5162	0.5751	0.4825	0.6597	0.6214		
33	0.6336	0.6191	0.6425	0.6271	0.6827	0.6517		
34	0.6585	0.6146	0.6644	0.6085	**0.6938**	0.6357		
35	0.6525	0.6338	0.6569	0.6456	0.6846	0.6594		
36	0.6702	0.6305	**0.6771**	0.6458	0.6894	0.6531		
37	**0.6709**	0.6368	0.6670	0.6525	0.6924	0.6474		
38		**0.6472**		**0.6665**		**0.6851**		
39		0.6394		0.6596		0.6846		
lm								

A Quantitative Results for Mistral-Nemo on MMLU benchmark w/o LL

Table 2: Mistral-7B-Instruct-v0.3: HE+ Scores for different training paradigma

Layers	Base 24 w/ LL	Base 24 w/o LL	Original	Finetuned
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20			0.6770	0.6931
21				
22				
23				
24	0.305	0.1		
25	0.299	0.1		
26	0.360	0.1		
27	0.372	0.1		
28	0.354	0.1		
29	0.360	0.1		
30	0.354	0.1		
31	0.354	0.1		
lm	0.354			

Table 3: General Training Parameters for SegProp Experiments

Parameter	**Value**
num_train_epochs	1
per_device_train_batch_size	8
gradient_accumulation_steps	6
max_grad_norm	1.0
learning_rate	1e-6
lr_scheduler_type	linear
warmup_ratio	0.03
weight_decay	0.0
bf16	True
fsdp	full_shard auto_wrap offload
group_by_length	True
load_best_model_at_end	True
prediction_loss_only	True