FLUIDPIPE: LOCAL LEARNING FOR LLM PRETRAINING AND ITS LIMITS

Anonymous authors
Paper under double-blind review

ABSTRACT

Pipeline parallelism (PP) distributes groups of model layers, called stages, across accelerators. The accelerators can sit idle while waiting for gradients from later stages. Local learning removes this wait by training intermediate stages with their own auxiliary losses, but the resulting features may be poorly suited to later stages. We present FLUIDPIPE (FP), a local-learning method for LLM pretraining that adds one decoder block to each intermediate stage's next-token predictor. These blocks are used only for local prediction and discarded at inference. Throughput gains over PP range from 13 to 73% across the tested configurations from 1B to 70B parameters. At 8B, two-stage FP reaches a nine-task mean accuracy within 0.16 points of full backpropagation after about 15B tokens, with 37% higher throughput on eight H100 GPUs. Using more stages, however, leaves fewer layers in each stage. At 1B, FP and three other local methods suffer an accuracy collapse with four stages of four layers. Across different four-stage partitions of the same sixteen layers, FP's accuracy ranges from 34.48 to 49.52. Preventing this collapse is the next step toward using local learning's throughput advantage to reduce LLM pretraining cost across more accelerators.

1 INTRODUCTION

Pipeline parallelism (PP) divides consecutive model layers into groups called *stages*, distributed across accelerators so the model can exceed one device's memory capacity (Huang et al., 2019; Narayanan et al., 2021b). Stages pass features forward, and full backpropagation sends gradients from the final prediction loss back through them. Earlier stages must wait for these gradients before their backward passes. Waiting for features or gradients can leave accelerators idle, creating the *pipeline bubble* (Strati et al., 2024). Pipeline schedules overlap small portions of a batch, called micro-batches, to reduce idle time, but retain this dependency (Narayanan et al., 2019; Qi et al., 2024; Wu et al., 2024) (Figure 1).

Local learning attaches an *auxiliary predictor*, a training-only network that makes predictions from an intermediate stage's features (Belilovsky et al., 2020; Laskin et al., 2020). The predictor's auxiliary

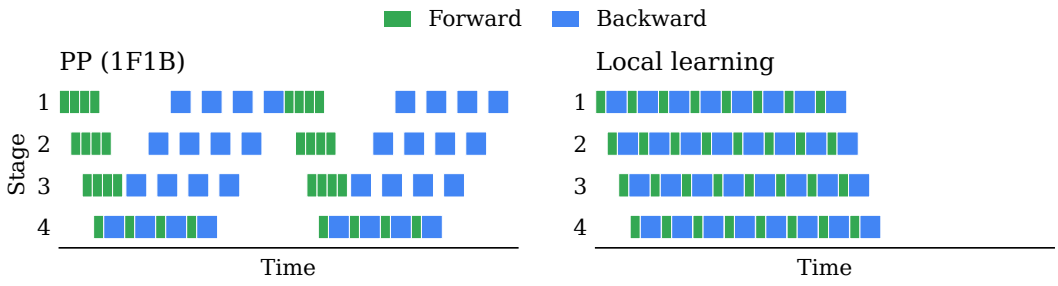


Figure 1: Local learning removes the wait for gradients from later stages. Schematic processing of eight micro-batches by four stages, with idle periods shown as gaps. PP uses one-forward-one-backward (1F1B) scheduling. Local learning still passes features forward but computes each stage's gradients locally.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

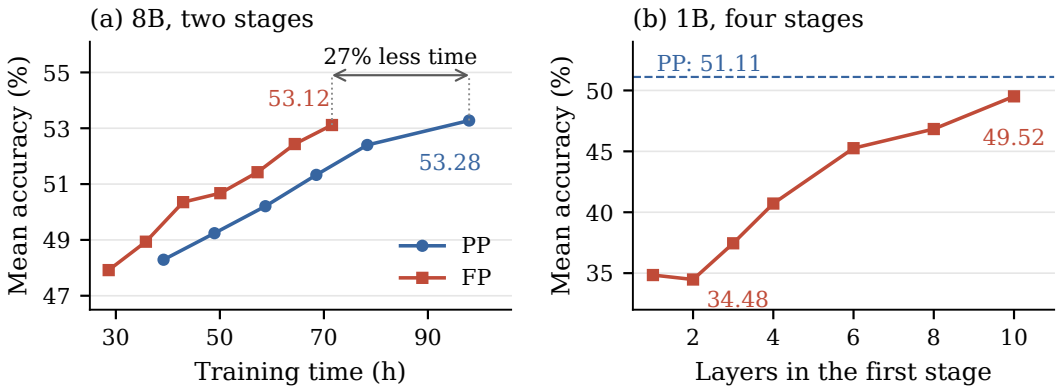


Figure 2: LLM accuracy under local learning depends on the layer partition. (a) Two-stage Llama-3.1 8B after 7,000 steps and 15B tokens. Training time uses the conversion in Section 4. (b) Seven four-stage partitions of the same sixteen-layer Llama-3.2 1B model after 36.7B tokens. Unequal splits test the effect of layer allocation on accuracy, not distributed efficiency. The dashed line is PP. Both panels show nine-task mean accuracy.

loss updates only that stage and predictor, without waiting for gradients from later stages. Features still pass between stages, but gradients do not. An early stage can then improve its own predictions without producing features that help later stages. The auxiliary computation can also offset the time saved by removing the backward dependency.

Prior methods such as Decoupled Greedy Learning (DGL) (Belilovsky et al., 2020), InfoPro (Wang et al., 2021) and AugLocal (Ma et al., 2024) demonstrate local learning for image classification. Laskin et al. (2020) also study four- and six-layer Transformers on the One Billion Word benchmark. We investigate whether local learning can preserve quality while accelerating billion-parameter LLM pretraining over tens of billions of tokens.

We present FLUIDPIPE (FP), a local-learning method that limits each auxiliary predictor to one decoder block and a next-token classifier. The block has the same architecture as a layer of the original model. It processes features for local prediction, while the next stage receives the features from before the block. These auxiliary blocks and classifiers are removed at inference. Short-budget ablations show lower perplexity with the block on both tested 1B models (Table 2).

With two sixteen-layer stages, 8B FP reaches a nine-task mean accuracy 0.16 points below full backpropagation after about 15B tokens (Figure 2a).[1] Its throughput is 37% higher on eight H100 GPUs. Across the tested configurations from 1B to 70B parameters, throughput gains over PP range from 13 to 73% (Section 4.2).

Using more stages reduces model memory per stage but also leaves fewer layers to learn together. We call stages with fewer layers *thinner* stages, and find that these splits can cause severe accuracy collapse. At 1B, FP, DGL, InfoPro and AugLocal all score more than five points below PP with four stages of four layers. At 8B, FP's accuracy falls from 53.12 with two stages to 38.68 with eight (Section 4.4).

To investigate the failure, we change where one stage ends and the next begins, the *stage boundaries*. Keeping the same sixteen layers and four stages, different allocations yield accuracies from 34.48 to 49.52 (Figure 2b). Which layers learn together matters, not just how many stages the model has. These unequal partitions are diagnostic experiments, not proposed deployment configurations.

We make two contributions. **First**, FP demonstrates two-stage 8B pretraining close to full-backpropagation accuracy at higher throughput. **Second**, we identify and investigate thin-stage accuracy collapse, a negative result shared by all four tested local methods at 1B and by FP at 8B. Changing stage boundaries alters its severity even at fixed model depth and stage count. Preventing this collapse without losing the throughput advantage is necessary to make local learning effective for LLM pretraining across more accelerators.

[1] Nine-task means and their gaps are computed before rounding.

2 Related Work

Pipeline parallelism. PipeDream, PipeDream-2BW and PipeMare relax synchronization to keep stages busy, using delayed updates or multiple weight versions (Narayanan et al., 2019; 2021a; Yang et al., 2021). These approaches retain cross-stage backpropagation. Zero-Bubble schedules distinguish the gradients needed by the preceding stage from those needed to update the current stage's weights, and delay the latter to fill idle periods (Qi et al., 2024). Local-learning methods, including FP, remove this backward exchange altogether.

Decoupled and local learning. Decoupled Greedy Learning (DGL) trains groups of layers through auxiliary classifiers without gradients from later groups (Belilovsky et al., 2020; 2021). For our LLM comparison, DGL's predictor uses a multilayer perceptron (MLP), a feedforward network applied separately to each token's features, before the vocabulary classifier. FP instead uses a decoder block, which also applies attention across preceding token positions. Laskin et al. (2020) demonstrate that local updates can achieve higher throughput than GPipe when training ResNets on Graphcore IPUs. We extend the investigation to billion-parameter decoder pretraining and downstream accuracy on GPUs.

Local-learning methods differ in how they supervise intermediate features. InfoPro adds a loss for reconstructing the input, encouraging features to retain information useful to later layers (Wang et al., 2021). Our language-model adaptation reconstructs the input token embeddings. Blockwise SSL uses self-supervised objectives, while SEDONA searches for auxiliary architectures (Siddiqui et al., 2024; Pyeon et al., 2021). PETRA instead uses reversible architectures to parallelize end-to-end training (Rivaud et al., 2025).

AugLocal builds auxiliary networks using the architectures of layers selected from later in the model, with fewer auxiliary layers toward the output (Ma et al., 2024). FP fixes this depth at one block matching the stage's own layers, with no per-stage depth tuning. When all decoder layers share an architecture, AugLocal's one-block setting can coincide with FP. Both methods forward features from before the auxiliary branch and discard that branch at inference. FP requires less auxiliary computation and achieves higher A100 throughput than the deeper AugLocal configurations in our comparison (Section 4.3). Our vision study also evaluates AugLocal's one-block construction (Section 4.6).

Additional connections to periodic synchronization and federated depth splits appear in Appendix A.

Table 1: LLM adaptations of the compared local-learning methods. Each predictor ends in a vocabulary classifier, which maps token features to vocabulary scores. Auxiliary branches are discarded at inference. Normalization layers are omitted.

Method	Processing before the vocabulary classifier	Additional objective
FP (no block)	None	None
FP	One decoder block	None
DGL	Token-wise MLP	None
AugLocal	Multiple decoder blocks, fewer toward the output	None
InfoPro	None	Embedding reconstruction

3 FluidPipe Design

Local prediction and forwarded features. We call the original model's decoder layers the *backbone*. FP partitions them into K consecutive stages $f_1, \ldots, f_K$. At each intermediate stage, the output features serve two purposes, predicting the next token locally and supplying the next stage's input. FP separates these paths with an auxiliary block a_s, followed by a vocabulary classifier g_s. The block has the same hidden width and attention structure as a backbone layer, hence *backbone-matched*. For a non-final stage,

$$h_s = f_s(\bar{h}_{s-1}), \qquad z_s = a_s(h_s), \qquad \hat{y}_s = g_s(z_s), \tag{1}$$

Here $\bar{h}_{s-1}$ denotes the preceding stage's features, treated as constant so that gradients do not flow into that stage. The local classifier predicts from z_s, while the next stage receives h_s, the features

from before the auxiliary block (Figure 3). This lets the auxiliary block specialize in next-token prediction without requiring later stages to use its outputs. We test both the benefit of this block and the choice to forward features from before it.

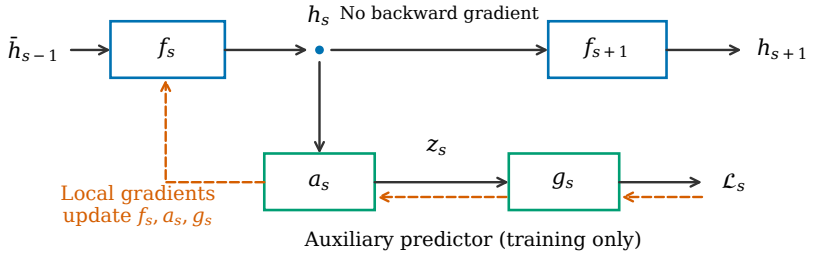


Figure 3: The FP auxiliary branch. Stage $s+1$ receives the backbone features h_s, before the auxiliary block a_s and vocabulary classifier g_s. Dashed arrows show local gradients updating f_s, a_s and g_s, but no parameters in another stage.

Stage-local optimization. Every stage predicts the same next-token targets y using its own scores $\hat{y}_s$ and loss,

$$\mathcal{L}_s = \mathcal{L}_{\text{task}}(y, \hat{y}_s), \tag{2}$$

where $\mathcal{L}_{\text{task}}$ is softmax cross-entropy over the vocabulary at each unmasked target position. The final stage uses the original model's classifier, without an auxiliary block. For $s < K$, the gradient of $\mathcal{L}_s$ updates f_s, a_s and g_s, but no parameters in any other stage. Thus a stage learns from its own predictions, but receives no correction from errors made by later stages. Its auxiliary block still affects the features it learns through this local gradient.

Each stage can run its backward pass after computing its local loss, without waiting for a later stage. If an optimizer step contains several micro-batches, their gradients are accumulated before updating that stage's parameters. Only the forward features are sent between stages. Further details and pseudocode are in Appendix G.

One backbone-matched auxiliary block. The auxiliary block lets each token attend to preceding tokens before prediction. The MLP in our DGL adaptation instead transforms each token's features independently (Table 1). FP adds $K-1$ auxiliary blocks during training, regardless of how the backbone layers are divided among stages. The auxiliary blocks and classifiers have separate trainable parameters and are discarded at inference. Deeper auxiliary networks provide more processing but add work to every local update. FP limits this additional work to one block per intermediate stage.

Block ablation in short pretraining. We test the block by pretraining Llama-3.2 1B and OLMo-2 1B (Team OLMo et al., 2025) with and without it for 20,000 steps at four stages. Each step processes fewer tokens than in the main pretraining study. Removing a_s retains the local classifier and next-token objective, and both variants have the same inference architecture. Adding the block lowers the full model's WikiText word perplexity from 208.9 to 174.0 on Llama and from 71.5 to 60.4 on OLMo-2 (Table 2). Section 4.3 compares the training cost of these auxiliary designs.

4 Experiments

We compare FP with full-backpropagation PP, then vary the auxiliary predictor and the division into stages to test where local learning loses accuracy.

Experimental setup. We pretrain Llama-3.2 1B and Llama-3.1 8B (Grattafiori et al., 2024) from scratch on FineWeb-Edu (Penedo et al., 2024). The 1B model has sixteen layers and trains for 17,500 steps, consuming 36.7B tokens. The 8B model has thirty-two layers and trains for 7,000 steps, consuming about 15B tokens. Unless stated otherwise, quality is the unweighted mean accuracy over nine downstream tasks, evaluated using all backbone layers and the final classifier. The tasks

Table 2: Full-model quality and training throughput with four stages after short-budget pretraining. WikiText word perplexity (lower is better) follows 20K training steps. Throughput is in tokens/s on four A100-40GB GPUs, using each implementation's reported batch settings. PP uses 1F1B. Both models have 1B parameters. Llama trains on FineWeb-Edu (Penedo et al., 2024) and OLMo-2 on Wikipedia. Selected learning rates and full recipes are in Appendix H.1.

	Llama-3.2		OLMo-2	
Method	Perplexity ↓	Tokens/s ↑	Perplexity ↓	Tokens/s ↑
PP	204.0	35,700	60.1	33,500
DGL	196.8	42,282	70.5	36,600
InfoPro	226.2	42,362	81.5	37,100
AugLocal	**172.9**	32,631	**59.4**	27,710
FP (no block)	208.9	—	71.5	—
FP	174.0	**44,492**	60.4	**41,431**

and per-task results are in Appendix C.2. The 1B PP reference uses the same backbone as FP at learning rate 3×10^{-4}. The three local-learning baselines use the same 1B token budget (Section 4.4). Throughput is the number of training tokens processed per second. Hardware and recipes are specified alongside each comparison and in Appendix H. Training time is computed by dividing token counts by throughput from dedicated timing runs, excluding evaluation and restart overhead.

4.1 Two-stage 8B pretraining approaches full-backpropagation accuracy

With two sixteen-layer stages trained locally from random initialization, FP reaches 53.12 nine-task mean accuracy against 53.28 for PP after 7,000 steps and 15B tokens (Figure 2a). At this checkpoint, the learning rate has not completed its cosine decay, which is configured to end at step 17,500. Each configuration uses one training seed. The mean gap is 0.16 points, but task-level differences are larger, including $+5.47$ points on BoolQ and -3.34 on HellaSwag (Appendix C.2). At 1B, FP leads PP at some early checkpoints but falls behind later (Appendix C.3). Whether the close 8B accuracy persists beyond the evaluated budget remains open.

On eight H100 GPUs, FP processes 56,990 tokens/s against 41,642 for PP (Table 3). This 37% throughput increase gives training times of 97.93 and 71.55 hours for PP and FP at that token budget, a 27% saving.

Table 3: Throughput in tokens/s, with gains over PP in parentheses and the faster FP variant in bold. H100 runs use two nodes, with active GPU counts below. PP uses all-forward-all-backward (AFAB) scheduling. FP + chunked computes the vocabulary loss in smaller pieces to save memory. PP does not use this optimization. The 70B result measures throughput only (Appendix F).

Scale	GPUs	Stages	PP	FP	FP + chunked
1B	4×H100	2	87,822	127,461 (+45%)	**130,554 (+49%)**
1B	4×H100	4	92,242	93,289 (+1%)	**104,266 (+13%)**
8B	8×H100	2	41,642	**56,990 (+37%)**	56,513 (+36%)
8B	8×H100	4	37,725	49,303 (+31%)	**53,754 (+42%)**
8B	8×H100	8	27,993	39,526 (+41%)	**44,017 (+57%)**
70B	64×H200	8	31,000	53,500 (+73%)[†]	

[†]One FP configuration and a 1,000-step PP reference, without a chunking comparison.

4.2 FP improves throughput from 1B to 70B

The speed advantage extends beyond two-stage 8B training. The best FP configuration in each partition improves throughput over PP by 13 to 73% across 1B, 8B and 70B (Table 3). With the GPU count held fixed, two stages give the highest FP throughput in both the 1B and 8B sweeps. At 8B, the relative gain over PP grows from 37% at two stages to 57% at eight, where the selected FP configuration uses chunked cross-entropy.

Each token prediction produces a score for every vocabulary item, called a logit. *Chunked cross-entropy* computes the loss in smaller pieces to avoid holding all these scores in memory at once. We apply it to FP because every stage has a vocabulary classifier, whereas PP has one only at its final stage. Table 3 shows both FP variants. Without chunking, FP remains faster in every H100 partition, with gains from 1 to 45%. The two-stage 8B comparison above uses no chunking for either method.

We tune PP's micro-batch size and gradient accumulation for each partition and report the fastest configuration found. PP uses AFAB, which completes the batch's forward passes before its backward passes, on H100 and H200. It uses 1F1B on A100. In the A100 sweep, 1F1B outperforms the tested Zero-Bubble implementation, whose memory footprint forces smaller micro-batches (Appendix B.4). Activation recomputation saves memory by repeating forward computation during the backward pass. It permits larger micro-batches and gives the fastest PP configuration on H200 (Appendix F.1). The selected FP configurations do not require recomputation. A cost model in Appendices B.1 and B.2 compares the saved pipeline idle time and backward communication with the extra auxiliary computation.

4.3 Auxiliary prediction trades quality against computation

Removing backward communication is common to local-learning methods. Their auxiliary predictors determine how much additional work each stage performs. FP improves both perplexity and A100 throughput over DGL and InfoPro on the two short-budget benchmarks (Table 2). AugLocal has the lowest perplexity, but FP processes 36% more tokens per second on Llama and 50% more on OLMo-2. With these deeper predictors, AugLocal is slower than PP on both models.

At the longer 36.7B-token budget, the downstream-accuracy ranking depends on the partition (Table 4). With two stages, FP reaches 49.65 against 47.33 for DGL and 49.02 for InfoPro, while AugLocal reaches 50.18. With four stages, DGL and InfoPro exceed FP, although all four local methods remain well below PP.

4.4 Accuracy collapses when LLMs are split into thin stages

At 1B, splitting the sixteen-layer model into four stages of four layers causes a large accuracy collapse. All four tested local methods remain more than five accuracy points below full backpropagation after 36.7B tokens (Table 4). PP reaches 51.11 and FP 40.71. DGL reaches 44.88, InfoPro 45.56 and AugLocal 35.73 at the same step. Even at two stages, 1B FP is 1.46 points below PP, compared with 0.16 at the shorter 8B budget. Each two-stage 8B partition contains twice as many layers as its 1B counterpart. We therefore test whether 8B FP retains its quality when those stages are made thinner.

The collapse persists at 8B. At step 7,000, accuracy falls from 53.12 with two stages to 50.17 with three, 48.67 with four, and 38.68 with eight (Table 5). These runs share the model, learning-rate schedule and 15B-token budget. Eight stages leave four layers per stage and a 14.59-point gap to PP. The larger relative throughput gains at these partitions do not preserve the two-stage model's accuracy. Two stages give both the highest accuracy and the highest absolute FP throughput in the 8B comparisons (Table 3).

Table 4: 1B full-network accuracy after 36.7B tokens. Local methods share a learning rate within each column, but not across columns (Appendix H). PP is the reference of Section 4.

Method	Two stages	Four stages
PP	51.11	
FP	49.65	40.71
DGL	47.33	44.88
InfoPro	49.02	45.56
AugLocal	50.18	35.73

Table 5: 8B full-network accuracy after about 15B tokens. Splits are approximately even. Each run uses one seed, learning rate 6×10^{-4} and the same schedule.

Stages	Layers per stage	Accuracy
PP	—	53.28
2	16	53.12
3	10 to 11	50.17
4	8	48.67
8	4	38.68

4.5 COLLAPSE DEPENDS STRONGLY ON EARLY BOUNDARY PLACEMENT AT 1B

The stage-count experiments change both the number of gradient boundaries and the number of layers that learn together. We compare twelve partitions of the same sixteen-layer model under one training recipe, including seven with four stages (Appendix H.10). The unequal partitions test how local learning fails, not efficient deployment.

Holding the number of stages fixed. Brackets list each stage's layer count from input to output. Changing [2,4,4,6] to [10,2,2,2] raises accuracy from 34.48 to 49.52 without changing the backbone or number of stages (Figure 2b). The gap to PP is 1.59 points, compared with 10.39 for the even split. The number of gradient boundaries alone cannot explain this difference.

Moving the thin stage from the beginning to the end. The partition [6,4,4,2] reaches 45.26 against 34.48 for [2,4,4,6]. The two-stage partition [2,14] also falls to 35.76, below the even four-stage split at 40.71. Training fourteen later layers together does not compensate for placing the first local loss after just two layers.

Holding the first stage fixed. With eight layers in the first stage, [8,6,2] and [8,2,6] reach 47.68 and 47.63, while [8,3,3,2] falls to 46.83. Later boundaries still matter, though less than first-stage depth in this series.

Prediction through successive stages. We next evaluate each stage's trained predictor to see whether later layers improve on the first stage's predictions. With two stages at 1B, the full-model predictor scores 1.19 points above the first-stage predictor (Appendix D.1). With four stages, successive predictors score 47.72, 40.91, 40.82 and 40.80 at step 17,000 (Figure 4a). Most of the drop appears at the second predictor, and the remaining predictors do not recover it. The first-stage predictor improves during training without a corresponding rise in the other three. A widening first-to-final gap also appears in eight-stage 8B training (Figure 4b). A possible explanation is that the first stage learns features suited to its own predictor but poorly suited to the later stages. Those stages can transform the features, but cannot send gradients back to change how they are produced.

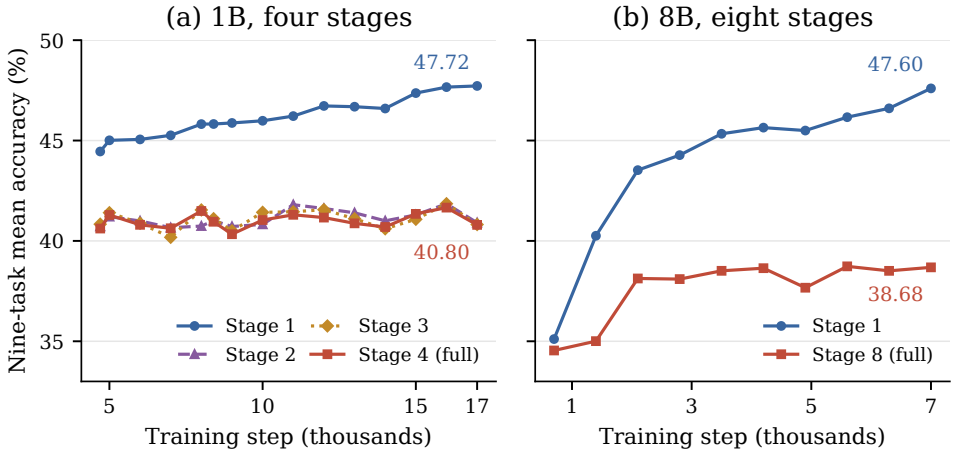


Figure 4: Later stages need not improve prediction under thin partitions. (a) All four FP predictors at 1B. (b) The first and final predictors at eight-stage 8B. Both models have four backbone layers per stage. Each intermediate predictor uses the backbone up to that stage and its trained auxiliary branch. The final predictor uses the full backbone and original classifier. All are evaluated at common checkpoints on the same nine tasks, without refitting. Further comparisons are in Appendix D.1.

4.6 CONTROLLED ANALYSIS IN A LOWER-COST VISION SETTING

We use ResNets on CIFAR-100 as a lower-cost setting for testing how the auxiliary block affects learning and the features passed between stages. Each stage predicts the image class through one

additional ResNet block and a linear classifier, in place of the decoder block and vocabulary classifier. We first check whether these models reproduce the collapse seen in LLMs.

Reproducing collapse. Purely local training loses 7.84 accuracy points against end-to-end training for ResNet-50 at sixteen stages, 8.02 for ResNet-101 at twenty-four, and 7.57 for ResNet-152 at thirty-two (Appendix I). On CINIC-10, another image-classification dataset, collapse is less severe and appears at higher stage counts (Appendix I). Unlike the 1B language model, two-stage vision networks tolerate a thin first stage (Appendix I). Thus collapse is not specific to language modeling, although the effect of first-stage depth differs between these settings.

The block helps more without cross-stage gradients. We fix ResNet-152 and train each variant for 200 epochs from identical backbone weights (Appendix I). Adding the block improves accuracy by 7.79 points at sixteen local stages, compared with 2.07 at four (Figure 5a). We then keep every auxiliary loss but allow it to backpropagate through earlier stages. At sixteen stages, the block's benefit then falls to 2.72 points. Training the same backbone from the final classifier's loss alone reaches 80.49% accuracy, compared with 81.00% when auxiliary losses also backpropagate through earlier stages (Appendix I).

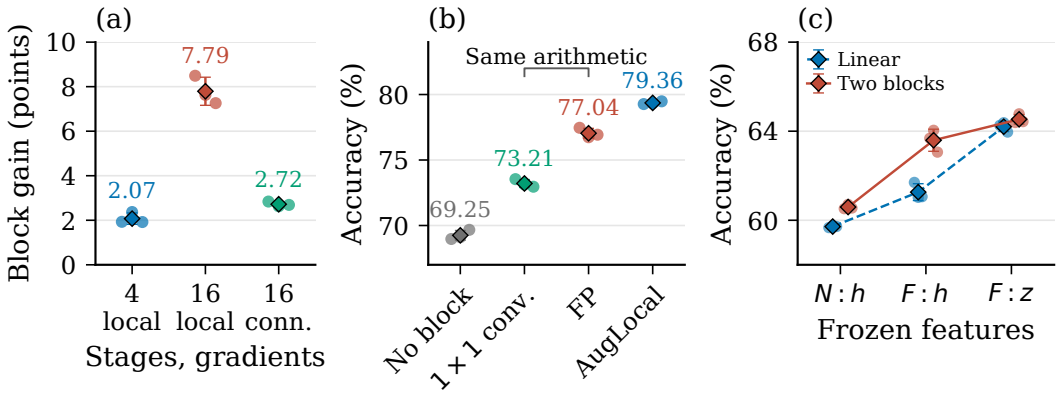


Figure 5: Controlled auxiliary-block study on ResNet-152, CIFAR-100. (a) Accuracy gain from adding the block when each loss updates only its own stage (local) or also earlier stages (conn.). (b) Sixteen-stage local accuracy. The 1×1 replacement uses as many forward operations as FP's auxiliary predictor. (c) New classifiers trained on fixed first-stage features from no block (N:h), FP before its block (F:h), and FP after it (F:z). Dots denote seeds, diamonds three-seed means, and bars sample standard deviations. Full protocols and seed-level results are in Appendix I.

Does extra computation explain the gain? We replace the block with two 1×1 convolutions, which transform feature channels at each image location without combining neighboring locations. We choose their width to match the number of multiply-add operations in a forward pass through FP's auxiliary predictor. This replacement reaches 73.21% accuracy against 77.04% for FP (Figure 5b). The gap is 3.83 points at equal forward computation. A wider one-block AugLocal branch reaches 79.36%, with greater auxiliary computation.

The gain reaches the forwarded features. After training FP and the no-block variant, we hold the first stage's weights fixed and train new classifiers on its outputs. These are either linear or contain two residual blocks before classification, allowing further processing of the fixed features. A two-block classifier reaches 63.59% on FP's forwarded features versus 60.59% without the block. A linear classifier gains 1.55 points (Figure 5c). Both classifiers perform best on the block's outputs. The block's benefit thus extends to the forwarded features, evaluated without the original predictor.

Keeping the block on the local prediction branch. The new classifiers favor the block's outputs. Would forwarding those outputs also improve training? We retrain sixteen-stage ResNet-152 with the next stage receiving features from after the auxiliary block. Accuracy falls from 77.04% to 72.58% when this changes only at the first boundary, and to 71.09% when it changes at every boundary

(Appendix I). The original choice, forwarding features from before the block, gives higher accuracy in these training comparisons and leaves the inference model unchanged.

4.7 Further results on fine-tuning

We also apply local learning to pretrained models. At 1B, switching from full backpropagation to local learning preserves substantially more accuracy than training locally from scratch (Appendix C.4).

Classification fine-tuning. With four stages, FP stays close to or above PP on five classification datasets using ViT-Large and BERT-Large (Appendix E). Across three-seed means, it leads the tested local methods on CIFAR-100 and AG News and is no more than 0.18 points below the best local baseline on the other tasks. The auxiliary block's effect is task-dependent (Appendix D.4). Two-stage ViT throughput is also higher than PP across the tested batch sizes (Appendix E.3).

Math fine-tuning. For Llama-2-7B fine-tuned on MetaMathQA, two-stage FP reaches PP's final GSM8K accuracy in fewer optimizer steps and scores higher on MATH (Appendix C.5).

5 Conclusion

Local learning can train an LLM faster by letting each stage compute its gradients without waiting for later stages. With one auxiliary decoder block per intermediate stage, FP reaches within 0.16 accuracy points of full backpropagation in two-stage 8B pretraining after 15B tokens. Throughput gains range from 13 to 73% over the tested PP configurations from 1B to 70B.

The obstacle to using more stages is accuracy collapse. It appears across different local-learning methods, and changing which layers train together can alter its severity even at a fixed stage count. Two stages give the strongest tested 8B result, but do not solve this problem for finer partitions. Developing methods that mitigate or prevent this collapse is the natural next step toward using local learning to reduce LLM pretraining cost across more accelerators.

REFERENCES

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled Greedy Learning of CNNs. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pp. 736–745. PMLR, 2020.

Eugene Belilovsky, Louis Leconte, Lucas Caccia, Michael Eickenberg, and Edouard Oyallon. Decoupled Greedy Learning of CNNs for Synchronous and Asynchronous Distributed Learning. *arXiv preprint arXiv:2106.06401*, 2021.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pp. 7432–7439, 2020.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 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)*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. DiLoCo: Distributed Low-Communication Training of Language Models. *arXiv preprint arXiv:2311.08105*, 2023.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. Zenodo, software release v0.0.1, 2021. URL `https://doi.org/10.5281/zenodo.5371629`.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 14068–14080, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks)*, volume 1, 2021.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Michael Laskin, Luke Metz, Seth Nabarro, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel Training of Deep Networks with Local Updates. *arXiv preprint arXiv:2012.03837*, 2020.

Chenxiang Ma, Jibin Wu, Chenyang Si, and Kay Chen Tan. Scaling Supervised Local Learning with Augmented Auxiliary Networks. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54, pp. 1273–1282. PMLR, 2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations (ICLR)*, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2381–2391, 2018.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 1–15, 2019.

Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-Efficient Pipeline-Parallel DNN Training. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139, pp. 7937–7947. PMLR, 2021a.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021b.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1525–1534, 2016.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 37, 2024.

Myeongjang Pyeon, Jihwan Moon, Taeyoung Hahn, and Gunhee Kim. SEDONA: Search for Decoupled Neural Networks toward Greedy Block-wise Learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero Bubble (Almost) Pipeline Parallelism. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE/ACM, 2020.

Stéphane Rivaud, Louis Fournier, Thomas Pumir, Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. PETRA: Parallel End-to-end Training with Reversible Architectures. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pp. 8732–8740, 2020.

Shoaib Ahmed Siddiqui, David Krueger, Yann LeCun, and Stéphane Deny. Blockwise Self-Supervised Learning at Scale. *Transactions on Machine Learning Research*, 2024.

Foteini Strati, Paul Elvinger, Tolga Kerimoglu, and Ana Klimovic. ML Training with Cloud GPU Shortages: Is Cross-Region the Answer? In *Proceedings of the 4th Workshop on Machine Learning and Systems (EuroMLSys)*, pp. 107–116, 2024.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, et al. 2 OLMo 2 Furious. *arXiv preprint arXiv:2501.00656*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*, 2023.

Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. Revisiting Locally Supervised Learning: an Alternative to End-to-end Training. In *International Conference on Learning Representations (ICLR)*, 2021.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing Multiple Choice Science Questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text (W-NUT)*, pp. 94–106, 2017.

Erik Wijmans, Brody Huval, Alexander Hertzberg, Vladlen Koltun, and Philipp Krähenbühl. Cut Your Losses in Large-Vocabulary Language Models. In *International Conference on Learning Representations (ICLR)*, 2025.

Houming Wu, Ling Chen, and Wenjie Yu. BitPipe: Bidirectional Interleaved Pipeline Parallelism for Accelerating Large Models Training. *arXiv preprint arXiv:2410.19367*, 2024.

Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher Aberger, and Christopher De Sa. PipeMare: Asynchronous Pipeline Parallel DNN Training. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 3, pp. 269–296, 2021.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4791–4800, 2019.

AI USE STATEMENT

We used AI assistants to help write and polish the manuscript, find related work, and prepare figures. The authors take responsibility for the final manuscript and its claims.

APPENDIX

Appendix roadmap.

Appendix A	Additional related work
Appendix B	Throughput analysis and measurements
Appendix C	$K{=}2$ quality details
Appendix D	Thin-stage pretraining detail
Appendix E	Fine-tuning details
Appendix F	70B throughput details
Appendix G	FP algorithm
Appendix H	Detailed per-experiment recipes
Appendix I	Vision collapse detail
Appendix J	Reproducibility details

A ADDITIONAL RELATED WORK

Periodic synchronization and depth-split federated training. FedAvg (McMahan et al., 2017) and DiLoCo (Douillard et al., 2023) train independent model replicas that periodically synchronize. This is Local SGD applied along the *data-parallel* axis, and every replica holds the whole model. FP partitions one model along depth, and its stages never exchange gradients. FedGKT (He et al., 2020) partitions a single model by depth between federated clients and a central server. Clients train the early layers locally and send extracted features to the server, which continues forward through the remaining layers. This depth-split, feature-forwarding topology closely mirrors FP's stage-local training, though FedGKT targets communication-constrained federated settings rather than pipeline-parallel throughput.

B THROUGHPUT ANALYSIS AND MEASUREMENTS

B.1 THROUGHPUT MODEL

Local learning avoids backward communication and some pipeline idle time, at the cost of computing auxiliary predictions and their gradients. It can also release stored features, or activations, sooner because the local backward pass does not wait for later stages. This reduces memory pressure and can avoid *activation recomputation*, which saves memory by repeating forward computation during the backward pass.

Consider a run with K stages and N_b mini-batches, each split into M micro-batches. Boundary s connects stage s to stage $s{+}1$ for $s < K$. Let t_s^{PP} and t_s^{FP} denote the forward and backward compute time for stage s on one micro-batch. Both include any activation recomputation, and t_s^{FP} also includes the auxiliary block and vocabulary classifier. Let τ_s^{act} be the activation-transfer time across boundary s, and τ_s^{grad} the gradient-return time. At the start of a batch, later stages wait for the first micro-batch to reach them. At the end, earlier stages finish while the remaining work drains from the pipeline. A synchronous fill-and-drain approximation counts M useful micro-batch slots and $K-1$ slots for this overhead, with each slot including forward and backward work. Under PP, each micro-batch sends activations forward and gradients backward across every boundary. Without compute–communication overlap, the slowest stage computation and largest boundary-transfer cost set the pace of each slot:

$$T_{\mathrm{PP}} \approx N_b\,(M + K - 1)\left(\max_s t_s^{\mathrm{PP}} + \max_{s<K}\left(\tau_s^{\mathrm{act}} + \tau_s^{\mathrm{grad}}\right)\right). \tag{3}$$

Under FP no gradient returns across a boundary and no stage waits for a later stage. Without a barrier across stages at each optimizer step, the initial fill need not repeat every mini-batch and becomes negligible for $N_b \gg K$. Each mini-batch then costs M slots at the pace of the slowest stage and its activation send:

$$T_{\mathrm{FP}} \approx N_b\,M\left(\max_s t_s^{\mathrm{FP}} + \max_{s<K}\tau_s^{\mathrm{act}}\right). \tag{4}$$

In this approximation, FP avoids returning gradients and repeatedly filling and draining the pipeline. At a fixed number of micro-batches M, the fill-and-drain overhead grows with the number of stages K. Removing it produces a speedup only if the saved time exceeds the auxiliary computation included in t_s^{FP}. Different PP schedules overlap different parts of this cost, so hardware comparisons use the named schedules in Section 4.2.

B.2 COMMUNICATION VOLUME

At each boundary s, PP exchanges activation volume v_s and gradient volume γ_s per mini-batch, summed over its M micro-batches. FP sends activations alone:

$$V_{\mathrm{PP}} = N_b \textstyle\sum_{s=1}^{K-1}(v_s + \gamma_s), \tag{5}$$

$$V_{\mathrm{FP}} = N_b \textstyle\sum_{s=1}^{K-1} v_s. \tag{6}$$

The difference is the volume of gradients that PP sends backward. Communication between data-parallel replicas, which train copies of the model on different data, is excluded (Rajbhandari et al., 2020).

B.3 A100 PER-ARCHITECTURE THROUGHPUT

On 4×A100-40GB, FP exceeds the best PP configuration at both $K{=}2$ and $K{=}4$. At $K{=}2$ with TP=2, FP reaches 50,100 tokens/s versus PP's 35,950 tokens/s ($+39\%$). At $K{=}4$, the margin remains $+25\%$ on Llama and $+24\%$ on OLMo-2 (Table 6). Among the local-learning baselines in the A100 comparison, DGL and InfoPro recover part of this advantage ($+9$–19% over PP) but remain below FP, whereas AugLocal falls below PP (Table 2).

B.4 A100 PP THROUGHPUT SWEEP

The best PP micro-batch size differs between the two 1B models. Llama peaks at size four and OLMo-2 at size two, both with 1F1B scheduling. Zero-Bubble PP is 25 to 30% slower than 1F1B at

Table 6: Throughput on 4×A100-40GB, in tokens/s. †1F1B is the one-forward-one-backward pipeline schedule and mbs the micro-batch size.

Architecture	(K, TP)	PP schedule	PP	FP	Δ vs PP
Llama-3.2 1B	(2, 2)	1F1B† mbs=4	35,950	50,100	+39%
Llama-3.2 1B	(4, 1)	1F1B† mbs=4	35,700	44,492	+25%
OLMo-2 1B	(4, 1)	1F1B† mbs=2	33,500	41,431	+24%

four stages in this sweep. Its additional memory footprint forces smaller micro-batches on the 40 GB devices, offsetting the scheduling benefit.

B.5 H100 (K, TP) throughput sweep on 1B and 8B

FP is faster than PP in all five H100 partitions both with and without chunked cross-entropy (Table 3). With chunking disabled, the gains are 45% and 1% at 1B with two and four stages, and 37%, 31% and 41% at 8B with two, four and eight stages. The best FP variant in each partition gives margins from 13 to 57% on H100 (Figure 6). Table 7 records the micro-batch and accumulation settings for both variants and PP. These settings maximize throughput within a partition, rather than hold the effective batch size fixed.

Table 7: Selected H100 throughput configurations, in tokens/s, on two nodes with four GPUs for 1B and eight for 8B. Each row is the highest reported rate for that method, partition and chunking setting. Micro-batch size counts sequences per replica, with sequence length 2048. PP uses AFAB with activation recomputation. FP uses no recomputation.

Model	(K, TP)	Method	Chunked	Micro-batch	Accumulation	Tokens/s
1B	(2, 2)	PP	No	4	8	87,822
1B	(2, 2)	FP	No	16	1	127,461
1B	(2, 2)	FP	Yes	32	1	130,554
1B	(4, 1)	PP	No	4	16	92,242
1B	(4, 1)	FP	No	8	1	93,289
1B	(4, 1)	FP	Yes	32	1	104,266
8B	(2, 4)	PP	No	16	8	41,642
8B	(2, 4)	FP	No	16	1	56,990
8B	(2, 4)	FP	Yes	16	1	56,513
8B	(4, 2)	PP	No	4	16	37,725
8B	(4, 2)	FP	No	8	1	49,303
8B	(4, 2)	FP	Yes	16	1	53,754
8B	(8, 1)	PP	No	1	32	27,993
8B	(8, 1)	FP	No	8	1	39,526
8B	(8, 1)	FP	Yes	16	1	44,017

B.6 Chunked cross-entropy

Chunked cross-entropy (Wijmans et al., 2025) computes the vocabulary loss on one token chunk at a time, avoiding storage of the full token-by-vocabulary logits tensor. Our implementation uses chunks of 4,096 tokens and leaves the model and prediction objective unchanged. FP has a prediction head at every stage, so this memory reduction can allow larger micro-batches across the pipeline. PP produces vocabulary logits only at its final stage. We therefore implemented and swept chunking for FP. PP can also use chunking, but this sweep retains its original loss implementation. Table 3 separates its effect from the non-chunked comparison. At 8B with two stages, the non-chunked FP configuration remains the fastest. At the other H100 partitions, the winning chunked configurations use larger micro-batches and achieve higher throughput.

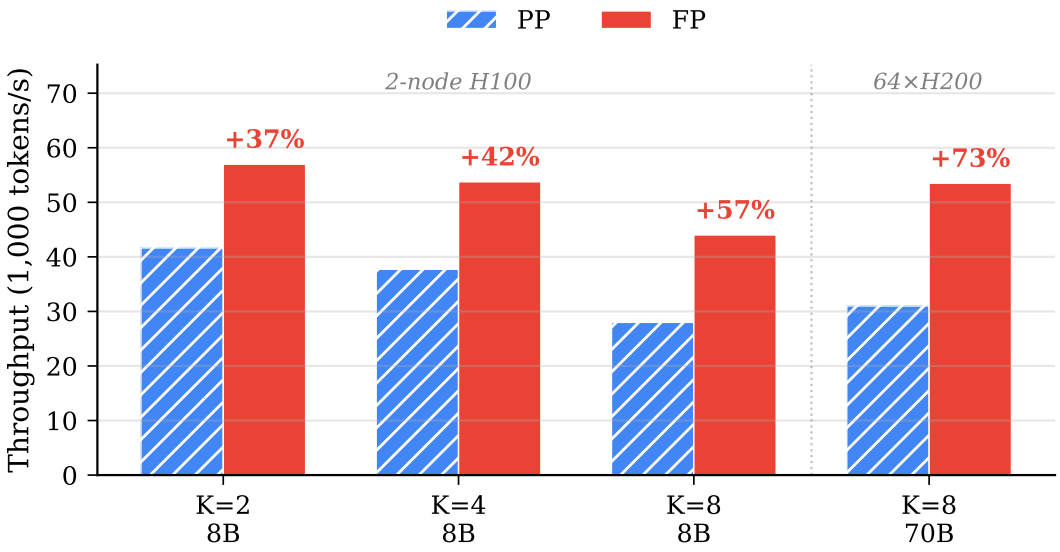


Figure 6: FP throughput versus PP across pipeline depths. The gap widens with K on 8B (2-node, 8-GPU H100-80GB) and extends to 70B (64×H200). Deltas computed from the best configuration per method.

C K=2 QUALITY DETAIL

C.1 8B ACCURACY ACROSS CHECKPOINTS

Why stop at 7,000 steps? We planned the same 17,500-step horizon as at 1B but stopped the 8B runs at 7,000 steps because of computational cost. This consumes 14,680,064,000 tokens, reported as about 15B. The runs retain the planned cosine schedule, which reaches its minimum learning rate at step 17,500. Their final reported checkpoint is an intermediate point in that schedule.

Table 8: The 8B comparison at every common saved checkpoint. Positive gaps favor PP, negative gaps favor FP. These are successive evaluations of the same runs, not independent training seeds.

Step	PP	FP	Gap
2,800	48.29	47.92	+0.37
3,500	49.24	48.94	+0.30
4,200	50.21	50.35	-0.15
4,900	51.33	50.67	+0.66
5,600	52.40	51.43	+0.97
7,000	53.28	53.12	+0.16

C.2 TASK-BY-TASK ACCURACY AT 1B AND 8B

The nine-task mean gives equal weight to HellaSwag, ARC-Easy, ARC-Challenge, PIQA, WinoGrande, OpenbookQA, SciQ, LAMBADA and BoolQ (Zellers et al., 2019; Clark et al., 2018; Bisk et al., 2020; Sakaguchi et al., 2020; Mihaylov et al., 2018; Welbl et al., 2017; Paperno et al., 2016; Clark et al., 2019). At 1B after 17,500 steps, FP's largest deficits against PP are 6.04 points on LAMBADA and 2.91 on BoolQ (Table 9). At 8B after 7,000 steps, FP's differences from PP are $+5.47$ points on BoolQ and -3.34 on HellaSwag. The smaller 8B mean gap combines stronger results on some tasks with a larger deficit on HellaSwag. Excluding BoolQ gives a 0.86-point gap over the remaining eight tasks.

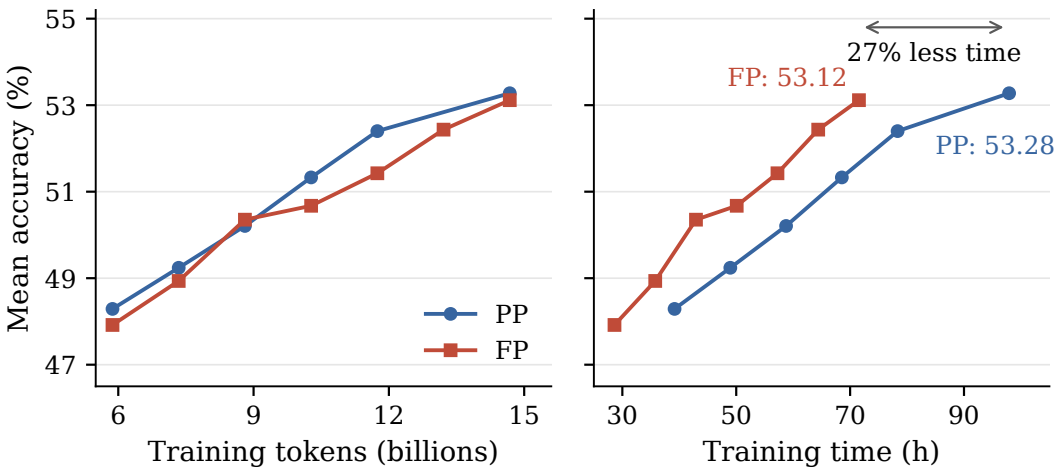


Figure 7: Llama-3.1 8B pretrained from scratch for 7,000 steps, about 15B tokens, with PP or two-stage FP. The left panel shows nine-task mean accuracy against tokens and the right panel shows the same evaluations against training time, using the conversion in Section 4. Markers indicate evaluated checkpoints, with the PP–FP gaps given in Table 8.

Table 9: Task accuracies at 1B after 17,500 steps and 36.7B tokens, and at 8B after 7,000 steps and 15B tokens, in percent, one seed per method. FP uses two local stages at both scales. All evaluations use the full network. Differences are FP minus PP in percentage points, computed before rounding.

	1B, 36.7B tokens			8B, 15B tokens		
Task	PP	FP	Difference	PP	FP	Difference
HellaSwag	44.79	43.64	-1.16	49.26	45.93	-3.34
ARC-Easy	57.11	55.85	-1.26	61.74	60.35	-1.39
ARC-Challenge	30.46	31.23	+0.77	35.15	34.39	-0.77
PIQA	68.06	68.34	+0.27	69.48	69.31	-0.16
WinoGrande	52.49	52.80	+0.32	52.33	54.14	+1.82
OpenbookQA	34.40	32.80	-1.60	35.60	34.60	-1.00
SciQ	74.30	72.80	-1.50	81.20	78.50	-2.70
LAMBADA	37.96	31.92	-6.04	38.48	39.10	+0.62
BoolQ	60.40	57.49	-2.91	56.24	61.71	+5.47
Nine-task mean	51.11	49.65	−1.46	53.28	53.12	−0.16

C.3 Early dynamics in 36.7B-token 1B pretraining

FP $K{=}2$ transiently leads PP at equal steps before PP pulls ahead later in training (Figure 9). At the final step, FP reaches 49.65 and PP reaches 51.11, a difference of 1.46 points (Figure 8b). The per-step trajectories and higher FP throughput yield savings at intermediate quality targets, but FP does not attain PP's peak of 51.22.

Computing time to a quality target. The 1B time conversion uses 2,097,152 tokens per step and throughput of 92,242 tokens/s for PP and 130,554 for two-stage FP. The PP rate is the fastest 1B PP configuration, at $K{=}4$, not the matched-depth $K{=}2$ rate of 87,822 tokens/s (Table 3). We linearly interpolate between adjacent evaluations to estimate when each run first reaches a target. The time reduction is $100(1 - t_{\mathrm{FP}}/t_{\mathrm{PP}})$. The reductions are 34.8%, 39.1% and 28.3% at accuracies 45, 46 and 47, respectively (Table 10).

C.4 Continuing pretraining with local learning

Can local learning continue pretraining after an initial phase of full backpropagation? The hybrid switches from PP to FP $K{=}2$ at step 7K, starting from the reference run's checkpoint, and applies

Table 10: Training time to intermediate nine-task accuracy targets at 1B, using linearly interpolated target crossings and the conversion in Section 4.

Target accuracy	PP hours	FP hours	Time reduction
45	19.55	12.75	34.8%
46	25.37	15.44	39.1%
47	30.16	21.62	28.3%

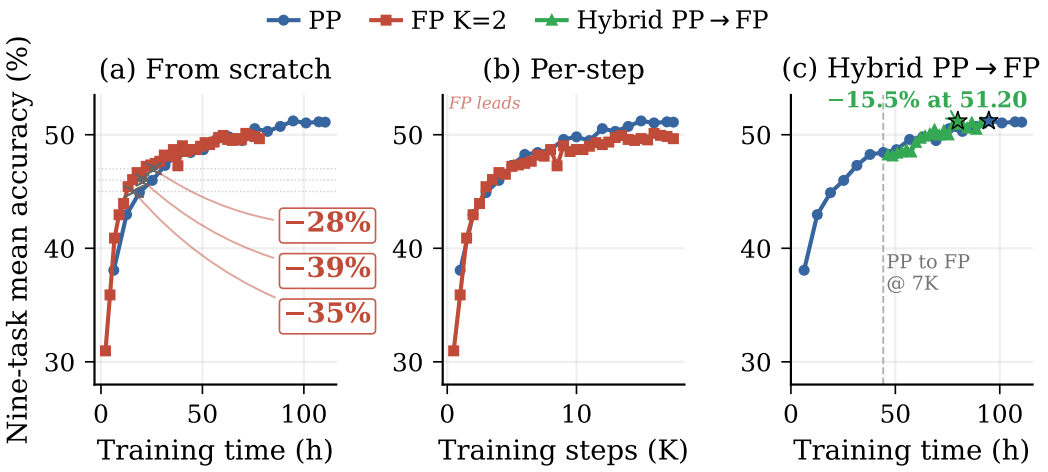


Figure 8: Two-stage FP on 1B pretraining for 36.7B tokens. (a) From-scratch time-to-target comparisons, with savings computed at the first chronological attainment of each target. (b) The same runs against training steps. (c) A PP→FP continuation reaches a target of 51.20 in 15.5% less training time than PP. The hybrid peak is 51.20 and the PP peak is 51.22, so the hybrid comparison uses the lower, jointly attained target.

a short transition learning-rate schedule to the newly introduced stage-local components. Starting with PP gives every layer feedback from the final loss before switching to cheaper local updates. At step 17,500, after 36.7B tokens, the PP→FP continuation reaches 51.05 nine-task mean accuracy, compared with 49.65 for FP trained locally from scratch and 51.11 for PP throughout.

Time to the same accuracy. The continuation peaks at 51.20 at step 15K, compared with 51.22 for PP (Figure 10). To compare time at the same attained accuracy, we use the hybrid's unrounded peak as the target and interpolate PP's first crossing of that target. PP takes 94.52 training hours and the hybrid takes 79.90, a reduction of 15.5%. The hybrid time includes the initial 7K steps at the PP rate and all subsequent steps at the FP rate.

Reversing the order. The mirror hybrid trains FP $K{=}2$ from scratch to step 7K and then continues with PP to step 17.5K. It reaches 50.10 at step 17,250 and 50.18 at step 17,500. The final accuracies are 50.18 for the mirror hybrid, 49.65 for from-scratch FP and 51.11 for PP.

C.5 7B MATH SFT

We also evaluate two-stage local learning in supervised fine-tuning (SFT) of Llama-2-7B (Touvron et al., 2023) on MetaMathQA (Yu et al., 2024). FP reaches 63.5 GSM8K (Cobbe et al., 2021) accuracy at step 7,000, exceeding PP's 63.4 at the end of its 9,257-step schedule with 24.4 percent fewer optimizer steps. At the same step it exceeds PP on MATH (Hendrycks et al., 2021) by 4.2 points (17.7 against 13.5). A short transition learning-rate schedule warms up the newly introduced stage-local components, as in the hybrid run of Appendix C.4. Continuing the run from step 7,000 to step 9,000 raises MATH from 17.7 to 18.0 (+4.5 points over PP) while GSM8K remains within 0.8 points of the PP reference. The recipe is in Table 28.

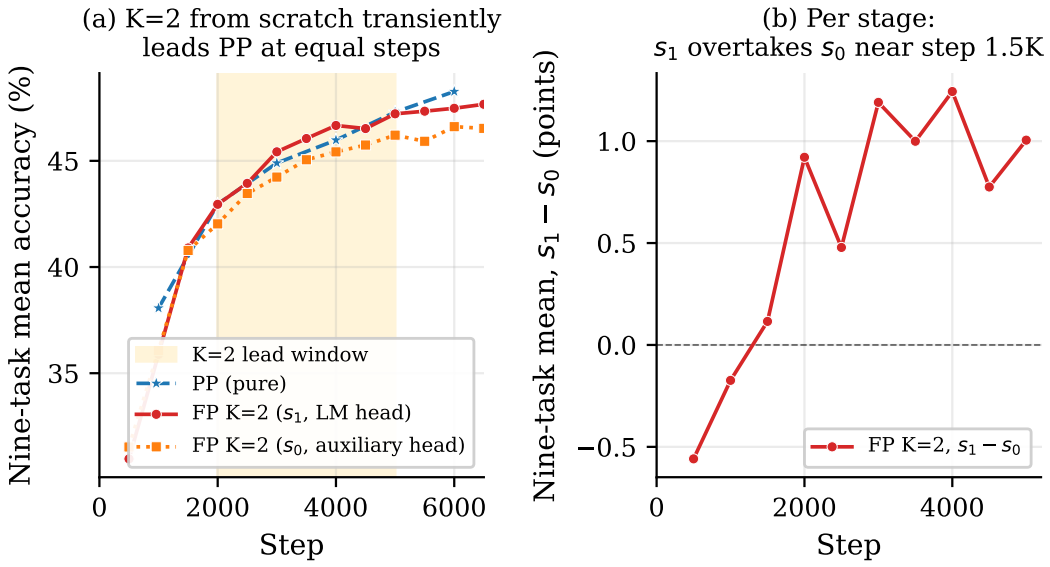


Figure 9: Early dynamics at $K=2$ on 1B pretraining for 36.7B tokens. (a) Nine-task mean accuracy of FP $K=2$ and PP against step. FP is above PP at steps 3,000 and 4,000, and below it at step 5,000. (b) The difference between the two stages of the FP run, s_1 minus s_0, which turns positive near step 1.5K.

C.6 Per-stage learning at K=2

The stage with the LM head (s_1) overtakes the auxiliary-head stage (s_0) by step 1,500 and remains ahead at every later saved checkpoint (Figure 11a). This comparison evaluates two prediction paths within the trained model, not the effect of removing the auxiliary objective.

D Thin-stage pretraining detail

D.1 Stage-by-stage prediction

Does each additional stage improve prediction, and where does this fail during collapse? We evaluate every stage head in the two-stage and four-stage 1B runs at common saved checkpoints, using the same nine downstream tasks. An intermediate readout passes the input through the backbone up to that stage, then its auxiliary block and vocabulary head. The full-model path uses every backbone stage and the final head, without auxiliary branches. All readouts use the weights learned during pretraining, with no additional fitting.

Figure 11 contrasts a setting in which the remaining stage improves prediction with one in which later stages fail to do so. At two stages, the full model remains ahead from step 1,500 onward and finishes at 49.65 against 48.46 for the first-stage predictor, a 1.19-point gain. At four stages, the largest drop is between the first and second predictors. It reaches 6.81 points at step 17,000, with little further change across the remaining heads (Table 11). The four-stage comparison ends at the last checkpoint evaluated through every head. The full-model score at step 17,500 remains the endpoint in the main tables.

These differences compare trained prediction paths, not independently trained smaller models or the causal contribution of each stage. In particular, the first-stage predictor uses the auxiliary block's outputs, while the next stage receives features from before that block. A lower later-stage score therefore does not by itself establish irreversible information loss. The vision study in Section 4.6 tests the forwarded features with fresh classifiers and compares alternative forwarding choices during training.

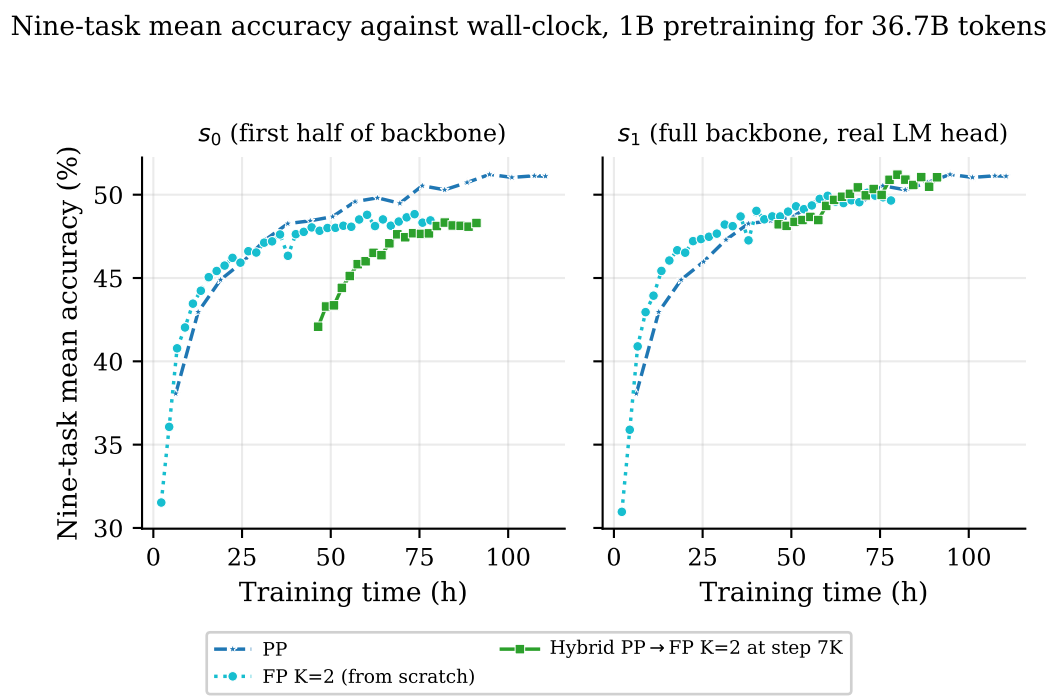


Figure 10: Hybrid schedules on 1B pretraining for 36.7B tokens, with training time computed using each phase's throughput. The PP→FP hybrid switches at step 7K and reaches its peak of 51.20 in 15.5% less time than PP first reaches the same target. The mirror FP→PP hybrid trains locally to step 7K and then continues with PP. Its evaluations at steps 17,250 and 17,500 appear in the full-network (s_1) panel.

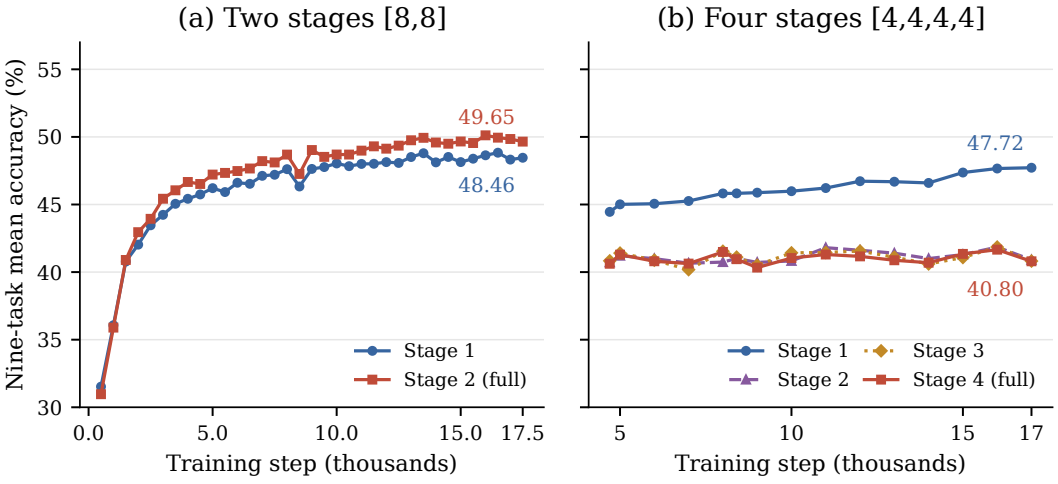


Figure 11: Accuracy through every stage head during 1B pretraining. Each panel follows one run under its respective recipe (Appendices H.2 and H.3). Curves show all checkpoints evaluated through every head, without smoothing. The final predictor overtakes the first at two stages, whereas the later three predictors remain below the first at four stages.

The first-to-final gap also widens at eight-stage 8B, where the two predictors finish at 47.60 and 38.68 (Figure 4b). Table 12 summarizes the first and final predictors across these FP runs and two other local methods.

Table 11: Accuracy and adjacent-head differences at each run's last checkpoint evaluated through every head. Δ is the accuracy minus the preceding head's accuracy, in percentage points, computed before rounding. Intermediate heads include their trained auxiliary branches.

Stages	Step	Readout after stage	Accuracy	Δ
2	17,500	1	48.46	—
		2 (full model)	49.65	+1.19
4	17,000	1	47.72	—
		2	40.91	-6.81
		3	40.82	-0.09
		4 (full model)	40.80	-0.02

Table 12: Nine-task mean accuracy at the first and last checkpoints evaluated through both prediction paths. Each row follows one training run. The different step intervals reflect available paired evaluations, not matched training budgets across rows.

		First-stage predictor		Full model	
Setting	Steps	First	Last	First	Last
FP, 1B, two stages	500–17,500	31.53	48.46	30.97	49.65
FP, 1B, four stages	4,700–17,000	44.45	47.72	40.62	40.80
FP, 8B, eight stages	700–7,000	35.11	47.60	34.54	38.68
InfoPro, 1B, four stages	2,500–17,500	41.90	45.93	41.47	45.56
AugLocal, 1B, four stages	2,500–17,500	35.44	34.97	35.78	35.73

This divergence is not shared by every local method that loses accuracy at four stages (Table 12). InfoPro improves through both prediction paths, while AugLocal remains weak through both. The rising first-stage score is therefore a feature of these FP trajectories, not a necessary signature of thin-stage collapse.

D.2 Per-stage loss spread

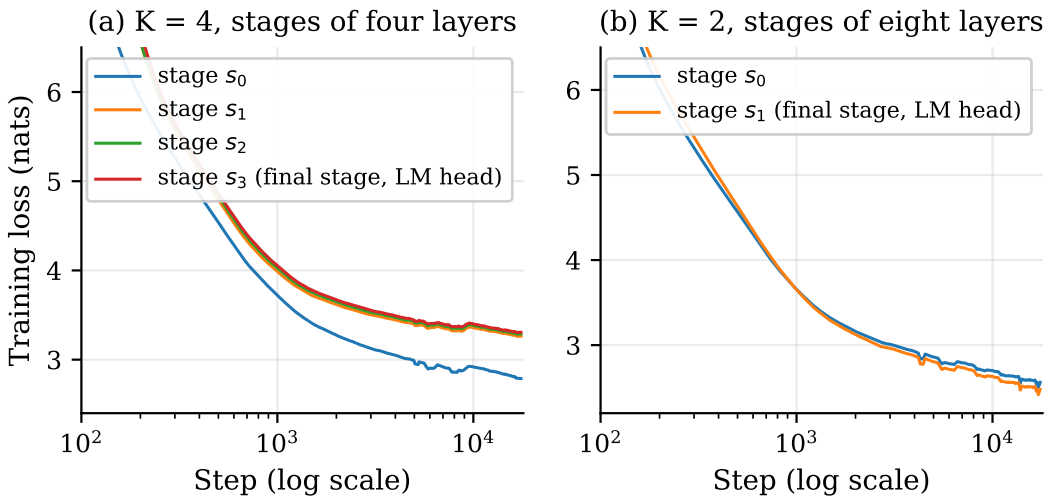


Figure 12: Per-stage training loss at (a) K=4 and (b) K=2 on 1B pretraining for 36.7B tokens. Averaged over steps 8K–10K, the losses span about 0.5 nats at four stages and 0.07 nats at two. Curves use a centered 25-sample moving average.

At four stages, the first stage reaches a lower local training loss than the later stages (Figure 12a). The mean losses over steps 8K–10K span about 0.5 nats, compared with 0.07 nats for two stages over the same interval. This separation is consistent with stages fitting their own objectives without an

end-to-end gradient to coordinate the network. The stage-thickness experiments in Section 4.5 test the effect of changing the boundaries directly.

D.3 AUXILIARY DESIGNS UNDER A SHORT PRETRAINING BUDGET

This comparison tests the auxiliary-block choice and the quality-throughput tradeoff among local-learning methods. Llama-3.2 1B (16 layers, 2048 hidden size, 32 heads) on FineWeb-Edu and OLMo-2 1B (Team OLMo et al., 2025) on Wikipedia are pretrained from scratch for 20,000 optimizer steps (about 300M tokens) at $K{=}4$ on 4×A100-40GB GPUs at sequence length 2048. FP, DGL, AugLocal and InfoPro are compared against PP with four stages. Quality is WikiText word perplexity (Merity et al., 2017) and throughput is measured on the same hardware. Each method runs at its own tuned learning rate, listed with the matched-rate comparison in Appendix H.1.

Table 13: $K{=}4$ local-learning comparison on 20K-step (about 300M-token) LLM pretraining. Quality is WikiText word perplexity, lower is better. Throughput is measured on 4×A100-40GB, higher is better.

	Word perplexity ↓		Throughput (tokens/s) ↑	
Method	Llama-3.2	OLMo-2	Llama-3.2	OLMo-2
PP	204.0	60.1	35,700	33,500
FP	174.0	60.4	**44,492**	**41,431**
FP (no auxiliary block)	208.9	71.5	—	—
DGL	196.8	70.5	42,282	36,600
AugLocal	**172.9**	**59.4**	32,631	27,710
InfoPro	226.2	81.5	42,362	37,100

Removing the block. FP reaches 174.0 word perplexity on Llama and 60.4 on OLMo-2, compared with 208.9 and 71.5 without the auxiliary block (Table 13). Both variants retain the same inference architecture.

Comparing auxiliary designs. FP combines lower word perplexity and higher A100 throughput than DGL and InfoPro on both architectures (Table 13). AugLocal improves word perplexity to 172.9 and 59.4, compared with 174.0 and 60.4 for FP, at throughput below PP. FP processes 36% and 50% more tokens per second than AugLocal on Llama and OLMo-2, respectively. The longer-budget comparison in Table 4 evaluates all four local methods and shows that their accuracy ranking changes.

D.4 AUXILIARY-BLOCK EFFECTS ACROSS REGIMES

Removing the block retains each stage's prediction head and local objective. Table 14 collects this direct comparison from the ViT-Large and BERT-Large results, with no distillation in either variant. The block's fine-tuning effect is task-dependent, ranging from -0.23 to $+1.49$ points across these ten comparisons. The largest fine-tuning improvement is $+1.49$ points on ViT-Large CIFAR-100 at four stages. The block improves AG News accuracy but lowers IMDB accuracy at both stage counts.

Table 14: Effect of adding the auxiliary block on classification fine-tuning accuracy, in percentage points. Positive values favor FP with the block. Differences use the three-seed mean accuracies in Tables 16 and 17. Both variants use the local task objective without distillation.

Model	Dataset	$K=2$	$K=4$
ViT-Large	CIFAR-100	$+0.20$	$+1.49$
ViT-Large	Flowers-102	$+0.04$	$+0.02$
ViT-Large	Pets	$+0.04$	-0.08
BERT-Large	IMDB	-0.23	-0.14
BERT-Large	AG News	$+0.24$	$+0.32$

E FINE-TUNING DETAILS

ViT-Large and BERT-Large fine-tuning results are mean test accuracies over three seeds. All FP results in this section use local supervision without inter-stage distillation. Their auxiliary block is one additional Transformer layer of the backbone's type, followed by the task classifier. DGL, AugLocal and InfoPro use the best evaluated epoch.[2] No validation split was held out.

Fine-tuning at four stages. FP has the highest reported local-method mean accuracy on CIFAR-100 and AG News (Table 15). DGL leads on Flowers-102, the no-block ablation on Pets and IMDB. Across the five tasks, FP is never more than 0.18 points below the best of DGL, InfoPro and AugLocal. The block's benefit is task-dependent, as the direct ablation in Appendix D.4 shows.

Table 15: Fine-tuning accuracy at $K=4$. FP uses the auxiliary block, and its no-block ablation is reported separately. Values are mean test accuracies over three seeds.

(a) ViT-Large

Method	CIFAR-100	Flowers-102	Pets
PP	93.37	99.11	93.47
FP	93.30	99.06	94.00
FP (no block)	91.81	99.04	94.08
DGL	92.78	99.24	94.01
AugLocal	92.30	99.15	93.83
InfoPro	91.97	99.21	93.88

(b) BERT-Large

Method	IMDB	AG News
PP	88.86	92.67
FP	93.91	94.07
FP (no block)	94.05	93.75
DGL	93.99	94.02
AugLocal	93.94	93.98
InfoPro	94.03	94.02

E.1 VIT-LARGE FINE-TUNING

At four stages, adding the block changes CIFAR-100 accuracy by $+1.49$ points, compared with $+0.02$ on Flowers-102 and -0.08 on Pets (Table 14). At two stages, it improves accuracy on all three tasks, with smaller changes than on four-stage CIFAR-100.

Table 16: ViT-Large fine-tuning accuracy (%). Training budgets are 40 epochs for CIFAR-100, 200 for Flowers-102, and 50 for Oxford Pets. Panel A reports $K=4$ and Panel B reports $K=2$. Best result per dataset in **bold**.

Variant	CIFAR-100	Flowers-102	Oxford Pets
Panel A: $K=4$			
PP (reference)	**93.37**	99.11	93.47
AugLocal	92.30	99.15	93.83
DGL	92.78	**99.24**	94.01
InfoPro	91.97	99.21	93.88
FP (no auxiliary block)	91.81	99.04	**94.08**
FP	93.30	99.06	94.00
Panel B: $K=2$			
FP (no auxiliary block)	93.21	99.21	93.44
FP	**93.41**	**99.25**	**93.48**

E.2 BERT-LARGE FINE-TUNING

At four stages, the block changes IMDB accuracy by -0.14 points and AG News by $+0.32$ points (Table 14). The signs are the same at two stages.

The InfoPro adaptation replaces the spatial decoder used for images with a sequence-compatible reconstruction objective.

[2] InfoPro's CIFAR-100 and Flowers-102 results use the best checkpoints from runs that ended before the scheduled training budget.

Table 17: BERT-Large fine-tuning accuracy (%) on IMDB and AG News. Panel A: $K{=}4$ (5 epochs, 4×A100-40GB). Panel B: $K{=}2$ (4×A100-40GB). Best result per dataset in **bold**.

Variant	IMDB	AG News
Panel A: $K{=}4$		
PP (reference)	88.86	92.67
DGL	93.99	94.02
AugLocal	93.94	93.98
InfoPro	94.03	94.02
FP (no auxiliary block)	**94.05**	93.75
FP	93.91	**94.07**
Panel B: $K{=}2$		
FP (no auxiliary block)	**89.74**	92.51
FP	89.51	**92.75**

E.3 Two-stage fine-tuning throughput

At $K{=}2$, FP reaches $1.36\times$ PP throughput at batch 64, narrowing to $1.14\times$ at batch 256 (Table 18).

Table 18: ViT-Large CIFAR-100 fine-tuning throughput (samples/s) on $4\times$A100, 2 nodes.

Algorithm	Batch 64	Batch 128	Batch 256
Pipeline Parallel	111.9	124.3	131.5
Tensor Parallel	73.9	77.5	79.7
Hybrid Parallel	89.0	95.7	100.6
FluidPipe	**152.3**	**153.9**	**150.5**
Speedup vs PP	$1.36\times$	$1.24\times$	$1.14\times$

F 70B throughput detail

At 70B on 64×H200, FP sustains $+73\%$ steady-state throughput over PP with $K{=}8$ inter-node stages and TP$=8$ (Table 3).

The PP baseline ran for 1,000 steps and was stopped for cost.

F.1 Eight-stage PP throughput on H200

We sweep PP micro-batch sizes $\mathrm{mbs} \in \{2, 4, 8, 16\}$ with and without activation recomputation on 64×H200-143GB GPUs. The model uses eight stages, TP$=8$ and sequence length 4096. Nanotron's 1F1B engine deadlocked at this scale, so we use AFAB. The micro-batch and accumulation settings are listed in Table 19. The best configuration, $\mathrm{mbs}{=}8$ with activation recompute, achieves 35,000 tokens/s on a 10-step sweep at 74 GB peak memory per device (Table 19). The full-run steady-state PP throughput is approximately 31,000 tokens/s once pipeline warmup completes (Table 3). Without recomputation the sweep identifies mbs$=4$, gradient accumulation$=8$ at 33,000 tokens/s and 123 GB peak memory. These configurations vary micro-batch size and accumulation as well as recomputation. The main comparison uses the selected configuration's full-run throughput.

F.2 Eight-stage FP throughput on H200

Without activation recompute, FP plateaus at ≈52,000 tokens/s (batch $\in \{8, 10, 12\}$). With recompute, throughput flattens at ≈34,000 tokens/s, a 35% penalty (both panels in Table 20). Although batch 10 and batch 12 are the highest 10-step sweep rows, batch 8 is within 0.6% of them and matches PP's production global batch when paired with gradient accumulation 32. The production comparison therefore uses mbs$=8$, grad. accum.$=32$, no recompute, and a training-only auxiliary block. This

Table 19: PP micro-batch and recomputation sweep on 64×H200. Selected configuration in bold. Sweep numbers slightly overestimate full-run throughput (35K vs full-run steady ≈31K, Table 3).

mbs	recompute	grad. accum.	tokens/s	Peak mem
8	**yes**	**32**	**35,000**	**74 GB**
4	no	8	33,000	123 GB
2	no	8	31,500	73 GB
16	yes	8	25,000	91 GB

matches PP's ≈1.05M tokens/step while using the no-recompute FP setting. Over the full 8,330-step production run, FP reaches 53,500 tokens/s steady-state, a $+73\%$ advantage over PP (Table 3).

Table 20: FP stage-local batch sweep on 64×H200 under both recomputation settings (K=8, TP=8). Without recompute, throughput plateaus at ≈52K tokens/s (batch $\in \{8, 10, 12\}$) then drops sharply under memory pressure. With recompute, throughput flattens at ≈34K tokens/s regardless of batch. The **bold** row is the production-comparison row used to match PP's global batch. Batch 10 and 12 are the highest short-sweep rows and are not used for the main comparison.

Batch	Recompute	Stage-0 tokens/s	Stage-0 peak mem
No activation recomputation			
24	No	OOM	—
20	No	OOM	—
16	No	37,400	102 GB
14	No	39,500	92 GB
12	No	52,000	81 GB
10	No	52,000	71 GB
8	**No**	**51,700**	**61 GB**
4	No	44,000	50 GB
With activation recomputation			
24	Yes	27,600	75 GB
16	Yes	34,200	58 GB
8	Yes	34,000	42 GB

G FLUIDPIPE ALGORITHM

Figure 3 shows the two feature paths used by the procedure below. Each stage loads the target labels locally, using the same dataset seed and example order within a data-parallel replica. Labels therefore need not be sent across stage boundaries.

Algorithm 1 FP stage-s procedure ($s \in \{1, \ldots, K\}$).

Input: stage s, stages K, dataset $\mathcal{D}$, epochs E
Initialize f_s and, for $s < K$, auxiliary block a_s and local head g_s
for epoch $\leftarrow 1$ **to** E **do**
  for each mini-batch from $\mathcal{D}$, split into M micro-batches **do**
    Clear local parameter gradients
    for each micro-batch (x, y), in data order **do**
      if $s = 1$ **then** $h_{s-1} \leftarrow x$
      else Receive a detached h_{s-1} from Stage $s-1$
      $h_s \leftarrow f_s(h_{s-1})$
      if $s < K$ **then**
        Send a detached copy of h_s to Stage $s+1$
        $\hat{y}_s \leftarrow g_s(a_s(h_s))$
      else
        $\hat{y}_s \leftarrow \mathrm{head}_{\mathrm{task}}(h_s)$
      $\mathcal{L} \leftarrow \mathcal{L}_{\text{task}}(y, \hat{y}_s)/M$ (Equation (2))
      Accumulate gradients through the local prediction path only
    Update f_s, a_s, g_s if $s < K$, otherwise f_K and the original head

H DETAILED EXPERIMENTAL RECIPES

The following recipes specify the training configurations used in the experiments. Here K is the number of stages, and TP and DP are the tensor- and data-parallel degrees. Micro-batch size (mbs) and gradient accumulation (accum) are given per replica. All LLM pretraining runs use AdamW with β_1=0.9, β_2=0.95, ε=10^{-8}, weight_decay=0.1, gradient clipping at 1.0, bf16 mixed-precision activations, fp32 gradient accumulation, and ZeRO stage 0. Deviations from this default are listed per recipe. Both PP and FP use separate input-embedding and output-classifier weights in the main two-stage comparisons at 1B and 8B. This avoids embedding-gradient synchronization between the first and final stages. Both comparisons use the `meta-llama/Llama-3.3-70B-Instruct` tokenizer with pre-tokenized FineWeb-Edu data.

H.1 K=4 1B 20K-STEP LOCAL-LEARNING BASELINE COMPARISON

The LLM adaptations instantiate the following auxiliary architectures. DGL uses an MLP head with hidden dimension 2048. AugLocal uses a pyramidal auxiliary network of depth 3 with up to 3 augmented blocks per local objective. InfoPro uses a reconstruction head with projection dimension 512 and an I_{xx} weight of 0.1.

The rates in Table 21 are each method's best rate from the sweep in Table 22. FP also has lower perplexity at a common learning rate on Llama-3.2 1B. At 3.0×10^{-4}, FP reaches 188.5 word perplexity against 227.1 for PP. On OLMo-2 1B, FP, DGL, AugLocal and PP already share the rate 1.5×10^{-4}.

H.2 K=2 1B 36.7B-TOKEN PRETRAINING

Table 23 specifies the two-stage run used for the main 1B quality comparison.

Table 21: Recipe for the four-stage local-learning baseline comparison reported in Table 13. All methods train for 20,000 updates, consuming 327.68M tokens on Llama and 245.76M tokens on OLMo-2.

Setting	Llama-3.2 1B (FineWeb-Edu)	OLMo-2 1B (Wikipedia)
Layers / hidden / heads / KV-heads	16 / 2048 / 32 / 8	16 / 2048 / 16 / 16
Vocab size	128,256	100,278
Tokenizer	`meta-llama/Llama-3.2-1B`	`allenai/OLMo-2-0425-1B`
Sequence length	2048	2048
Optimizer	AdamW (default)	AdamW (default)
LR (FP)	4.0×10^{-4}	1.5×10^{-4}
LR (DGL, AugLocal)	3.0×10^{-4}	1.5×10^{-4}
LR (InfoPro)	4.0×10^{-4}	4.0×10^{-4}
LR (PP=4)	2.0×10^{-4}	1.5×10^{-4}
Schedule	cosine, warmup 500, decay 18000→20000, min_lr=0	(same)
mbs / accum (FP)	2 / 4	2 / 3
mbs / accum (PP=4)	2 / 4	2 / 3
Train steps	20,000	20,000
Total tokens (all methods)	327.68M	245.76M
Local stages	4	4
FP layer splits	[4, 4, 4, 4] + auxiliary block	[4, 4, 4, 4] + auxiliary block
Hardware	4×A100-40GB	4×A100-40GB

Table 22: Learning-rate sweep for four-stage, 20K-step pretraining, evaluated by WikiText word perplexity. Selected rates are in Table 21. The Llama rows also compare PP and FP at a common learning rate.

Model	Method	LR	Word perplexity
Llama-3.2 1B	PP	2.0×10^{-4}	204.0
Llama-3.2 1B	PP	3.0×10^{-4}	227.1
Llama-3.2 1B	FP	3.0×10^{-4}	188.5
Llama-3.2 1B	FP	4.0×10^{-4}	174.0
Llama-3.2 1B	FP (no auxiliary block)	3.0×10^{-4}	226.3
Llama-3.2 1B	FP (no auxiliary block)	4.0×10^{-4}	208.9
Llama-3.2 1B	DGL	3.0×10^{-4}	196.8
Llama-3.2 1B	AugLocal	3.0×10^{-4}	172.9
Llama-3.2 1B	InfoPro	4.0×10^{-4}	226.2
OLMo-2 1B	PP	1.0×10^{-4}	66.51
OLMo-2 1B	PP	1.5×10^{-4}	60.08
OLMo-2 1B	PP	3.0×10^{-4}	188.15
OLMo-2 1B	FP	1.5×10^{-4}	60.39
OLMo-2 1B	FP	2.0×10^{-4}	61.67
OLMo-2 1B	FP (no auxiliary block)	1.0×10^{-4}	75.44
OLMo-2 1B	FP (no auxiliary block)	1.5×10^{-4}	71.5
OLMo-2 1B	FP (no auxiliary block)	2.0×10^{-4}	74.47
OLMo-2 1B	FP (no auxiliary block)	3.0×10^{-4}	173.49
OLMo-2 1B	DGL	1.5×10^{-4}	70.5
OLMo-2 1B	DGL	3.0×10^{-4}	73.5
OLMo-2 1B	AugLocal	1.5×10^{-4}	59.4
OLMo-2 1B	AugLocal	3.0×10^{-4}	63.9
OLMo-2 1B	InfoPro	4.0×10^{-4}	81.5

Table 23: Recipe for the $K{=}2$ 1B 36.7B-token FP pretraining run (Appendix C.3).

Setting	Value
Model	Llama-3.2 1B (16 layers, hidden 2048, 32 heads, 8 KV heads)
Vocab / sequence length	128,256 / 2048
Tokenizer	`meta-llama/Llama-3.3-70B-Instruct`
Dataset	FineWeb-Edu, 35B-token pre-tokenized snapshot
Optimizer	AdamW (default)
LR	5.0×10^{-4}
Schedule	cosine, warmup 1000, decay 1000→17500, min_lr=5.0×10^{-5}
mbs / accum / DP	32 / 32 / 1
Tokens per step	2,097,152
Train steps	17,500
Total tokens	36,700,160,000
Local stages	2
FP layer splits	[8, 8] + auxiliary block
Hardware	1×8-GPU H200-143GB node

H.3 K=4 1B 36.7B-TOKEN PRETRAINING

Table 24: Training configuration for the K=4 Llama-3.2 1B pretraining run in Section 4.4.

Setting	Value
Dataset	FineWeb-Edu (pre-tokenized)
Backbone layers / hidden width	16 / 2,048
Vocabulary / sequence length	128,256 / 2,048
Tokenizer identifier	`meta-llama/Llama-3.3-70B-Instruct`
Optimizer	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, $\varepsilon = 1.0\times10^{-8}$)
Weight decay / gradient clipping	0.1 / 1.0
Learning rate / minimum	6.0×10^{-4} / 6.0×10^{-5}
Schedule / warmup steps	cosine / 1,000
Decay start / duration (steps)	1,000 / 16,500
mbs / accumulation / DP	16 / 64 / 1
Tokens per step / training steps	2,097,152 / 17,500
Total tokens	36,700,160,000
Local stages	4
Backbone partition	[4, 4, 4, 4]
Auxiliary block	enabled

The PP baseline uses the same model, dataset, sequence length, tokens per step and total step count, with learning rate 3.0×10^{-4}.

H.4 HYBRID SCHEDULES AT K=2

The PP→FP hybrid is identical to Appendix H.2 except that optimizer state is loaded from the reference PP run's checkpoint at step 7,000 and training continues to step 17,500 in the FP regime. The mirror hybrid trains FP K=2 from scratch with the reference run's learning-rate schedule to step 7,000, then continues with PP at a learning rate of 0.75×10^{-4} to step 17,500.

H.5 8B THROUGHPUT SWEEP

The 8B sweep configurations are throughput measurements, not learning runs. Each configuration uses Llama-3.1-8B (32 layers, hidden 4096, 32 heads, 8 KV heads, intermediate 14336, vocab 128,256), sequence length 2048, AdamW, and a deliberately flat learning-rate schedule (warmup 5, decay_steps 1, lr=10^{-4}) over 80 training steps. The reported tokens/s value is the steady-state per-step throughput. The sweep grid is $K \in \{2, 4, 8\}$, TP $\in \{1, 2, 4\}$, mbs $\in \{1, 2, 4, 8, 16, 32, 64\}$, with optional flags for the auxiliary block, the aux_head_proj_dim=0 projection variant, chunked cross-entropy (Appendix B.6), and activation recompute. PP configurations use Nanotron's pp=K with no FP. FP configurations use num_fp_stages=K with even layer splits and the matching (K, TP). Hardware: 2-node 8-GPU H100-80GB (4 GPUs per node, 200 Gbps cross-node interconnect). The selected configurations are tabulated in Table 7.

H.6 70B THROUGHPUT

Table 25 gives the configurations for the throughput comparison in Appendix F.

H.7 VIT-LARGE FINE-TUNING

The vision fine-tuning runs use the original FluidPipe codebase (Hydra-based) rather than the Nanotron extension.

Backbone: ViT-Large/16 (`vit_large_patch16_224.orig_in21k` from `timm`), image resolution 224×224. Optimizer (all three datasets): SGD, lr=0.03, momentum=0.9, weight_decay=0. Effective batch size 2,048 (per-device 512 with gradient accumulation 4). Three random seeds per configuration. Checkpoint selection is described in Appendix E.

Table 25: 70B throughput recipes for FP $K{=}8$ and the PP $K{=}8$ baseline reported in Section 4.2 and Appendices F.1 and F.2.

Setting	FP $K{=}8$	PP $K{=}8$ baseline
Model	Llama-3.3-70B (80 layers, hidden 8192, 64 heads, 8 KV heads, intermediate 28672)	(same)
Vocab / sequence length	128,256 / 4096	128,256 / 4096
Tokenizer / dataset	`meta-llama/Llama-3.3-70B-Instruct` / FineWeb-Edu 15B snapshot	(same)
Optimizer	AdamW (default)	AdamW (default)
LR	1.5×10^{-4}	1.5×10^{-4}
Schedule	cosine, warmup 2000, decay 80000→100000, min_lr=1.5×10^{-5}	(same)
mbs / accum	8 / 32	8 / 32
Tokens per step	1,048,576	1,048,576
Parallelism (Nanotron view, per TP-group)	dp 1 / pp 1 / tp 8	dp 1 / pp 8 / tp 8
FP num_fp_stages	8 (the FluidPipe launcher assembles 8 such TP-groups, one per FP stage)	—
FP auxiliary block / layernorm freeze	enabled / enabled	—
Total GPUs	64 (8 nodes × 8 H200-143GB)	64 (same layout)

Table 26: ViT-Large fine-tuning settings at $K{=}4$ (Appendix E.1). CIFAR-100 uses 4×A100, and Flowers-102 and Pets use 2×V100.

Dataset	Budget (epochs)	Sweep steps	Warmup steps	Eval cadence
CIFAR-100	40	10,000	500	every 2 epochs
Flowers-102	200	800	500	every 5 epochs
Oxford Pets	50	350	500	every 5 epochs

The splitter partitions the stem, the 24 Transformer blocks, and the final normalization and classification module. The distributions [7, 6, 7, 6] at $K{=}4$ and [14, 12] at $K{=}2$ count those modules, not only Transformer blocks.

H.8 BERT-LARGE FINE-TUNING

Backbone: `bert-large-uncased` (24 layers, hidden 1024, 16 heads, 336M parameters), maximum sequence length 512. Three random seeds per configuration. Checkpoint selection follows the same protocol as ViT-Large (Appendix E).

Table 27: BERT-Large fine-tuning per-dataset settings (Appendix E.2). Both datasets run on 4×A100-40GB.

Setting	IMDB	AG News
Optimizer	AdamW ($\beta_1{=}0.9$, $\beta_2{=}0.999$, $\varepsilon{=}10^{-8}$, wd=0.01)	(same)
LR	2.0×10^{-5}	3.0×10^{-5}
Scheduler	warmup-then-constant	warmup-then-linear-decay
Warmup ratio	0.06	0.06
Effective batch	2,048	2,048
Train epochs	5	5
Sequence length	512	512

Layer splits are [6, 6, 6, 6] at $K{=}4$ and [12, 12] at $K{=}2$.

H.9 LLAMA-2-7B METAMATHQA SFT

Table 28: Recipe for the 7B math SFT run reported in Appendix C.5.

Setting	Value
Model	Llama-2-7B (32 layers, hidden 4096, 32 heads, intermediate 11008, vocab 32,000)
Initial weights	`meta-llama/Llama-2-7b-hf`
Sequence length	512
Tokenizer	`meta-llama/Llama-2-7b-hf`
Dataset	`meta-math/MetaMathQA`, completion-only loss, chat template `metamath`
Optimizer	AdamW fused (β_1=0.9, β_2=0.999, ε=10^{-8}, wd=0, clip 1.0), fp32 grad accum, ZeRO 0
LR	4.0×10^{-5}
Schedule	cosine, warmup 277, decay starting 277, min_lr=4.0×10^{-6}
mbs / accum	16 / 8
Effective batch	128 sequences ($\sim$ 65,536 tokens) per step
Train steps	9,257
Local stages	2
FP layer splits	[16, 16] + auxiliary block
Random seed	42
Hardware	2-node, 8-GPU H100-80GB cluster (same hardware as the 8B throughput sweep)
Evaluation:	
GSM8K	`openai/gsm8k` (main, test), 0-shot CoT, generate-until, exact match, max gen 512
MATH	`hendrycks/math` 7-subtask group, weighted exact-match by subtask size
Eval harness	`lm-eval-harness` v0.4 default settings

H.10 Stage-thickness series at 1B, purely local series at 8B, and the vision runs

The twelve 1B partitions of Section 4.5 share the recipe of Table 24 with a learning rate of 6×10^{-4}, the rate of the four-stage local baselines, and differ only in the layer split. We deliberately vary the compute balance to study the learning effect of boundary placement, not to recommend unequal partitions for distributed training. Each run is evaluated on its own last stage, the full network with the language-model head. Table 29 lists every partition.

Table 29: The 1B stage-thickness series. Nine-task mean accuracy at step 17,500, one seed, each run evaluated on its own last stage. The PP reference is 51.11.

Partition	Stages	Layers in the first stage	Accuracy
[1,4,4,7]	4	1	34.84
[2,4,4,6]	4	2	34.48
[3,4,4,5]	4	3	37.45
[4,4,4,4]	4	4	40.71
[6,4,4,2]	4	6	45.26
[8,3,3,2]	4	8	46.83
[10,2,2,2]	4	10	49.52
[2,14]	2	2	35.76
[5,5,6]	3	5	44.25
[8,6,2]	3	8	47.68
[8,2,6]	3	8	47.63
[2]×8	8	2	34.67

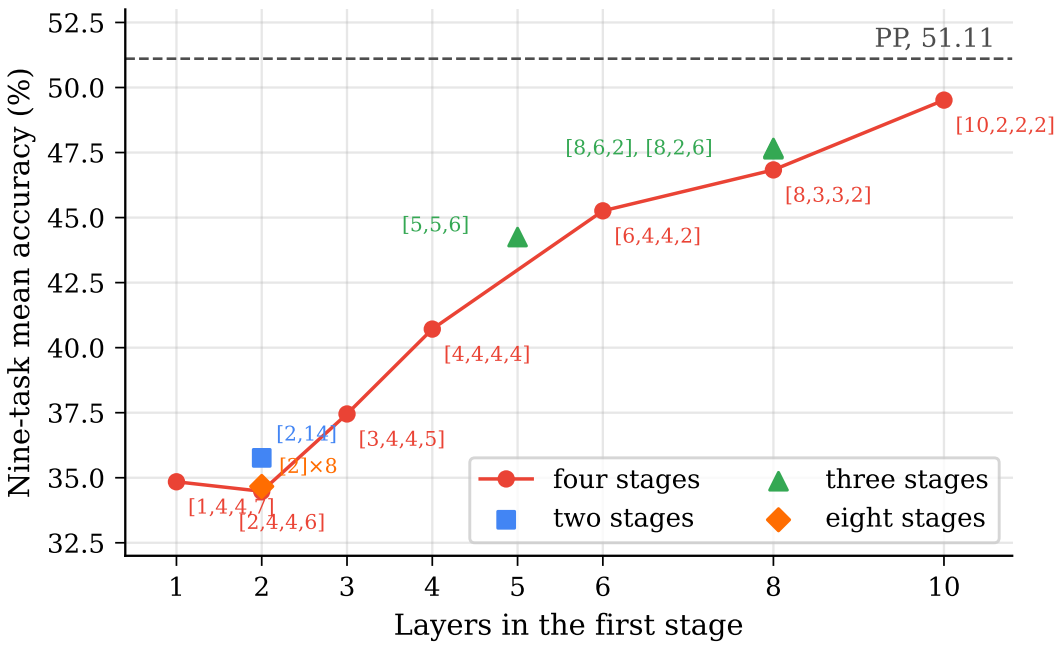


Figure 13: The twelve partitions in the 1B fixed-rate series, evaluated at step 17,500 after 36.7B tokens. Labels give layer counts in forward order. The dashed line is the PP reference, 51.11. The seven four-stage partitions also appear in Figure 2b.

The 8B purely local series of Section 4.4 trains Llama-3.1 8B (32 layers, hidden 4096, 32 heads, 8 KV heads, vocabulary 128,256) on FineWeb-Edu for 7,000 steps of 2,097,152 tokens at sequence length 2048, with AdamW at a learning rate of 6×10^{-4}, warmup 1,000 and cosine decay over 16,500 steps. The PP reference at 8B uses the same schedule.

Partitions are [16,16], [10,11,11], [8,8,8,8] and [4]×8, each evaluated on its own last stage. For the central two-stage comparison, both methods use bfloat16 model weights and float32 gradient accumulation. Both use AdamW with β_1=0.9, β_2=0.95, ε=10^{-8}, weight decay 0.1, gradient clipping at 1.0 and seed 42. The minimum learning rate is 6×10^{-5}.

The vision runs of Section 4.6 and Appendix I train ResNet-50, ResNet-101 and ResNet-152 with SGD, momentum 0.9, 500 warmup steps and cosine decay. On CIFAR-100 they train for 200 epochs at batch 256, learning rate 0.2 and weight decay 5×10^{-4}, on CINIC-10 for 300 epochs at learning rate 0.4 and weight decay 1×10^{-4}. The stem is 3×3 with stride 1, the augmentation is crop-and-pad plus flip. Seeds are 42, 43 and 44, and single-seed configurations are marked where they appear. All K stages run in one process on one device, so these runs do not measure distributed-training throughput.

I VISION COLLAPSE DETAIL

This lower-cost setting supports the controlled analyses in Section 4.6. The stage-count sweep checks that thin-stage collapse occurs in image classification before the component comparisons examine the auxiliary block.

CIFAR-100 stage-count series. Table 30 gives every seed underlying Figure 14. Each model is divided into approximately equal stages, placing the remaining splitter units in the last stages. The units count the stem, residual blocks and final classifier. The ResNet-101 two-stage partition is [17,18]. The unequal two-stage partitions below are separate experiments.

CINIC-10. Collapse on CINIC-10 appears at more stages and remains smaller than on CIFAR-100. The end-to-end references are 88.48, 88.43 and 88.65 for ResNet-50, 101 and 152, averaged over

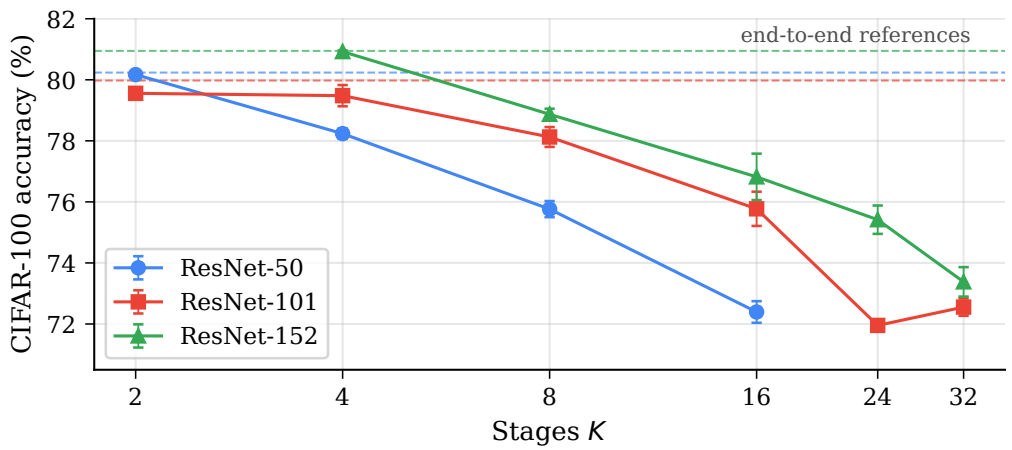


Figure 14: Stage-count sweep on CIFAR-100 for three ResNets trained purely locally for 200 epochs. Dashed lines are the end-to-end references. Markers are three-seed means and bars sample standard deviations (Table 30).

Table 30: CIFAR-100 test accuracy after 200 epochs for the stage-count series. BP is end-to-end backpropagation. Numbered entries use that many local stages. Entries are final-checkpoint accuracies, with means and sample standard deviations over three seeds.

Model	Stages	Seed 42	Seed 43	Seed 44	Mean ± s.d.
ResNet-50	BP	80.39	79.97	80.35	80.24 ± 0.23
ResNet-50	2	80.28	80.13	80.10	80.17 ± 0.10
ResNet-50	4	78.08	78.21	78.42	78.24 ± 0.17
ResNet-50	8	75.46	75.94	75.89	75.76 ± 0.26
ResNet-50	16	72.28	72.11	72.79	72.39 ± 0.35
ResNet-101	BP	79.94	79.78	80.21	79.98 ± 0.22
ResNet-101	2	79.69	79.60	79.38	79.56 ± 0.16
ResNet-101	4	79.53	79.80	79.11	79.48 ± 0.35
ResNet-101	8	78.44	77.79	78.15	78.13 ± 0.33
ResNet-101	16	76.26	75.90	75.16	75.77 ± 0.56
ResNet-101	24	72.01	71.92	71.93	71.95 ± 0.05
ResNet-101	32	72.88	72.42	72.36	72.55 ± 0.28
ResNet-152	BP	80.88	80.91	81.05	80.95 ± 0.09
ResNet-152	4	80.88	80.94	80.95	80.92 ± 0.04
ResNet-152	8	78.95	79.00	78.66	78.87 ± 0.18
ResNet-152	16	77.59	76.80	76.07	76.82 ± 0.76
ResNet-152	24	75.88	74.95	75.42	75.42 ± 0.47
ResNet-152	32	72.84	73.53	73.77	73.38 ± 0.48

three seeds. At sixteen stages, the respective accuracy losses are 5.32, 3.11 and 1.81 points. At 32 stages, ResNet-101 and ResNet-152 lose 5.61 and 5.03 points. ResNet-50 loses only 0.28 points at two stages and 2.98 at eight, while ResNet-101 and ResNet-152 slightly exceed their references at four stages.

Deep supervision in the stage-count sweep. Keeping the auxiliary losses while allowing their gradients to reach earlier stages is often called deep supervision. These configurations have higher accuracy than local FP at each tested partition (Table 31). The controlled comparison below matches initialization and clipping across variants to separate the block's effect from gradient connectivity.

Controlled auxiliary-block and gradient-connectivity study. The controlled block comparisons use separate training runs from the stage-count sweep, with backbone initialization matched across variants. We independently vary whether each auxiliary classifier has a residual block and whether losses can backpropagate across stage boundaries. Every non-final stage retains its auxiliary classifier

Table 31: ResNet-152 stage-count sweep on CIFAR-100 after 200 epochs, final test top-1 accuracy, three-seed means. BP and connected auxiliary training use global clipping, while local FP uses per-stage clipping, all at a maximum norm of 1.0. The auxiliary block is present in both auxiliary-training configurations.

Stages	BP	Connected auxiliaries	Local FP
4	80.95	81.49	80.92
16	80.95	79.91	76.82
32	80.95	77.10	73.38

and classification objective, including when the block is removed. All runs use the CIFAR-100 recipe above and the final classifier at epoch 200. For each seed, we restore identical backbone weights and batch-normalization statistics before constructing the variants. The auxiliary branches use a separate random-number stream, so changing a branch does not change the starting backbone or the data augmentation sequence. This is the paired initialization used in Section 4.6. After backpropagation, we cap each stage's gradient norm at 1.0 independently, including its auxiliary branch in that stage's parameter group. This per-stage clipping rule is the same whether gradients cross boundaries or not. The sixteen-stage partition is twelve stages of three splitter units followed by four stages of four units, and the four-stage partition is [13,13,13,13]. These units count the stem, residual blocks and final classifier.

The local block effect is 7.79 ± 0.63 points at sixteen stages and 2.07 ± 0.26 at four (Table 33). At sixteen stages, its effect under connected gradients is 2.72 ± 0.11 points. The difference between these block effects, computed within each seed before averaging, is 5.07 ± 0.54 points. Equivalently, the loss from cutting gradients changes from 9.03 ± 0.66 points without the block to 3.95 ± 0.40 with it. Gradient isolation and auxiliary design interact rather than contributing independent shares of the accuracy loss.

Removing all auxiliary objectives leaves backpropagation from the final classifier alone, with the same partition and per-stage clipping rule. This reference reaches $80.49 \pm 0.35\%$ accuracy, versus $81.00 \pm 0.32\%$ with connected auxiliary training. The paired gain from adding the auxiliaries is 0.50 ± 0.49 points (Table 33).

Table 32: The block's benefit depends on gradient connectivity. ResNet-152 on CIFAR-100, sixteen stages, final accuracy after 200 epochs. Entries are mean $\pm$ sample standard deviation over three paired seeds, with per-stage clipping in all conditions. Block effects are computed within each seed.

Cross-stage gradients	No block	FP block	Block effect
Disconnected	69.25 ± 0.37	77.04 ± 0.37	7.79 ± 0.63
Connected	78.28 ± 0.30	81.00 ± 0.32	2.72 ± 0.11

Table 33: Controlled ResNet-152 auxiliary comparisons on CIFAR-100 after 200 epochs. All conditions start from identical backbones within each seed and clip each stage's gradient norm independently. Entries give each seed's final accuracy and the three-seed mean $\pm$ sample standard deviation. The final-loss-only reference retains the same partition and clipping rule without auxiliary objectives.

Stages	Condition	Seed 42	Seed 43	Seed 44	Mean $\pm$ s.d.
16	No block, local	68.97	69.11	69.67	69.25 ± 0.37
16	FP, local	77.46	76.74	76.93	77.04 ± 0.37
4	No block, local	78.85	78.43	78.78	78.69 ± 0.23
4	FP, local	80.78	80.80	80.70	80.76 ± 0.05
16	Cost-matched pointwise branch, local	73.54	73.13	72.96	73.21 ± 0.30
16	Shallow AugLocal branch, local	79.27	79.33	79.48	79.36 ± 0.11
16	No block, connected	78.36	78.53	77.94	78.28 ± 0.30
16	FP, connected	81.20	81.16	80.63	81.00 ± 0.32
16	Final loss only, per-stage clipping	80.82	80.12	80.54	80.49 ± 0.35

The shallow AugLocal construction reaches 79.36 ± 0.11, exceeding FP by 2.32 ± 0.44 points. It takes the architecture of the backbone's final residual block, expanding the auxiliary width and adding downsampling, whereas FP preserves the stage's channel width. The comparison holds backbone initialization, stage partitions and the auxiliary-block initialization policy fixed. It tests the shallow AugLocal construction rather than reproducing its full training recipe. The equal-computation comparison replaces FP's block with two 1×1 (pointwise) convolutions, each operating on the feature channels at one spatial position. Batch normalization and a ReLU separate the convolutions. The second convolution is followed by batch normalization, addition of the branch input, and a ReLU. We choose the hidden width to match the forward multiply-accumulate count of FP's block plus classifier, counting convolution and linear-layer operations on both sides. This branch reaches 73.21 ± 0.30 against 77.04 ± 0.37 for FP, a paired gap of 3.83 ± 0.20 points.

What changes in the forwarded features? We use the final checkpoints of the paired sixteen-stage ResNet-152 models, with and without the auxiliary block. The first stage contains the stem and two residual blocks, and produces a $256 \times 32 \times 32$ feature map. We compare the no-block model's output ($N{:}h$), FP's output before its auxiliary block ($F{:}h$), and the same FP output after that block ($F{:}z$). The source weights and batch-normalization statistics remain fixed throughout fitting and evaluation.

Each representation receives a fresh linear classifier after global average pooling, or two fresh channel-preserving residual blocks followed by pooling and a linear classifier. All classifiers train for 50 epochs on the CIFAR-100 training split, with deterministic image preprocessing and no augmentation. They use SGD at learning rate 0.1, momentum 0.9, weight decay 10^{-4}, batch size 256 and cosine decay to zero. Initialization and example order are shared across representations, using seeds 4242 and 4243, respectively. The three backbone seeds are the replication units. We report test accuracy at the final checkpoint.

Raw feature scales differ enough to confound fitting with a common learning rate and weight decay. We therefore apply batch normalization without learned scale or offset before the first learnable layer of every fresh classifier. It follows pooling in the linear classifier and precedes the first convolution in the two-block classifier. Its statistics are learned only from training images and held fixed at test time. The original auxiliary heads are evaluated unchanged as a separate reference.

Table 34: Fresh-classifier accuracy on frozen first-stage features, and accuracy of the original auxiliary heads. All source models are ResNet-152 on CIFAR-100 after 200 epochs. Fresh classifiers train for 50 epochs. Means and sample standard deviations use the three backbone seeds.

Features	Classifier	Seed 42	Seed 43	Seed 44	Mean $\pm$ s.d.
No block	Linear	59.67	59.76	59.72	59.72 ± 0.05
No block	Two blocks	60.52	60.71	60.55	60.59 ± 0.10
No block	Original head	58.05	58.09	58.31	58.15 ± 0.14
FP, before block	Linear	61.70	61.03	61.06	61.26 ± 0.38
FP, before block	Two blocks	63.68	64.04	63.06	63.59 ± 0.50
FP, after block	Linear	64.26	64.38	63.96	64.20 ± 0.22
FP, after block	Two blocks	64.39	64.78	64.43	64.53 ± 0.21
FP, after block	Original head	63.20	63.19	63.22	63.20 ± 0.02

FP's forwarded features improve fresh-classifier accuracy by 1.55 ± 0.42 points for the linear classifier and 3.00 ± 0.43 for the two-block classifier (Table 35). The benefit extends beyond the original auxiliary predictor to features that a new learner can use. After the auxiliary block, accuracy rises by another 2.94 ± 0.40 points for the linear classifier and 0.94 ± 0.37 for the two-block classifier. Both tested learners favor post-block features over pre-block features.

These comparisons measure how effectively the classifiers use frozen features, not all information present in them. The following routing experiment tests whether the block's outputs also help when the later stages learn from them throughout training.

Table 35: Paired accuracy differences from Table 34, in percentage points. Each difference is computed within a backbone seed before averaging. Spreads are sample standard deviations.

Comparison	Seed 42	Seed 43	Seed 44	Mean $\pm$ s.d.
Before minus no block, linear	2.03	1.27	1.34	1.55 ± 0.42
Before minus no block, two blocks	3.16	3.33	2.51	3.00 ± 0.43
After minus before, linear	2.56	3.35	2.90	2.94 ± 0.40
After minus before, two blocks	0.71	0.74	1.37	0.94 ± 0.37

Forwarding before or after the auxiliary block. The frozen-feature results favor the block's outputs, but FP forwards the features before it. We test this choice by training the sixteen-stage ResNet-152 with three forwarding rules. Standard FP forwards the backbone features at every boundary. The alternatives forward the auxiliary block's output at the first boundary only, or at all fifteen boundaries. All three rules keep the same local classification objectives and cut every cross-stage gradient. They use the initialization files for seeds 42, 43 and 44 from the controlled block study, the same layer partition, and its 200-epoch training recipe. The standard FP checkpoints are reused, and the six alternatives train from the corresponding initial states.

Table 36: Forwarding choice on ResNet-152, CIFAR-100, sixteen local stages. Accuracy uses the final checkpoint after 200 epochs, with means and sample standard deviations over three seeds. The FP control was evaluated on 9,984 test images and the alternatives on 10,000. Retained blocks are auxiliary blocks required at inference, in addition to the original backbone.

Forwarded features	Seed 42	Seed 43	Seed 44	Mean $\pm$ s.d.	Retained blocks
Before block (FP)	77.46	76.74	76.93	77.04 ± 0.37	0
After first block	72.64	72.52	72.57	72.58 ± 0.06	1
After every block	71.03	71.41	70.83	71.09 ± 0.29	15

Forwarding after the first block loses 4.47 ± 0.31 accuracy points, and forwarding after every block loses 5.95 ± 0.56 relative to standard FP. These differences are computed within each seed before averaging, and standard FP is more accurate in all three seeds. The training models contain the same auxiliary blocks, but forwarding their outputs retains those blocks on the inference path. The tested alternatives thus require more inference computation without improving accuracy. The frozen-feature and full-training comparisons favor different representations, so the most predictive frozen representation is not necessarily the better input for later-stage training.

Position of a single cut. At two stages, ResNet-50 on CIFAR-100 (reference 80.24) with a first stage of 1, 2, 4, 6, 8 and 12 blocks scores 79.63, 79.18, 79.66, 79.10, 80.17 and 80.20, and ResNet-101 (reference 79.98) with a first stage of 3, 5, 9, 13, 17 and 26 blocks scores 79.69, 80.39, 80.49, 80.13, 79.69 and 79.18. The eight-block ResNet-50 split is a three-seed mean, the other values are single seeds. The largest cut-position spread is 1.31 points, on ResNet-101, with no monotone trend. In the seed-42 comparison, ResNet-50 with a one-block first stage scores 79.63 at two stages and 75.46 with eight thin stages at the same first-stage fraction. Thin stages throughout the network produce a much larger loss than a thin first stage alone.

Stage count against boundary fraction. At sixteen stages, accuracies span 4.43 points across the three ResNets. Using eight stages for ResNet-50, sixteen for ResNet-101 and twenty-four for ResNet-152 cuts 41.2–45.1% of the boundaries between splitter units and narrows the accuracy spread to 0.36 points (Table 30). This alignment is consistent with relative stage thickness mattering more than stage count alone. ResNet-101 at 32 stages scores above its 24-stage run with non-overlapping seed ranges, which is unexplained.

J Reproducibility statement

All LLM pretraining experiments use the Nanotron distributed training framework with the FP extension. Appendix H specifies the model, optimizer, schedule, batch size and parallelism settings

for each experiment family. Training and evaluation code, configurations and dataset-preparation instructions will be released upon acceptance.

The 1B quality suite includes the 36.7B-token from-scratch runs, the hybrid schedules, and the $K{=}4$ 36.7B-token runs. LLM downstream evaluations use the publicly available `lm-eval-harness` (Gao et al., 2021) framework (v0.4) with default settings. We report the unweighted mean of HellaSwag, ARC-Easy, ARC-Challenge, PIQA, WinoGrande, OpenbookQA, SciQ, LAMBADA, and BoolQ for the nine-task downstream metric. WikiText word perplexity is the metric of the 20K-step ($\sim$300M-token) $K{=}4$ pretraining comparisons.

The 7B math SFT run uses Llama-2-7B fine-tuned on MetaMathQA with the Nanotron FP extension and evaluates GSM8K and MATH. Vision and language classification fine-tuning experiments use ViT-Large on CIFAR-100, Flowers-102, and Oxford Pets, and BERT-Large on IMDB and AG News. Appendix E reports the dataset-specific schedules, hardware, seed conventions, and task-native accuracy metrics. The vision experiments of Section 4.6 and Appendix I use the original FluidPipe codebase, and their per-seed results will be released with it. Every local-learning run is evaluated through the full network and its original final task head. For LLM evaluation, the trained stages are arranged in sequence, the auxiliary blocks and local heads are removed, and the full network is exported to a standard HuggingFace checkpoint format.