CONFIG2WEIGHTS: GENERATING LLM INITIALIZATIONS FROM ARCHITECTURE CONFIGURATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training large language models (LLMs) from scratch remains prohibitively expensive, motivating growing interest in weight-space generative modeling as a route to direct parameter synthesis. Existing generative approaches, however, are largely confined to small vision models or model-specific, task-conditioned low-rank adapters, while heuristic model expansion methods rely on rigid, architecture-specific transformations. We propose Config2Weights, a generative initialization framework that learns from a collection of pretrained LLM weights to initialize new architectures conditioned solely on their architectural configuration. Config2Weights comprises two complementary variants: a direct configuration-to-weight mapping, and a conditional latent flow that operates in the latent space of a pretrained conditional variational autoencoder over model weights. We evaluate Config2Weights on LoRA adapters and full-model weights in both in-distribution and out-of-distribution settings, including unseen architectures and configurations. On held-out Llama models, adapters generated by Config2Weights improve final HellaSwag accuracy over random initialization by up to 12.35 points. On full models, Config2Weights improves early pre-training over random initialization and remains competitive with existing weight-expansion methods.

1 INTRODUCTION

The capabilities of modern large language models (LLMs) such as PaLM (Chowdhery et al., 2023) and LLaMA (Meta Llama Team, 2024) rest on large-scale pre-training, in which billions of parameters are optimized over massive text corpora at substantial computational cost. This makes new architectural variants expensive to develop: even modest changes to depth, width, attention structure, or feed-forward dimensions may require restarting pre-training from random initialization, a bottleneck for architecture exploration and model scaling.

One way to reduce this cost is to reuse pre-trained checkpoints. Prior work has explored knowledge distillation (Qin et al., 2022), model morphism (Wei et al., 2016), and function-preserving transformations (Chen et al., 2016), and recent expansion strategies (Samragh et al., 2024; Du et al., 2024) accelerate convergence by mapping a smaller parent model to a larger target through width expansion or layer stacking. These methods improve early optimization but rely on hand-designed structural correspondences, which limits them to targets that follow a rigid parent-child expansion pattern (Table 1).

In parallel, weight-space generative modeling treats model weights as a data modality and synthesizes parameters directly, through hypernetworks (Charakorn et al., 2025) or diffusion-based generators (Soro et al., 2025). Existing approaches, however, are demonstrated on small vision models (Peebles et al., 2022), modular parameter subsets (Schürholt et al., 2024), or model-specific low-rank adapters (Liu et al., 2026; Shao et al., 2025), and generate weights for a fixed architecture. Full transformer checkpoints are harder still: their parameters are high-dimensional, heterogeneous across tensor types, and strongly structured by architectural context.

This raises two questions: *can a generator trained on one set of architectures produce useful weights for an architecture family it has never seen, and can this extend from low-rank adapters to full model weights?* We address them by viewing pre-trained weights not only as the final output of optimization but as samples from an *architecture-conditioned distribution*, which a model zoo allows us to learn. The resulting initialization does not replace pre-training; it places the target model in a more favorable

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.

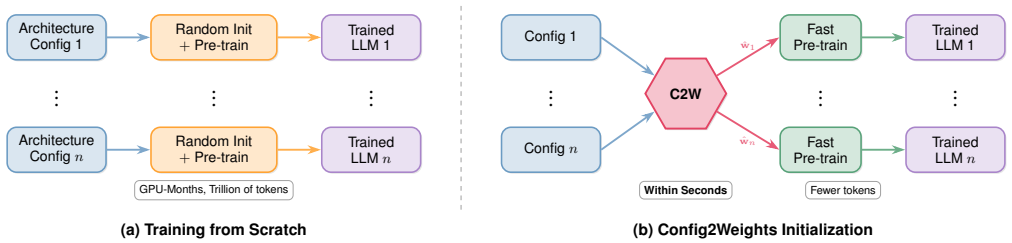


Figure 1: **C2W vs. standard training.** Standard training initializes each architecture from scratch, whereas C2W generates architecture-conditioned initializations to improve fixed-budget pretraining efficiency.

region of weight space before optimization begins, reducing early training inefficiency under a fixed compute budget.

We introduce **Config2Weights** (C2W), a framework for architecture-conditioned generative initialization of decoder-only language models. C2W represents heterogeneous parameter tensors as fixed-size chunks paired with hierarchical structural descriptors, encodes relational structure through a typed architecture graph processed by a relational graph attention network (Busbridge et al., 2019), and learns the configuration-to-weight mapping with two variants: a direct mapping and a conditional latent flow in the latent space of a pretrained conditional variational autoencoder. At inference, only the target architectural specification is required. Our main contributions are:

- **Architecture-conditioned weight generation.** We formulate LLM initialization as conditional generative modeling in weight space and introduce C2W, which generates both LoRA adapters and full-model initializations from architectural configurations.
- **Unified representation of heterogeneous weights.** We develop a chunk-based representation that places parameter tensors of different shapes and semantic roles in a common generation space while preserving architectural and positional context.
- **Cross-architecture evaluation from adapters to full models.** Generated LoRA adapters transfer to a model family excluded entirely from generator training and improve fine-tuning on held-out Llama models. On full decoder-only models up to 3B parameters, generated initializations improve pre-training over random initialization on held-out architectures.

2 Related Work

Generative Models as Weight Generators. A complementary direction treats model parameters as a data modality and explicitly models their distribution, a paradigm recently formalized under the unified framework of Weight Space Learning (WSL) (Han et al., 2026). Generative approaches, including Variational Autoencoders (VAE) (Kingma & Welling, 2022), diffusion models (Yang et al., 2024), and flow matching (FM) (Lipman et al., 2023), learn stochastic latent distributions over neural network weights, from which parameter tensors can be sampled, potentially conditioned on auxiliary information such as architecture (Peebles et al., 2022). These methods define explicit probabilistic models over weight space and support sampling-based initialization.

Hypernetworks and Hyper-Representations for Weight Synthesis. Hypernetworks (Ha et al., 2017; Chauhan et al., 2024) generate model parameters through auxiliary networks conditioned on external inputs, formulating weight generation as a deterministic mapping rather than a stochastic distribution. While early approaches struggled to scale beyond small networks, recent works have explored context-aware parameter synthesis for LLMs. For instance, SHINE (Liu et al., 2026) introduces an in-context hypernetwork to generate adapters from task descriptions, while Doc-to-LoRA (Charakorn et al., 2026) maps entire documents to weight space to bypass gradient-based fine-tuning. However, these methods primarily focus on synthesizing low-dimensional parameter subsets or adapters (Charakorn et al., 2025; Shao et al., 2025; Lv et al., 2024). Scaling such deterministic mappings to full, heterogeneous transformer checkpoints remains an open challenge.

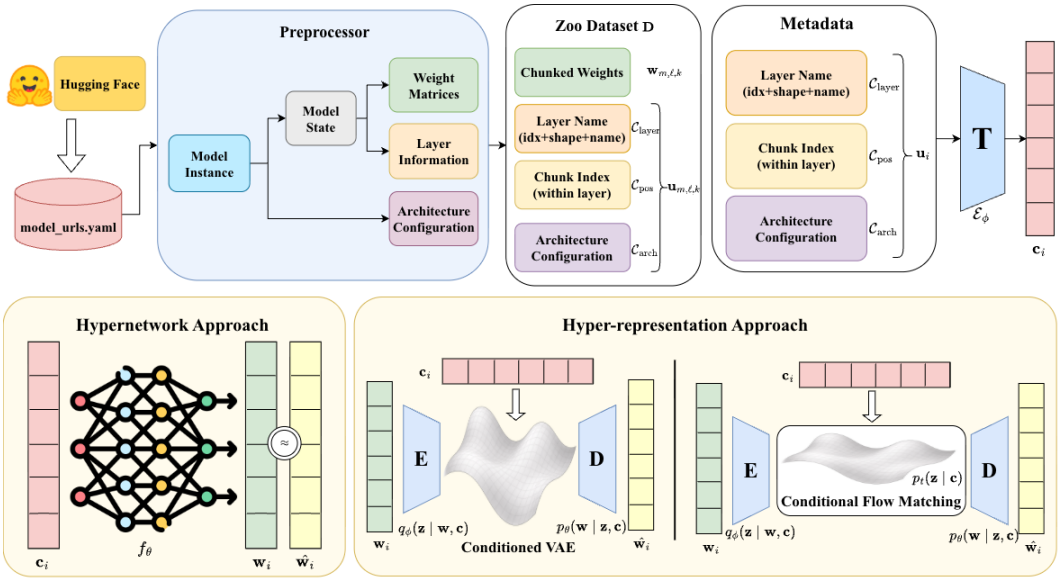


Figure 2: **Overview of the C2W framework.** *(Top)* The preprocessing pipeline extracts weight matrices, layer information, and architecture configuration from pre-trained LLMs sourced via `model_urls.yaml`, constructing the Zoo Dataset $\mathcal{D}$ by pairing weight chunks $\mathbf{w}_{m,\ell,k}$ with hierarchical descriptors $(\mathcal{C}_{\text{layer}}, \mathcal{C}_{\text{pos}}, \mathcal{C}_{\text{arch}})$, which are encoded into conditioning embeddings $\mathbf{c}_i$ via $\mathcal{E}_\phi$. *(Bottom)* Conditioned on $\mathbf{c}_i$, we study two variants: MAPPER, a deterministic hypernetwork f_θ that directly regresses weight chunks, and FLOW, a conditional VAE with encoder $q_\phi(\mathbf{z} \mid \mathbf{w}, \mathbf{c}_i)$ and decoder $p_\theta(\mathbf{w} \mid \mathbf{z}, \mathbf{c}_i)$, whose latent distribution is learned with Conditional Flow Matching.

Table 1: Overview of knowledge inheritance methods for initializing large language models. The table categorizes methods based on whether they operate on depth, width, data-free distillation, whether they handle multiple architectures, and whether they use architecture conditioning.

Method	Depth	Width	Data-Free	Multi-Arch.	Arch. Cond
LEMON (Wang et al., 2023)	✓	✓	✓	No	No
HyperCloning (Samragh et al., 2024)	×	✓	✓	No	No
G_{stack} Du et al. (2024)	✓	✓	✓	No	No
C2W	✓	✓	✓	Yes	Yes

Knowledge Inheritance and Structured Reuse. Another line of work accelerates training by reusing or transforming pre-trained weights. Net2Net (Chen et al., 2016) pioneered function-preserving expansions, followed by progressive growth and structural reuse for Transformers (Gong et al., 2019; Yang et al., 2020; Shen et al., 2022; Yao et al., 2024; Samragh et al., 2024; Du et al., 2024; Gesmundo & Maile, 2023). Methods like bert2BERT (Chen et al., 2022), Knowledge Inheritance (Qin et al., 2022), LEMON (Wang et al., 2023; Chen et al., 2024), and latent-space merging approaches (Soro et al., 2026) align, map, or combine pre-trained models to reduce training cost but require structural correspondence and do not generalize to new architectures.

Pre-training of Large-Scale Language Models. Scaling laws demonstrate that language model performance improves predictably with increased model size and compute, motivating large-scale pre-training from GPT-2/3 (Brown et al., 2020) to PaLM (Chowdhery et al., 2023) and LLaMA (Touvron et al., 2023). Such models require substantial computational resources and multi-petabyte corpora, motivating alternatives to full end-to-end pre-training.

3 APPROACH

3.1 PROBLEM DEFINITION

Pre-trained weights are often represented as flat high-dimensional vectors, a view that discards the structure induced by the architecture. We instead treat them as a structured data modality whose organization depends on architectural context. Let $\mathcal{D}_{\text{source}} = \{(\mathbf{W}^{(m)}, \mathcal{A}^{(m)})\}_{m=1}^{M}$ be a model zoo,

where $\mathbf{W}^{(m)}$ is the complete parameter set of model m and $\mathcal{A}^{(m)}$ its architectural schema. We posit an architecture-conditioned distribution $p(\mathbf{W} \mid \mathcal{A})$ over functional parameterizations and learn a conditional generative model $p_\theta(\mathbf{W} \mid \mathcal{A})$ from $\mathcal{D}_{\text{source}}$ that approximates it. For a new or held-out architecture $\mathcal{A}_{\text{new}}$, we sample $\mathbf{W}_{\text{init}} \sim p_\theta(\cdot \mid \mathcal{A}_{\text{new}})$ and use it as the starting point for standard training. The goal is a more informed initialization than random sampling, which improves early training dynamics under a fixed compute budget (Figure 2). Unlike knowledge-inheritance and model-growth methods, which require an explicit source-to-target transformation such as depth stacking or width expansion from a specific parent checkpoint, our model conditions directly on the target architecture (Table 1).

3.2 Weight Space Representation

LLM parameters are organized in tensors of heterogeneous shape, semantics, and scale. We map them to a unified modality in two stages. First, each layer-wise parameter block $\mathbf{W}_\ell^{(m)}$ is flattened via $\text{vec}(\cdot)$ and partitioned into contiguous chunks $\{\mathbf{w}_{m,\ell,k}\}_{k=1}^{S_{m,\ell}}$ of fixed length K, with $\mathbf{w}_{m,\ell,k} \in \mathbb{R}^K$ and $S_{m,\ell} = \lceil P_{m,\ell}/K \rceil$; the final chunk is zero-padded if necessary. Second, each chunk is paired with a hierarchical metadata tuple $\mathbf{u}_{m,\ell,k} = (\mathcal{C}_{\text{arch}}, \mathcal{C}_{\text{layer}}, \mathcal{C}_{\text{pos}})$ that preserves its structural identity: $\mathcal{C}_{\text{arch}}$ captures global context (model-wide hyperparameters and qualitative tags), $\mathcal{C}_{\text{layer}}$ the tensor's functional role and original shape, and $\mathcal{C}_{\text{pos}}$ the chunk index k relative to $S_{m,\ell}$, which preserves spatial topology for reconstruction. Indexing all chunks in the zoo by i yields the training set $\mathcal{D}_{\text{chunk}} = \{(\mathbf{w}_i, \mathbf{u}_i)\}_{i=1}^{J}$, which reduces the task to learning $p(\mathbf{w} \mid \mathbf{u})$.

3.3 Architectural Conditioning

We map each metadata tuple $\mathbf{u}_i$ to a continuous embedding $\mathbf{c}_i \in \mathbb{R}^{d_c}$ aligned with the weight geometry.

Flat conditioning. The flat encoder processes each chunk independently: symbolic components (layer types, tags) are encoded by a pre-trained transformer encoder (Zhang et al., 2025), numerical components (depth, chunk index) by sinusoidal encodings, and both are concatenated and projected to $\mathbf{c}_i = \mathcal{E}_\phi(\mathbf{u}_i)$. This provides a first-order prior on where a chunk resides in weight space, but it treats each tensor in isolation and ignores relational structure intrinsic to the architecture, for instance that $\mathbf{W}_Q$ and $\mathbf{W}_K$ interact through a shared attention mechanism, or that $\mathbf{W}_{\text{down}}^{(\ell)}$ feeds layer $\ell{+}1$ through the residual stream.

Graph conditioning. For larger architectures, we replace the flat encoder with a graph encoder that provides node-level embeddings. For each model we construct a typed relational graph $\mathcal{G} = (V, E, \tau_V, \tau_E)$, where τ_V and τ_E assign node and edge types. Nodes correspond to the global model, transformer layers, and non-bias weight tensors; nine edge types encode hierarchical containment, intra-block attention and MLP coupling, cross-depth same-role continuity, and weight tying (Appendix E). A relational graph attention network with edge-type-specific key and value projections produces per-node embeddings $\{\mathbf{h}_i\} = \textsc{GraphEnc}_\phi(\mathcal{G})$. The conditioning vector for chunk k of tensor node i is $\mathbf{c}_{i,k} = \text{MLP}\big([\,\mathbf{h}_i \parallel \mathbf{e}_i^{\text{meta}} \parallel \mathbf{p}_k\,]\big) \in \mathbb{R}^{d_c}$, where $\mathbf{e}_i^{\text{meta}}$ encodes tensor-level attributes (shape, depth, weight-tying status) and $\mathbf{p}_k$ is a sinusoidal position vector. Because message passing propagates information across structurally adjacent tensors before generation, each $\mathbf{c}_{i,k}$ reflects the full architectural context and captures inter-tensor coherence that the flat encoder does not.

3.4 Generative Models

Given $\mathbf{c}$, we model $p_\theta(\mathbf{w} \mid \mathbf{c})$ with two variants: a direct configuration-to-weight mapping (Mapper, also denoted Direct), and a conditional latent flow that operates in the latent space of a conditional VAE (Flow, also denoted CFM). Mapper tests how much of the weights the configuration determines; Flow captures the remaining variability.

3.4.1 DIRECT MAPPING (MAPPER)

MAPPER or Direct assumes that weights are a deterministic function of $\mathbf{c}$. A hypernetwork $f_\theta : \mathbb{R}^{d_c} \to \mathbb{R}^K$ is trained to minimize $\mathcal{L}_{\text{dir}} = \mathbb{E}_{(\boldsymbol{c},\boldsymbol{w})} \left[\|\boldsymbol{w} - f_\theta(\boldsymbol{c})\|_2^2 + \lambda\big(1 - \cos(\boldsymbol{w}, f_\theta(\boldsymbol{c}))\big) \right]$, where the cosine term discourages trivial magnitude matching by encouraging directional alignment. As a point estimate, f_θ collapses the variance of pre-trained parameters: strong performance indicates that the architecture constrains the weights tightly, while a persistent gap to FLOW indicates residual variability.

3.4.2 CONDITIONAL LATENT FLOW (FLOW)

FLOW or CFM proceeds in two stages: a conditional VAE learns a latent representation of weight chunks, and a flow model learns to generate in that latent space.

Conditional VAE. A latent variable $\boldsymbol{z} \in \mathbb{R}^{d_z}$ ($d_z \ll K$) captures stochastic degrees of freedom not determined by $\mathbf{c}$. An encoder $q_\phi(\boldsymbol{z} \mid \boldsymbol{w}, \boldsymbol{c})$, decoder $p_\theta(\boldsymbol{w} \mid \boldsymbol{z}, \boldsymbol{c})$, and learned conditional prior $p_\psi(\boldsymbol{z} \mid \boldsymbol{c})$ are trained jointly with a β-weighted evidence lower bound:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{(\boldsymbol{c},\boldsymbol{w})} \Big[\mathbb{E}_{\boldsymbol{z}\sim q_\phi} \, \ell_{\text{rec}}(\boldsymbol{w}, p_\theta(\cdot \mid \boldsymbol{z}, \boldsymbol{c})) + \beta \, D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{w}, \boldsymbol{c}) \,\|\, p_\psi(\boldsymbol{z} \mid \boldsymbol{c})) \Big]. \tag{1}$$

Latent–configuration alignment. The latent code $\boldsymbol{z}$ and the conditioning vector $\boldsymbol{c}$ encode complementary information: stochastic weight variation and deterministic architectural structure. We align them in a shared d-dimensional space through projection heads $h_w : \mathbb{R}^{d_z} \to \mathbb{R}^d$ and $h_c : \mathbb{R}^{d_c} \to \mathbb{R}^d$. With $\tilde{\boldsymbol{u}} = h_w(\boldsymbol{z})$ and $\tilde{\boldsymbol{v}} = h_c(\boldsymbol{c})$, matched pairs in a minibatch of size B are aligned with a symmetric InfoNCE objective:

$$\mathcal{L}_{\text{NCE}} = -\frac{1}{B} \sum_{i=1}^{B} \left[\log \frac{\exp(\text{sim}(\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{v}}_i)/\tau_{temp})}{\sum_j \exp(\text{sim}(\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{v}}_j)/\tau_{temp})} + \log \frac{\exp(\text{sim}(\tilde{\boldsymbol{v}}_i, \tilde{\boldsymbol{u}}_i)/\tau_{temp})}{\sum_j \exp(\text{sim}(\tilde{\boldsymbol{v}}_i, \tilde{\boldsymbol{u}}_j)/\tau_{temp})} \right], \tag{2}$$

where $\text{sim}(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b} / (\|\boldsymbol{a}\| \|\boldsymbol{b}\|)$ and $\tau_{temp} > 0$. To stabilize joint optimization of the contrastive and KL terms, we regularize the cross-correlation of the projected embeddings toward the identity (Zbontar et al., 2021), $\mathcal{L}_{\text{BT}} = \|\text{Corr}(\tilde{U}, \tilde{V}) - I\|_F^2$. The full objective is $\mathcal{L} = \mathcal{L}_{\text{VAE}} + \lambda_{\text{NCE}} \mathcal{L}_{\text{NCE}} + \lambda_{\text{BT}} \mathcal{L}_{\text{BT}}$, with $(\beta, \lambda_{\text{NCE}}, \lambda_{\text{BT}})$ annealed to prioritize reconstruction early and increase alignment gradually.

Latent flow matching. A simple parametric prior p_ψ, such as a diagonal Gaussian, only approximates the aggregated posterior induced by pre-trained weights. We therefore learn the latent distribution with Conditional Flow Matching (CFM) (Lipman et al., 2024). A vector field $v_\theta : \mathbb{R}^{d_z} \times \mathbb{R}^{d_c} \times [0, 1] \to \mathbb{R}^{d_z}$ transports $\boldsymbol{z}_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ toward posterior codes $\boldsymbol{z}_1 \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{w}, \boldsymbol{c})$ of the trained encoder. Along the linear interpolant $\boldsymbol{z}_t = (1-t)\boldsymbol{z}_0 + t\boldsymbol{z}_1$, it regresses the displacement:

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{\boldsymbol{c},\, \boldsymbol{z}_0,\, \boldsymbol{z}_1,\, t} \left[\|v_\theta(\boldsymbol{z}_t, \boldsymbol{c}, t) - (\boldsymbol{z}_1 - \boldsymbol{z}_0)\|_2^2 \right]. \tag{3}$$

This design decouples representation learning (CVAE) from generation (flow). The flow network acts as a dynamic hypernetwork: rather than regressing weights directly, it predicts the velocity field that transports noise toward latent regions of pre-trained weight chunks.

Generation. At inference, generating weights for a target architecture requires only its metadata $\mathbf{u}$; no pre-trained weights are needed. For each chunk, we compute the conditioning embedding, integrate the learned ODE from noise to a latent code, and decode it into parameter space (Algorithm 1).

4 EXPERIMENTS

We first evaluate C2W on LoRA adapters and then on full model weights, asking whether architectural configuration alone recovers functional weights across heterogeneous architectures (Q1), whether generated weights transfer to an architecture family unseen during generator training (Q2), whether generated initialization improves pre-training over random initialization and model morphisms (Q3), and whether full-model generation extends to held-out architectures and to the 3B scale (Q4). Appendix A lists the setting of every experiment.

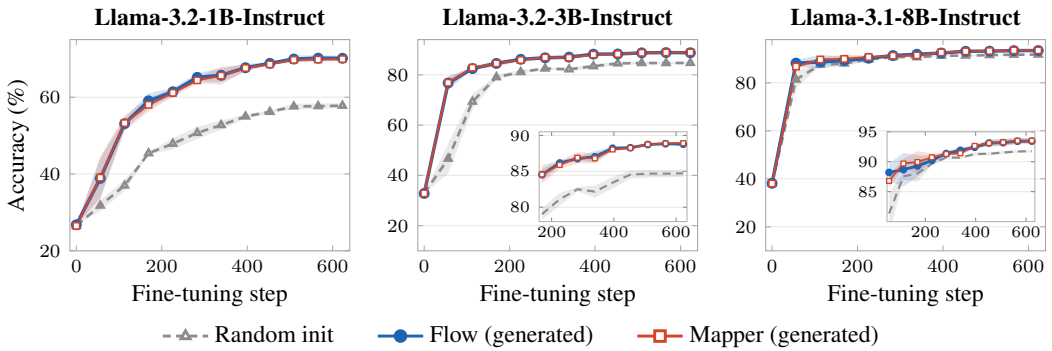


Figure 3: **Cross-architecture LoRA generation on held-out Llama models.** HellaSwag accuracy during LoRA fine-tuning starting from random initialization and from adapters generated by Flow and Mapper. Lines show the mean over seeds 11, 123, and 777; shaded regions show ± 1 standard deviation. Insets zoom into later steps for 3B and 8B.

4.1 LoRA Adapters

We first generate LoRA adapters, comparing Mapper and Flow (Section 3.4) on a zoo of 330 adapters over 22 models from six families and 15 datasets, with the Llama family held out entirely (Appendix B).

In-distribution reconstruction (Q1). On HellaSwag, adapters generated by Flow stay within 0.56 points of the original on all 22 training models, whereas Mapper drops by up to 4.37 points (Table 7).

Transfer to a held-out family (Q2). We initialize LoRA fine-tuning of three held-out Llama models with a generated HellaSwag adapter, testing an unseen family on a seen task. Starting within 0.22 points of random initialization, both generated adapters have a higher mean at every later checkpoint, and Flow and Mapper end 12.35 and 12.13 points higher on 1B, 4.05 and 4.20 on 3B, and 1.68 and 1.76 on 8B (Figure 3).

4.2 Full-Model Weights

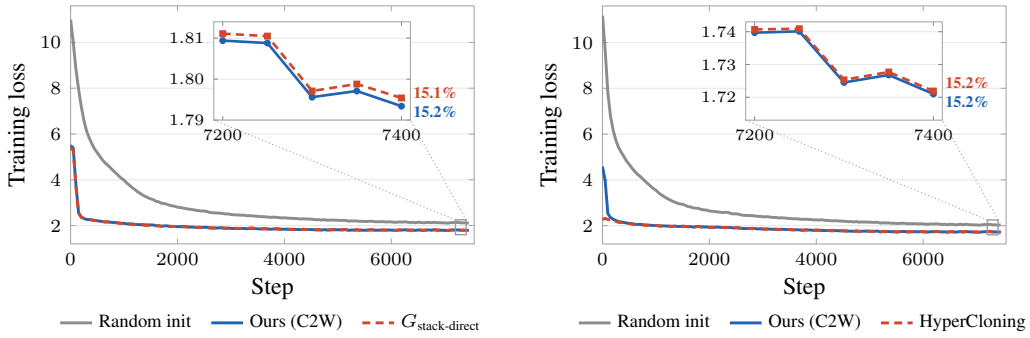
Setup. We next generate full model weights and compare against random initialization and, where applicable, model morphisms (G_{stack} and HyperCloning) (Du et al., 2024; Samragh et al., 2024). Source models and targets are listed in Appendix C, and training details in Appendix F.4.

Table 2: Zero-shot functional recovery of synthesized MobileLLM-R1-140M checkpoints, evaluated directly on downstream benchmarks without fine-tuning. Base is the original pre-trained checkpoint. MBPP and HumanEval report pass@1.

Strategy	MMLU	MBPP	HumanEval	IFEval	GSM8K
Base	24.41±0.00	**4.80**±0.01	**17.68**±0.03	12.01±0.01	19.26±0.01
Direct	24.03	4.02	17.02	11.15	18.58
CVAE	**24.85**±0.10	4.60	16.46±0.03	13.49±0.01	19.33±0.01
CFM	24.43±0.10	4.30	17.56±0.02	**13.50**±0.01	**19.43**±0.01

4.2.1 Predictability of Weights from Configuration

We first test whether configuration recovers pre-trained weights on a single architecture, `MobileLLM-R1-140M` (Zhao et al., 2025), training each generator to predict its weight chunks from configuration and chunk index. Table 2 evaluates the synthesized checkpoints directly, without fine-tuning; CVAE and Flow entries are the mean and standard deviation over three generator samples. Mapper scores below both generative variants on MMLU, MBPP, IFEval, and GSM8K, while Flow matches or exceeds the pre-trained checkpoint on MMLU, IFEval, and GSM8K and is within 0.12 points on HumanEval. Although Mapper is cheaper at inference, a deterministic mapping is too restrictive for configuration-to-weight prediction in this setting.



(a) Ours vs. $G_{\text{stack-direct}}$ (b) Ours vs. HyperCloning

Figure 4: Training loss for expanded target models under different initialization strategies. **(a)** OPT-1.3B expanded from OPT-350M via Gstack-direct. **(b)** Pythia-1.7B expanded from Pythia-410M via HyperCloning. All runs compare C2W, the respective morphism baseline, and random initialization under identical optimization settings. Insets zoom into steps 7200 to 7400; percentages give the reduction in training loss relative to random initialization at step 7400.

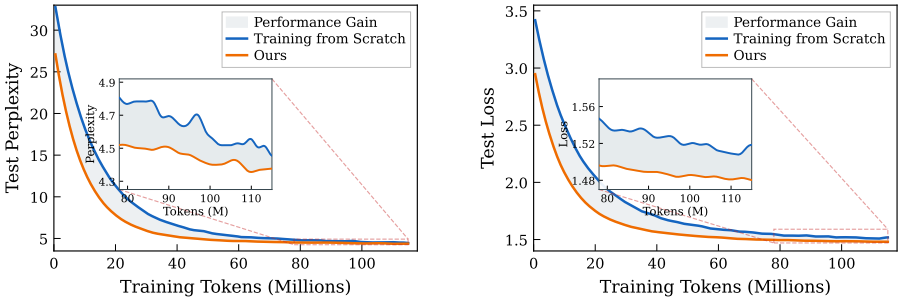


Figure 5: Test perplexity and test loss of MobileLLM-R1-140M from random initialization and from C2W Flow version. C2W achieves 3.29% lower perplexity and 2.39% lower loss.

4.2.2 Comparison with Model Morphisms

We compare against two model morphisms that expand a pre-trained parent: HyperCloning widens Pythia-410M to Pythia-1.7B, and G_{stack} stacks layers of OPT-350M to initialize OPT-1.3B. A single generator initializes both targets, and all runs are pre-trained for approximately 1B tokens under matched settings (Appendix F.4). Figure 4 reports training loss.

Against G_{stack}, C2W reduces loss rapidly during warm-up and maintains slightly lower loss through later training; against HyperCloning, it closes an initial gap and tracks the baseline closely. In both settings, C2W improves over random initialization. The curves share their overall shape because all methods train the same target architecture under matched settings; the initialization changes the starting point and early trajectory. Because both targets are expansions of models in the source zoo, the generated weights may lie close to those of structurally neighboring sources. The difference from morphisms is that one generator serves both expansion directions without handcrafted source-to-target transformations.

4.2.3 Held-Out MobileLLM Architecture

Since the morphism targets are expansions of source models, we next initialize MobileLLM-R1-140M, which is excluded from generator training. One source model, MobileLLM-R1-360M, belongs to the same family, so this setting tests a held-out configuration, complementing the held-out family of Section 4.1. Both initializations are pre-trained for 1B tokens on SlimPajama-6B (Appendix F.4).

The two initializations have similar test loss and perplexity (Figure 5) but differ on downstream accuracy (Figure 6, Table 12). The largest gain is on ARC-Easy, from 31.44% to 36.49% (+5.05); the gains on ARC-Challenge (+0.17) and MMLU are smaller. On MMLU, random initialization peaks at

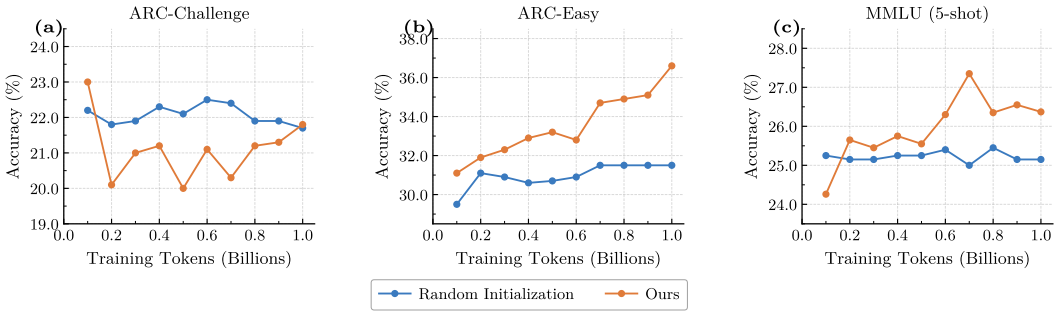
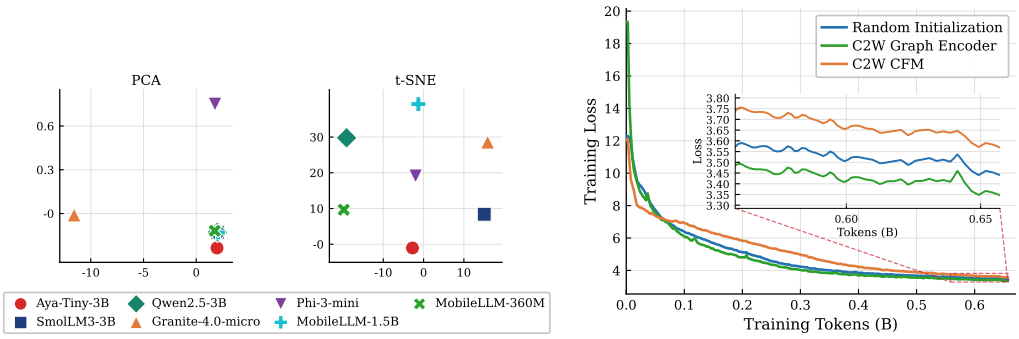


Figure 6: Downstream evaluation throughout pre-training on a fixed 1B-token compute budget. **(a)** ARC-Challenge, **(b)** ARC-Easy, and **(c)** MMLU (5-shot) accuracy are reported at regular checkpoints for both random initialization and C2W.



(a) Architecture embeddings via PCA and t-SNE.

(b) Training loss for Llama-3.2-3B on FineWeb-Edu.

Figure 7: Scaling C2W to 3B parameters. (a) The graph encoder produces well-separated embeddings across model families. (b) C2W Graph Encoder converges faster than random initialization; C2W CFM underperforms at this scale.

25.45% and ends at 25.12%, whereas C2W rises from 24.26% to 26.37%, ending 1.25 points higher, with a peak of 27.37%. Over three additional seeds (Table 13), C2W keeps a higher mean ARC-Easy accuracy (35.41% vs. 33.61%), whereas ARC-Challenge accuracy is identical in mean (16.81%) and MMLU remains close to chance for both initializations (25.64% vs. 25.28%). Cross-entropy averages token prediction over the full pre-training distribution and need not reflect differences on individual downstream tasks.

4.2.4 Scaling to 3B Parameters

We then initialize `meta-llama/Llama-3.2-3B`, which is unseen during generator training, from a zoo of seven open-weight LLMs from six families (Table 10). We compare FLOW with flat conditioning against C2W with graph conditioning (Section 3.3), which regresses weight chunks directly; the two pipelines also differ in chunk size, decoder, and training objective (Appendix F.5). The graph encoder separates model families in both PCA and t-SNE projections (Figure 7a).

On FineWeb-Edu, graph-conditioned C2W converges faster than random initialization throughout and reaches the lowest loss at 0.66B tokens (3.35 vs. 3.44), whereas FLOW with flat conditioning does not match random initialization within the same budget (Figure 7b). The graph-conditioned pipeline therefore reaches a held-out 3B target. We hypothesize that flat conditioning does not distinguish the larger number of tensor roles at this scale, but because the pipelines differ in other components, the comparison does not isolate conditioning.

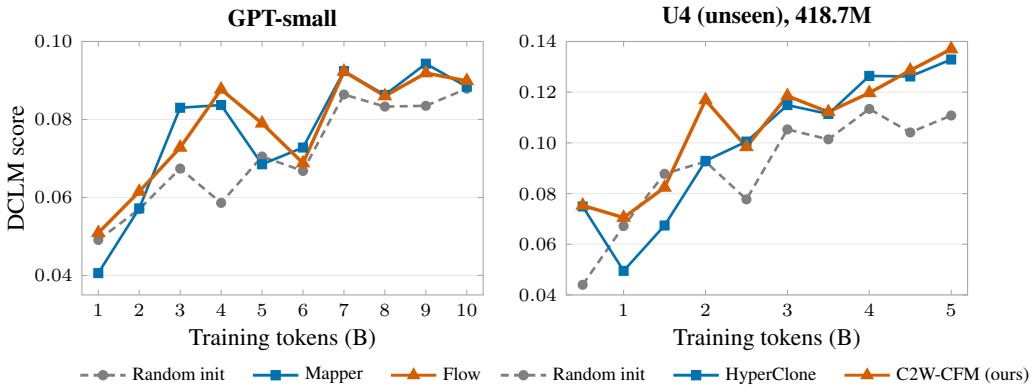


Figure 8: **DCLM score during pre-training from generated initializations.** Left: GPT-small, 10B tokens. Right: U4 (12L $\times$ 1536W $\times$ 24H, 418.7M), an unseen architecture larger than every model in the pretrained set, 5B tokens, seed 111.

4.2.5 Pre-training from Generated Initializations

We extend pre-training to 10B tokens on GPT-small and 5B tokens on U4, two held-out targets, the latter from an unseen family and larger than every source model (Q3, Q4). A generator trained on eight models that we pre-train ourselves initializes both, and we track the DCLM score against random initialization and, on U4, HyperCloning (Samragh et al., 2024) (Appendix C).

Results. On GPT-small (Figure 8, left), Flow scores above random initialization at all ten checkpoints and Mapper at eight. Both reach the final random-initialization score (0.0879 at 10B) by 7B tokens. At 10B, the three runs converge: 0.0899 (Flow), 0.0884 (Mapper), and 0.0879 (random). On U4 (Figure 8, right), Flow scores above random initialization at nine of ten checkpoints and above HyperCloning at eight. HyperCloning starts on par with Flow at 0.5B (0.0749 vs. 0.0753) but falls below random initialization at 1.0B (0.0495 vs. 0.0672). Flow surpasses the final random-initialization score (0.1108) at 2.0B tokens and ends at 0.1370, compared with 0.1329 for HyperCloning and 0.1108 for random initialization. Unlike on GPT-small, where the three runs converge by 10B, the gain over random initialization on U4 remains at the end of the 5B-token budget.

4.3 Ablation on Latent Alignment

Finally, we ablate latent alignment on MobileLLM-R1-140M (Figure 9). With MSE alone, the Winogrande accuracy of the generated model plateaus below that of the original model. Adding latent alignment improves functional recovery and exceeds the original accuracy at epochs 80 and 140; within the alignment objective, the Barlow Twins term adds negligible effect over InfoNCE. We use latent alignment by default.

5 Conclusion

We presented C2W, an architecture-conditioned framework for generating LLM initializations directly in weight space. By modeling pre-trained parameters as structured data conditioned on architectural metadata, C2W moves beyond random initialization and handcrafted model-growth rules. Generated LoRA adapters transfer to a model family excluded from generator training and improve fine-tuning over random initialization. On full decoder-only LLMs, C2W improves pre-training dynamics, is competitive with specialized expansion baselines, and generalizes to held-out architectures. These findings suggest that architectural structure provides a useful signal for generative initialization, opening a path toward more flexible and compute-efficient LLM architecture exploration.

Limitations and Future Directions

Due to computational constraints, our study is limited to target models up to 3B parameters and fixed-budget pre-training runs. The LoRA transfer experiment covers one held-out model family on a target task seen during generator training. While C2W improves convergence and downstream performance over random initialization, generated checkpoints do not yet match fully pre-trained models in zero-shot quality, and cross-architecture generation remains more challenging than cross-dataset generation. Future work should scale the model zoo, extend training horizons, and improve sampling quality for stronger generalization to unseen architectures.

Reproducibility Statement

All source models are publicly released checkpoints on the Hugging Face Hub, and all pre-training and fine-tuning data (SlimPajama-6B, FineWeb-Edu, and the LoRA datasets) are publicly available. Appendix A lists the setting of every experiment, including source models, targets, chunk size, number of training chunks, generator training, and number of runs. Appendices B and C list the LoRA and full-model zoos, Appendix E specifies the architecture graph, and Appendix F gives the generator architectures, objectives, and optimization hyperparameters. The code will be released upon acceptance.

Broader Impact

By generating effective initializations directly from architectural metadata, this work can reduce the computational and environmental cost of training large language models, broadening access for resource-constrained researchers and institutions. While lowering the barrier to producing capable models may amplify existing misuse risks, these concerns are shared across efficiency-oriented research and are best addressed through complementary efforts in alignment and responsible deployment. Our method does not introduce new model capabilities; it offers a more efficient path to the same endpoint.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1877–1901, 2020.

Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational graph attention networks, 2019. URL https://arxiv.org/abs/1904.05811.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-lora: Instant transformer adaption, 2025.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts, 2026. URL https://arxiv.org/abs/2602.15902.

Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A. Clifton. A brief review of hypernetworks in deep learning. *Artificial Intelligence Review*, 57(9), August 2024. ISSN 1573-7462. doi: 10.1007/s10462-024-10862-8. URL http://dx.doi.org/10.1007/s10462-024-10862-8.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. bert2BERT: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2134–2148, May 2022. doi: 10.18653/v1/2022.acl-long.151.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer, 2016.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Shiyao Cui, Tingwen Liu, Shuohuan Wang, Yu Sun, and Hua Wu. LEMON: Reviving stronger and smaller LMs from larger LMs with linear parameter fusion. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, August 2024.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Wenyu Du, Tongxu Luo, Zihan Qiu, Zeyu Huang, Yikang Shen, Reynold Cheng, Yike Guo, and Jie Fu. Stacking your transformers: A closer look at model growth for efficient LLM pre-training. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Andrea Gesmundo and Kaitlin Maile. Composable function-preserving expansions for transformer architectures, 2023.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2337–2346, 2019.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

Xiaolong Han, Zehong Wang, Bo Zhao, Binchi Zhang, Jundong Li, Damian Borth, Rose Yu, Haggai Maron, Yanfang Ye, Lu Yin, and Ferrante Neri. A survey of weight space learning: Understanding, representation, and generation, 2026. URL https://arxiv.org/abs/2603.10090.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. URL https://arxiv.org/abs/1312.6114.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code, 2024. URL https://arxiv.org/abs/2412.06264.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass, 2026. URL https://arxiv.org/abs/2602.06358.

Chuancheng Lv, Lei Li, Shitou Zhang, Gang Chen, Fanchao Qi, Ningyu Zhang, and Hai-Tao Zheng. HyperLoRA: Efficient cross-task generalization via constrained low-rank adapters generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 16376–16393, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.956. URL https://aclanthology.org/2024.findings-emnlp.956/.

Meta Llama Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A. Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints, 2022. URL https://arxiv.org/abs/2209.12892.

Yujia Qin, Yankai Lin, Jing Yi, Jiajie Zhang, Xu Han, Zhengyan Zhang, Yusheng Su, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Knowledge inheritance for pre-trained language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3921–3937, July 2022. doi: 10.18653/v1/2022.naacl-main.288.

Mohammad Samragh, Iman Mirzadeh, Keivan Alizadeh Vahid, Fartash Faghri, Minsik Cho, Moin Nabi, Devang Naik, and Mehrdad Farajtabar. Scaling smart: Accelerating large language model pre-training with small model initialization. In *NeurIPS Workshop*, 2024.

Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning, 2024. URL https://arxiv.org/abs/2406.09997.

Yihua Shao, Minxi Yan, Yang Liu, Siyu Chen, Wenjie Chen, Xinwei Long, Ziyang Yan, Lei Li, Chenyu Zhang, Nicu Sebe, Hao Tang, Yan Wang, Hao Zhao, Mengzhu Wang, and Jingcai Guo. In-context meta lora generation, 2025.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. Staged training for transformer language models, 2022.

Bedionita Soro, Bruno Andreis, Hayeon Lee, Wonyong Jeong, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Bedionita Soro, Aoxuan Silvia Zhang, Bruno Andreis, Jaehyeong Jo, Song Chong, and Sung Ju Hwang. LS-merge: Merging language models in latent space. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VSDV0SWwOC.

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

Yite Wang, Jiahao Su, Hanlin Lu, Cong Xie, Tianyi Liu, Jianbo Yuan, Haibin Lin, Ruoyu Sun, and Hongxia Yang. Lemon: Lossless model expansion, 2023.

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 564–572, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/wei16.html.

Cheng Yang, Shengnan Wang, Chao Yang, Yuechuan Li, Ru He, and Jingqiao Zhang. Progressively stacking 2.0: A multi-stage layerwise training method for bert training speedup, 2020.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications, 2024. URL https://arxiv.org/abs/2209.00796.

Yiqun Yao, Zheng Zhang, Jing Li, and Yequan Wang. Masked structural growth for 2x faster language model pre-training. In *The Twelfth International Conference on Learning Representations*, 2024.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction, 2021. URL https://arxiv.org/abs/2103.03230.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025. URL https://arxiv.org/abs/2506.05176.

Changsheng Zhao, Ernie Chang, Zechun Liu, Chia-Jung Chang, Wei Wen, Chen Lai, Sheng Cao, Yuandong Tian, Raghuraman Krishnamoorthi, Yangyang Shi, and Vikas Chandra. Mobilellm-r1: Exploring the limits of sub-billion language model reasoners with open training recipes, 2025. URL https://arxiv.org/abs/2509.24945.

A Experimental Protocols and Generator Cost

Table 3 summarizes the setting of each experiment. Within each experiment, all generators are trained on the same chunk-level data. For MAPPER and the CVAE, training stops once reconstructed weights match the target weights to approximately two decimal places, within the maximum number of epochs listed.

Table 4 reports generator size and hardware. The LoRA zoo and the LoRA generators were trained on a TPU v6e with 8 chips; each LoRA generator is trained for 1,000 epochs on 155,694 chunks. The full-model generators fit on a single 80GB GPU; multiple GPUs were used to reduce training time. Generator training is a one-time cost: a trained generator can initialize multiple target architectures, so we report it separately from downstream pre-training. Generating a checkpoint takes approximately 3–5 minutes with FLOW, depending on target size and chunk batching, and under 60 s for Llama-3.2-3B with the graph pipeline (Appendix F.6).

Table 3: **Experimental protocols.** Training chunk counts are approximate because tensors are chunked independently and final chunks may be padded. Ep.: maximum training epochs.

Sec.	Source models	Target(s)	Chunk size	Train chunks	Generator training	Runs
4.1	330 LoRAs, 22 models	22 training models; Llama 1B/3B/8B	16,384	156K	MAPPER/CVAE/FLOW 1K ep.	3 seeds (held-out)
4.2.1	MobileLLM-R1-140M	MobileLLM-R1-140M	16,384	9K	MAPPER/CVAE 10K ep.; FLOW 4K ep., batch 64	3 samples
4.2.2	App. C	OPT-1.3B, Pythia-1.7B	16,384	100K	MAPPER/CVAE 10K ep.; FLOW 4K ep.	1 seed
4.2.3	App. C	MobileLLM-R1-140M	16,384	90K	MAPPER/CVAE 10K ep.; FLOW 4K ep.	1 seed; 3 seeds (Tab. 13)
4.2.4	Table 10	Llama-3.2-3B	2,048 / 16,384	8–9M / 1.1M	Graph 1K ep.; FLOW 4K ep.	1 seed

Table 4: **Generator size and hardware.** –: not reported.

Sections	Generator	Trainable parameters	Hardware
4.1	MAPPER	–	TPU v6e (8 chips)
4.1	CVAE	–	TPU v6e (8 chips)
4.1	FLOW	–	TPU v6e (8 chips)
4.2.1–4.2.3	MAPPER	25M	1–2 H100
4.2.1–4.2.3	CVAE	200M (100M enc., 100M dec.)	1–2 H100
4.2.1–4.2.3	FLOW	25M (19M conditioning)	1–2 H100
4.2.4	Graph enc.–dec.	330M enc., 100M dec.	4 H100

B LoRA Zoo and Full LoRA Results

Zoo. LoRA adapters simplify the problem while keeping architectural diversity. Table 5 lists the training and held-out models, and Table 6 the datasets and adapter training settings. One adapter is trained per model–dataset pair, with rank 8 on the `q_proj` and `v_proj` projections of all language-decoder layers. Neither the weights nor the adapters of the held-out Llama family (Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct, and Llama-3.1-8B-Instruct) are used to train the generator;

for these models we report 12 checkpoints from step 0 to step 623. The generator is trained on the weights of these pre-trained adapters, collected per architecture and chunked with $K = 16{,}384$, which yields 155,694 training chunks.

Table 5: **LoRA zoo models.** Training and held-out models by family.

Role	Family	Checkpoints	#
Training	Qwen3	Qwen3-0.6B, Qwen3-4B-Instruct-2507, Qwen3-14B	3
	Qwen2.5	Qwen2.5-{0.5B, 3B, 7B}-Instruct	3
	OLMo	Olmo-3-7B-Instruct, OLMo-2-0425-1B-Instruct, OLMo-2-1124-{7B, 13B}-Instruct	4
	Gemma	gemma-3-270m-it, gemma-7b-it, gemma-2-{2b, 9b}-it, gemma-4-12B-it	5
	SmolLM2	SmolLM2-{135M, 360M, 1.7B}-Instruct	3
	LFM2.5	LFM2.5-230M, LFM2.5-350M, LFM2.5-1.2B-Instruct, LFM2.5-2.6B	4
	Total		22
Held-out[‡]	Llama	Llama-3.2-{1B, 3B}-Instruct, Llama-3.1-8B-Instruct	3

[‡]Target dataset: HellaSwag.

Table 6: **LoRA zoo datasets and training settings.** One adapter is trained per model–dataset pair ($22 \times 15 = 330$ adapters).

Domain	Datasets
Commonsense	ARC-Easy, ARC-Challenge, BoolQ, OpenBookQA, PIQA, HellaSwag[†], WinoGrande
Math	Hendrycks MATH, MathQA, Math-IIO-68K-Mini, MuMath
Coding	Glaive-Code-Assistant-v2, Python-Codes-25k, CodeAlpaca-20k, Code-74k-ShareGPT

[†]Target dataset for the held-out Llama models.

Setting	Value
Training method	SFT, completion-only
Target projections	`q_proj`, `v_proj`
Layer coverage	All decoder layers
Excluded modules	Vision/audio towers, MM proj.
Rank / α	8 / 16
Dropout / bias	0 / none
Max. examples	5,000 per dataset
Epochs / LR	1 / 10^{-4}
Effective batch size	8
Weights / adapters	BF16 / FP32

In-distribution reconstruction. Table 7 reports, for each of the 22 training models, HellaSwag accuracy on all 10,042 validation examples for the original adapter and for the adapters generated by MAPPER and FLOW (Section 4.1). MAPPER drops by more than 2 points on three models, with the largest drop on LFM2.5-2.6B. As with full-model weights (Section 4.2.1), the generative variant recovers function more reliably than direct mapping.

Held-out family: step-by-step analysis. Table 8 reports HellaSwag accuracy at every checkpoint for the held-out Llama models of Figure 3, as the mean and standard deviation over three seeds. Because HellaSwag is among the training datasets, this setting tests an unseen architecture family on a seen task. At step 0, the generated adapters are within 0.22 points of random initialization, so the gain arises during fine-tuning. The gain over random initialization is largest early: at step 56, FLOW and MAPPER improve by 7.07 and 7.26 points on 1B, 30.10 and 30.23 on 3B, and 6.86 and 5.44 on 8B. Both exceed the final accuracy of random initialization before training ends, at step 169 on 1B, step 226 on 3B, and steps 339 (FLOW) and 396 (MAPPER) on 8B. The ± 1 standard deviation bands of both generated initializations are separated from those of random initialization at every step after 0 on 1B and 3B, and on 8B at step 56 and from step 396 onward. FLOW and MAPPER differ by at most 1.02, 0.33, and 1.42 points on 1B, 3B, and 8B. The gain at step 623 decreases as model size increases, and the generated adapters, although a good initialization for LoRA fine-tuning, still require additional tuning.

Table 7: **In-distribution LoRA reconstruction.** HellaSwag accuracy (%) on all 10,042 validation examples for the original adapter and the adapters generated by MAPPER and FLOW. Δ: generated minus original (pp).

Model	Original	MAPPER	Δ	FLOW	Δ
Qwen3-0.6B	55.23	53.14	−2.09	54.67	−0.56
Qwen3-4B-Instruct-2507	89.39	88.89	−0.51	89.29	−0.11
Qwen3-14B	93.25	93.03	−0.22	93.30	+0.05
Qwen2.5-0.5B	43.59	42.82	−0.77	43.18	−0.41
Qwen2.5-3B	86.53	85.44	−1.09	86.30	−0.23
Qwen2.5-7B	91.42	91.18	−0.24	91.30	−0.12
OLMo-2-0425-1B	59.30	58.32	−0.99	59.41	+0.11
OLMo-2-1124-7B	87.01	86.38	−0.64	86.88	−0.14
OLMo-2-1124-13B	90.84	90.70	−0.14	90.78	−0.06
Olmo-3-7B	85.13	84.71	−0.42	85.01	−0.12
Gemma-3-270M	25.63	25.06	−0.57	25.92	+0.29
Gemma-2-2B	84.66	83.90	−0.77	84.71	+0.05
Gemma-7B	86.12	84.38	−1.74	85.61	−0.51
Gemma-2-9B	94.48	94.42	−0.06	94.49	+0.01
Gemma-4-12B	94.39	94.29	−0.10	94.54	+0.15
SmolLM2-135M	24.40	24.49	+0.09	24.20	−0.20
SmolLM2-360M	25.89	26.10	+0.21	25.88	−0.01
SmolLM2-1.7B	60.60	58.16	−2.44	60.43	−0.17
LFM2.5-230M	27.62	28.72	+1.10	27.78	+0.16
LFM2.5-350M	32.13	32.76	+0.64	32.32	+0.20
LFM2.5-1.2B	61.20	61.00	−0.20	61.00	−0.20
LFM2.5-2.6B	75.39	71.02	−4.37	74.96	−0.44

Table 8: **Transfer to the held-out Llama family.** HellaSwag accuracy (%) at each fine-tuning step, mean ± standard deviation over seeds 11, 123, and 777. Best mean per model and step in bold. Δ_{623}: gain of the mean over random initialization at the final step.

	Llama-3.2-1B-Instruct			Llama-3.2-3B-Instruct			Llama-3.1-8B-Instruct		
Step	Random	FLOW	MAPPER	Random	FLOW	MAPPER	Random	FLOW	MAPPER
0	26.55±0.00	**26.77**±0.09	26.49±0.02	32.81±0.00	32.81±0.14	**32.84**±0.06	37.95±0.00	**38.13**±0.08	38.05±0.00
56	31.78±1.41	38.85±5.23	**39.04**±5.50	46.62±5.99	76.72±2.73	**76.85**±2.12	81.35±3.48	**88.21**±1.47	86.79±1.05
113	37.00±1.48	53.11±1.40	**53.26**±2.02	69.32±3.05	82.49±0.39	**82.82**±0.46	87.55±0.57	88.67±2.81	**89.70**±0.81
169	45.40±0.32	**59.05**±2.01	58.03±1.75	79.04±0.80	84.53±0.82	**84.56**±1.28	88.04±1.41	89.30±1.81	**89.95**±1.81
226	47.89±1.47	**61.42**±0.95	61.14±1.10	81.10±0.90	**86.16**±0.37	85.89±0.58	89.69±0.26	90.23±0.80	**90.74**±0.66
283	50.71±1.57	**65.17**±1.14	64.40±0.82	82.50±0.30	86.76±0.23	**86.89**±0.78	90.71±0.31	**91.37**±0.35	91.29±0.21
339	52.74±1.24	**65.71**±1.78	65.61±2.32	82.17±0.84	**87.03**±0.80	86.81±0.69	90.67±0.19	**91.90**±0.62	91.42±0.56
396	54.98±0.54	**67.68**±1.04	67.61±0.83	83.42±0.78	**88.21**±0.25	88.04±0.30	91.28±0.28	92.41±0.23	**92.65**±0.27
453	56.19±0.41	**68.82**±0.24	68.55±0.08	84.54±0.57	**88.28**±0.20	88.26±0.22	91.33±0.26	**93.13**±0.57	93.12±0.50
509	57.59±0.72	**69.94**±0.92	69.73±0.65	84.68±0.53	88.73±0.12	**88.75**±0.05	91.54±0.26	93.19±0.59	**93.26**±0.55
566	57.67±0.68	**70.17**±0.78	69.82±0.52	84.67±0.49	88.84±0.08	**88.88**±0.16	91.71±0.22	93.42±0.50	**93.50**±0.46
623	57.79±0.67	**70.14**±0.69	69.92±0.58	84.72±0.52	88.77±0.10	**88.92**±0.22	91.76±0.19	93.44±0.47	**93.52**±0.42
Δ_{623}		+12.35	+12.13		+4.05	+4.20		+1.68	+1.76

C FULL-MODEL ZOO

Table 9 lists the pre-trained checkpoints used to train and evaluate C2W on full model weights. In Sections 4.2.1–4.2.4, all checkpoints are open-weight decoder-only Transformers from the Hugging Face Hub in their publicly released pre-trained form, except Phi-3-mini, for which the instruction-tuned release is used (Table 10). Table 10 gives the Hugging Face identifiers of the seven source models of the 3B experiment, which span six families and 360M–3.8B parameters; the target, Llama-3.2-3B, is a 28-layer GQA model. Section 4.2.5 uses models that we pre-train ourselves (Table 11).

Pre-training from generated initializations. The generator of Section 4.2.5 is trained on the eight models of Table 11, each pre-trained for 10B tokens, with 7–14 layers, widths 704–1024, and

Table 9: **Full-model zoo.** Generator training models and evaluated targets for each full-model experiment.

Section	Generator training models	Target(s)
4.2.1 Predictability	MobileLLM-R1-140M-base	MobileLLM-R1-140M-base
4.2.2 Morphisms	Pythia-410M, OPT-350M, MobileLLM-R1-360M-base, Qwen3-0.6B-Base	OPT-1.3B ($G^{\uparrow}_{\text{stack}}$-direct from OPT-350M); Pythia-1.7B (HyperCloning from Pythia-410M)
4.2.3 Held-out MobileLLM	MobileLLM-R1-360M-base, SmolLM2-360M, SmolLM2-135M, Qwen3-0.6B-Base	MobileLLM-R1-140M-base
4.2.4 3B scale	Seven models (Table 10)	Llama-3.2-3B
4.2.5 Pre-training	Eight models (Table 11)	GPT-small; U4
4.3 Alignment ablation	MobileLLM-R1-140M-base, SmolLM2-135M, Gemma-3-270M	MobileLLM-R1-140M-base

Table 10: Model set used for architecture-embedding and scaling analysis.

Model family	Hugging Face model ID
Aya-Tiny-3B	`CohereLabs/tiny-aya-base`
SmolLM3-3B	`HuggingFaceTB/SmolLM3-3B-Base`
Qwen2.5-3B	`Qwen/Qwen2.5-3B`
Granite-4.0-micro	`ibm-granite/granite-4.0-micro-base`
Phi-3-mini	`microsoft/Phi-3-mini-128k-instruct`
MobileLLM-1.5B	`facebook/MobileLLM-1.5B`
MobileLLM-360M	`facebook/MobileLLM-R1-360M-base`

89.7M–203.7M parameters; the table reports each model's loss at 10B tokens. Weights are split into chunks of 16,384 parameters, giving 64,758 training chunks. Neither target is used to train the generator. GPT-small is pre-trained for 10B tokens, with checkpoints every 1B. U4 (12L $\times$ 1536W $\times$ 24H, 418.7M) belongs to the unseen U family and is wider (1536 vs. 1024) and larger (418.7M vs. 203.7M) than any source model; it is pre-trained for 5B tokens (learning rate 3×10^{-4}, seed 111), with checkpoints every 0.5B. On GPT-small we compare MAPPER and FLOW against random initialization; on U4 we compare FLOW against random initialization and HyperCloning. The earlier full-model runs stop at 1B tokens or less, and their held-out targets are expansions of source models or belong to a source family; this experiment tests longer pre-training and a larger target from an unseen family. These experiments were run on the same TPU v6e server (8 chips) as the LoRA experiments.

D ADDITIONAL FULL-MODEL RESULTS

Downstream results on the held-out MobileLLM architecture. Table 12 summarizes the downstream results of Section 4.2.3 (seed 42): final ARC-Challenge and ARC-Easy accuracy and peak MMLU (5-shot) accuracy for C2W and random initialization.

Three-seed replication on the held-out MobileLLM architecture. To measure run-to-run variability, we repeat the pre-training of Section 4.2.3 with seeds 111, 123, and 412 and evaluate MMLU (5-shot) and ARC (25-shot) after 1B tokens (Table 13). The largest difference is on ARC-Easy, where C2W has a higher mean accuracy than random initialization (35.41 vs. 33.61 acc; 34.43 vs. 32.80 acc_norm). ARC-Challenge accuracy is identical in mean (16.81), and MMLU remains close to chance for both initializations (25.64 vs. 25.28).

Ablation on latent alignment. Figure 9 complements Section 4.3. With MSE alone, the Winogrande accuracy of the generated MobileLLM-R1-140M plateaus below that of the original model (dashed line). With latent alignment, the generated model recovers more of the original accuracy and exceeds it at epochs 80 and 140.

Table 11: **Pretrained set used to train the generator.** Loss is measured at 10B tokens; rows are sorted by loss.

Model	Layers	Width	Heads	Params	Loss (10B)
R10	12	1024	16	203.7M	2.8932
R9	12	896	14	161.7M	2.9578
R8	14	768	12	138.6M	2.9775
F1-XL	12	768	12	124.4M	3.0238
F2-XL	10	768	12	110.3M	3.0585
R7	9	768	12	103.2M	3.0911
F4-XL	9	704	11	89.7M	3.1172
F3-XL	7	832	13	100.9M	3.1321

Table 12: Downstream benchmark performance. The best result in each row is shown in bold.

Benchmark	Ours	Scratch	Gain
ARC Challenge	**21.84%**	21.67%	+0.17
ARC Easy	**36.49%**	31.44%	+5.05
MMLU (5-shot, Peak)	**27.37%**	25.45%	+1.92

E Architecture Graph Construction

Each model is represented as a typed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau)$ with three node types and nine edge types that capture the relational structure of a decoder-only Transformer.

Node types. The graph contains (i) a single GLOBAL node representing the model, (ii) one LAYER node per transformer block, and (iii) one TENSOR node per non-bias parameter. Each tensor node is assigned a *role* $r \in$ {q, k, v, o, q_norm, k_norm, gate, up, down, input_norm, post_attn_norm, final_norm, embed, lm_head} and a *family* $f \in$ {structural, attn_proj, mlp_proj, norm, embed_head} by pattern-matching on the Hugging Face parameter name (14 roles, 5 families).

Edge types. Nine bidirectional edge types encode: (1) GLOBAL_LAYER / LAYER_TENSOR: hierarchical containment; (2) DEPTH_SAME_ROLE: same-role tensors in consecutive layers; (3) ATTN_INTRA: all pairs among Q, K, V, O within a layer; (4) MLP_INTRA: gate↔up, gate↔down, up↔down; (5) EMBED_GLOBAL / LM_HEAD_GLOBAL / FINAL_NORM_HEAD: head connections; (6) TIE: embed↔lm_head when weights are shared.

Node features. Each node v_i carries a 14-dimensional numeric feature vector (normalized layer depth, log-scaled shape, architecture scalars H/M, I/M, V/M, head counts, tie flag), a learnable 64-dimensional role embedding, and a 32-dimensional family embedding, concatenated and projected to $\mathbb{R}^{d_g}$ by a two-layer MLP before message passing.

Relational graph attention. Let $\tau_{ij} = \tau_E(e_{ij})$ denote the relation type of edge $e_{ij} = (v_j, v_i)$. Each R-GAT layer computes multi-head attention with edge-type-specific key and value matrices $\mathbf{W}_K^{(\tau_{ij})}, \mathbf{W}_V^{(\tau_{ij})} \in \mathbb{R}^{d_g \times d_g}$ and a learned scalar bias $b_{\tau_{ij}}^{(a)}$ per head a. Attention weights use a segment softmax over edges sharing the same destination node, followed by a residual connection, LayerNorm, and a SiLU feed-forward block. We use $N_\ell = 16$ layers, $d_g = 1024$, and $H = 16$ heads for 3B-scale experiments, and $N_\ell = 6$ at smaller scales.

Per-tensor metadata encoder. A 7-dimensional vector $\mathbf{m}_i = [\text{rows}/M,\ \text{cols}/M,\ \ell/(L{-}1),\ \mathbb{1}_{2D},\ \mathbb{1}_{1D},\ \log(n{+}1)/\log(M^2),\ \mathbb{1}_{\text{tied}}]$ is processed by a two-layer MLP to produce $\mathbf{e}_i^{\text{meta}} \in \mathbb{R}^{64}$.

Table 13: **Held-out MobileLLM architecture: three-seed replication.** Accuracy (%) of MobileLLM-R1-140M after 1B tokens of pre-training on SlimPajama-6B from random initialization and from C2W, for seeds 111, 123, and 412. MMLU is 5-shot; ARC is 25-shot.

Benchmark	Initialization	Seed 111	Seed 123	Seed 412	Mean ± std
MMLU (acc)	Random	25.30	25.04	25.52	25.28 ± 0.24
	C2W	25.66	25.72	25.54	25.64 ± 0.09
ARC-Challenge (acc)	Random	16.04	16.89	17.49	16.81 ± 0.73
	C2W	16.47	16.72	17.24	16.81 ± 0.39
ARC-Challenge (acc_norm)	Random	20.48	20.73	21.59	20.93 ± 0.58
	C2W	20.65	21.59	21.93	21.39 ± 0.66
ARC-Easy (acc)	Random	33.04	33.88	33.92	33.61 ± 0.50
	C2W	36.91	34.68	34.64	35.41 ± 1.30
ARC-Easy (acc_norm)	Random	31.99	33.21	33.21	32.80 ± 0.70
	C2W	35.06	34.55	33.67	34.43 ± 0.70

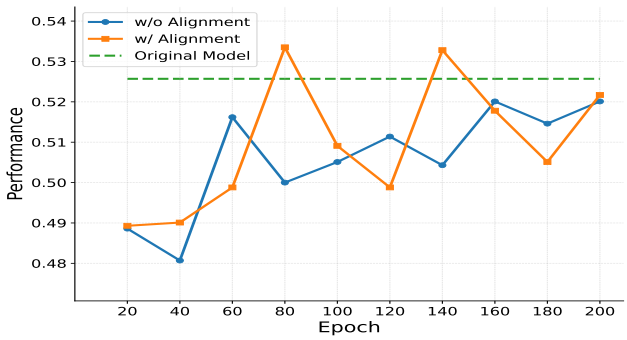


Figure 9: Winogrande accuracy of the generated `MobileLLM-R1-140M` across training epochs, with and without latent alignment regularization. The dashed line marks the original model.

Chunk position encoding. Each chunk position k is encoded as a 32-dimensional vector: (i) 4 basic features (normalized chunk index, start and end offsets, valid fraction); (ii) 8 row and column coordinates for 2D tensors; (iii) 20 sinusoidal features at frequencies $2^0, \ldots, 2^9$.

F Implementation Details

F.1 Weight Chunking and Metadata

Each parameter tensor $\mathbf{W} \in \mathbb{R}^{r \times c}$ is serialized row-major and partitioned into contiguous chunks of K scalars (final chunk zero-padded), yielding $S = \lceil rc/K \rceil$ chunks per tensor. We use $K = 2{,}048$ for the graph pipeline and $K = 16{,}384$ for the CVAE and LoRA experiments. Each chunk carries the 32-dimensional positional descriptor of Appendix E. The table below shows the layer-level metadata used for conditioning for the first layers of MobileLLM.

F.2 HashGrid Positional Memory

The graph encoder operates at tensor-node granularity and cannot resolve intra-tensor position. We therefore augment the conditioning signal with a multi-resolution hash grid (Müller et al., 2022) that provides per-chunk positional memory without additional per-model parameters.

Coordinates. Each chunk is mapped to three scalars in $[0, 1]$ encoding intra-tensor position, layer depth, and row dimension:

$$x_c = \frac{k}{\max(S-1, 1)}, \quad x_\ell = \frac{\ell}{\max(L-1, 1)}, \quad x_s = \frac{\log(r+1)}{\log(32769)}. \tag{4}$$

Table 14: Layer-level metadata used for conditioning: first 15 rows from MobileLLM-360M.

Layer Name	Layer Type	Description	Shape
model.embed_tokens	Embedding	Embedding: vocab 32000, dim 1152	32000×1152
model.layers.0.self_attn.q_proj	Linear	Linear: $1152 \to 1152$	1152×1152
model.layers.0.self_attn.k_proj	Linear	Linear: $1152 \to 384$	384×1152
model.layers.0.self_attn.v_proj	Linear	Linear: $1152 \to 384$	384×1152
model.layers.0.self_attn.o_proj	Linear	Linear: $1152 \to 1152$	1152×1152
model.layers.0.mlp.gate_proj	Linear	Linear: $1152 \to 3072$	3072×1152
model.layers.0.mlp.up_proj	Linear	Linear: $1152 \to 3072$	3072×1152
model.layers.0.mlp.down_proj	Linear	Linear: $3072 \to 1152$	1152×3072
model.layers.0.input_layernorm	LlamaRMSNorm	RMSNorm over last dimension	1152
model.layers.0.post_attention_layernorm	LlamaRMSNorm	RMSNorm over last dimension	1152
model.layers.1.self_attn.q_proj	Linear	Linear: $1152 \to 1152$	1152×1152
model.layers.1.self_attn.k_proj	Linear	Linear: $1152 \to 384$	384×1152
model.layers.1.self_attn.v_proj	Linear	Linear: $1152 \to 384$	384×1152
model.layers.1.self_attn.o_proj	Linear	Linear: $1152 \to 1152$	1152×1152
model.layers.1.mlp.gate_proj	Linear	Linear: $1152 \to 3072$	3072×1152

Encoding. The grid uses $N_{\text{lev}} = 8$ resolution levels with $F = 8$ features per level ($d_{\text{hash}} = 64$). Level i has resolution $\lfloor b^i r_{\min} \rfloor$ with $r_{\min} = 8$, $r_{\max} = 512$, and $b = \exp(\log(r_{\max}/r_{\min})/(N_{\text{lev}}-1))$. Each level indexes a table of $T = 2^{19}$ entries via

$$h(x_i, y_i, z_i) = \big[(x_i \cdot \pi_1) \oplus (y_i \cdot \pi_2) \oplus (z_i \cdot \pi_3) + \delta\big] \bmod T, \tag{5}$$

where π_1, π_2, π_3 are large primes and $\delta = (f_{\text{id}} \cdot 37 + r_{\text{id}} \cdot 1009) + i \cdot 2053$ is a deterministic per-(family, role, level) offset. Features are trilinearly interpolated over the eight surrounding cells.

Integration. The hash feature $\mathbf{g}_i \in \mathbb{R}^{64}$ is concatenated with the graph embedding $\mathbf{h}_i$, metadata embedding $\mathbf{e}_i^{\text{meta}}$, and positional descriptor $\mathbf{p}_i$:

$$\mathbf{z}_i = \text{FamilyLatentHead}\big([\mathbf{h}_i \| \mathbf{e}_i^{\text{meta}} \| \mathbf{p}_i \| \mathbf{g}_i]\big), \tag{6}$$

$$\hat{\mathbf{w}}_i = \text{RowDecoder}\big([\mathbf{z}_i \| \mathbf{e}_i^{\text{meta}} \| \mathbf{p}_i \| \mathbf{g}_i]\big). \tag{7}$$

Family-specific heads are instantiated for `attn_proj`, `mlp_proj`, and `embed_head`; normalization vectors use a shared `NormVectorHead`. The hash grid contributes $8 \times 524{,}288 \times 8 = 33.6\,\text{M}$ parameters.

F.3 Conditional VAE

The conditional VAE uses a ViT-style architecture operating on 16,384-dimensional weight chunks, treated internally as sequences of 512 tokens embedded at dimension 768. The encoder and decoder are 3-block transformer stacks with 8 heads and MLP expansion ratio 4.0. The latent space has dimension 512, represented as 32 latent tokens. Architectural metadata is encoded by a `ConfigEncoder` built on `sentence-transformers/all-MiniLM-L6-v2` (maximum length 256), producing a 768-dimensional conditioning vector through a learnable projection from a 1024-dimensional base embedding.

The training objective combines MSE reconstruction, the VAE KL divergence, and a symmetric InfoNCE alignment loss between latent codes and conditioning embeddings. An optional Barlow Twins regularizer (Zbontar et al., 2021) ($\lambda_{\text{BT}} = 0.01$) had negligible additional effect in our experiments (Section 4.3). The model is trained with AdamW at learning rate 2.5×10^{-5}.

F.4 Training Setup

Graph-pipeline training. The generator is trained on seven open-weight LLMs: `CohereLabs/tiny-aya-base`, `SmolLM3-3B-Base`, `Qwen2.5-3B`, `granite-4.0-micro`, `Phi-3-mini`, `MobileLLM-1.5B`, and `MobileLLM-R1-360M`. Weights are streamed per tensor from safetensors shards without loading full checkpoints into memory. Each training step samples 32 attention, 32 MLP, 8 normalization, and 4 embedding nodes per model, with 16 chunks per matrix node and 64 per embedding node. The loss is

$$\mathcal{L} = \alpha(\mathcal{L}_1 + 0.1\,\mathcal{L}_2) + \lambda\,\mathcal{L}_{\text{stat}}, \tag{8}$$

Table 15: **Pre-training data source composition.** Data source proportions for the full SlimPajama-627B corpus and the 6B-token pre-training subset used after initialization.

Data Source	SlimPajama-627B (%)	SlimPajama-6B (%)
CommonCrawl	52.2	54.1
C4	26.7	28.7
GitHub	5.2	4.2
Books	4.2	3.7
ArXiv	4.6	3.4
Wikipedia	3.8	3.1
StackExchange	3.3	2.8

where $\mathcal{L}_{\text{stat}} = |\mu_{\text{pred}} - \mu_{\text{tgt}}| + |\sigma_{\text{pred}} - \sigma_{\text{tgt}}|$, $\alpha = 5.0$, and λ is annealed from 0 to 0.1 over the first 100 epochs. Optimizer: AdamW ($\eta = 2 \times 10^{-4}$, $\beta = (0.9, 0.95)$, weight decay 10^{-2}, gradient clip 1.0).

Morphism comparison. All runs of Section 4.2.2 are pre-trained on the same SlimPajama subset (Table 15) for approximately 1B tokens under matched optimization settings.

Downstream pre-training. All MobileLLM-R1-140M runs use the same setup, with identical hyperparameters and data ordering for generated and randomly initialized models: SlimPajama-6B corpus (Table 15), context length 2,048, total budget 10^9 tokens, micro-batch 8×4 gradient-accumulation steps (65,536 tokens per update), AdamW ($\eta = 5 \times 10^{-4}$, weight decay 0.1, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, gradient clip 1.0), cosine schedule with a 10 M-token warmup, a single H100 GPU with `bfloat16` mixed precision, and seed 42 (seeds 111, 123, and 412 for the replication in Table 13). Checkpoints are saved every 10^8 tokens.

F.5 Graph and Flow Pipelines at the 3B Scale

The two pipelines compared in Section 4.2.4 share the chunk-based representation but differ in more than conditioning. The graph pipeline uses $K = 2{,}048$, node embeddings from the relational graph encoder combined with HashGrid features, family-specific decoding heads (Appendix F.2), and the loss in equation 8; its training set contains approximately 8–9M chunks. The Flow pipeline uses $K = 16{,}384$, flat conditioning, the CVAE decoder (Appendix F.3), and the CVAE and flow-matching objectives; its training set contains approximately 1.1M chunks. Because these components change together, the comparison in Figure 7b does not isolate the effect of conditioning.

F.6 Generating Llama-3.2-3B Weights

At inference, C2W synthesizes a complete initialization for `meta-llama/Llama-3.2-3B` from its configuration alone. **(1)** The model is instantiated on `torch.device("meta")` (zero memory) to extract parameter names and shapes, yielding a graph with 28 layers, GQA projections (24 query and 8 KV heads), gate/up/down MLP, RMSNorm, and a tied embed/lm_head, for a total of 1,476 tensor nodes. **(2)** Each tensor is split into $S = \lceil P/2{,}048 \rceil$ chunks (approximately 1.5 M in total). **(3)** Per-chunk conditioning $\mathbf{c}_i = [\mathbf{h}_i \| \mathbf{e}_i^{\text{meta}} \| \mathbf{p}_i \| \mathbf{g}_i]$ is decoded by the family-specific head. **(4)** Chunks are concatenated, trimmed, cast to `bfloat16`, and loaded for pre-training. The full pipeline completes in under 60 s on a single A100.

F.7 Weight Generation with Flow

Algorithm 1 summarizes chunk-wise generation with Flow.

G Statistical Analysis of Pre-trained Weights

We analyze the distribution of weight tensors from all transformer blocks of four models in our zoo to characterize the structure that C2W must capture. Table 16 and Figure 10 summarize the results.

Algorithm 1 C2W: Weight Generation with FLOW

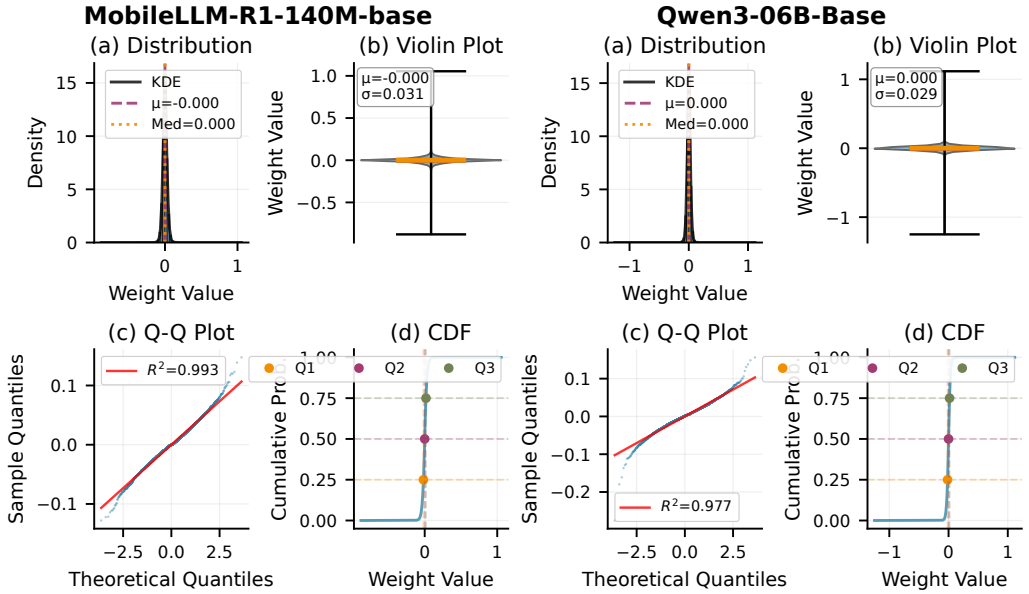
Require: Architecture $\mathcal{A}$, encoder $\mathcal{E}_\phi$, flow v_θ, decoder p_θ
Ensure: Initialized weights $\boldsymbol{W} = \{\boldsymbol{W}_\ell\}_{\ell=1}^{L}$

```
for layer ℓ = 1, ..., L do
    for chunk k = 1, ..., S_ℓ do
        c_{ℓ,k} ← E_φ(u_{ℓ,k})
        z_0 ~ N(0, I)
        z_1 ← ODESOLVE(v_θ, z_0, c_{ℓ,k}, t: 0→1)
        w_{ℓ,k} ~ p_θ(· | z_1, c_{ℓ,k})
    end for
    W_ℓ ← RESHAPE([w_{ℓ,1}; ...; w_{ℓ,S_ℓ}])
end for
return W
```

Table 16: Weight–value distribution statistics per model.

Model	Mean	Std	Median	Q1	Q3	Pos.%	Neg.%	Zero%	Outliers%
MobileLLM-R1-140M	-2.0e-05	0.0305	0.0000	-0.0175	0.0175	49.1	49.1	1.8	1.0
MobileLLM-R1-360M	2.2e-05	0.0318	0.0000	-0.0194	0.0194	50.0	50.0	0.0	0.8
Qwen-0.6B	6.0e-06	0.0288	0.0000	-0.0170	0.0170	50.0	50.0	0.0	0.9
Gemma-3-270M	-2.8e-05	0.0342	0.0000	-0.0171	0.0171	49.8	49.8	0.4	1.2

Symmetry and concentration. Across architectures, weights are sharply concentrated around zero and nearly symmetric, with means within $\pm 3 \times 10^{-5}$ and $\sigma \approx 0.03$. The interquartile range is approximately $[-0.017,\ 0.017]$, and positive and negative proportions are balanced at approximately 49–50% each. Both MobileLLM ($\kappa = 2.92$) and Qwen3 ($\kappa = 2.53$) exhibit pronounced leptokurtosis.



(a) Analysis of MobileLLM-R1-140M-base (b) Analysis of Qwen3-0.6B-Base.t

Figure 10: **Statistical profiling of weight parameters for MobileLLM-R1-140M and Qwen3-0.6B.** (a) KDE plots show zero-centred, highly concentrated weight distributions. (b) Violin plots highlight the narrow spread ($\sigma \approx 0.03$) and high kurtosis. (c) Q–Q plots exhibit strong linearity in the central region ($R^2 > 0.97$) but deviations at the extremes, indicating heavy-tailed behaviour and outlier weights. (d) CDF plots confirm the tight clustering of quartiles around zero with long, low-mass tails.

Heavy tails and non-Gaussianity. Q–Q plots reveal a two-regime structure: the central mass is close to Gaussian ($R^2 \approx 0.99$), while the tails deviate systematically. For Qwen3, more than 1.1×10^7 parameters fall outside IQR-based fences; MobileLLM has more than 2×10^6 such outliers,

reaching $|w| \approx 1$. The log-density by sign shows approximately linear decay extending well beyond Gaussian support.

Cross-architecture invariance. These statistics remain stable from 140 M to 0.6 B parameters (Table 16): the standard deviations, kurtosis profiles, and tail behaviour are nearly identical despite a $4\times$ difference in parameter count. This suggests that the statistical structure of pre-trained weights is governed more by shared training dynamics and normalization schemes than by model size. It motivates the design of C2W: a single conditional flow that synthesizes the observed zero-centred, heavy-tailed distribution, capturing both the concentrated Gaussian-like core and the rare but functionally important outliers that standard random initialization cannot produce.