MOMENTUM STREAMS FOR OPTIMIZER-INSPIRED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

The residual update of a pre-norm Transformer layer admits an interpretation as one step of a first-order optimizer acting on a surrogate token energy, wherein the attention and MLP sublayers function as gradient oracles. Based on this observation, we build a family of optimizer-inspired Transformers (triple-momentum, Adam/AdamW, Muon, SOAP) and compare them under matched compute. In our main pretraining experiment, the triple-momentum TMMFormer achieves the lowest validation loss, outperforming the vanilla Transformer and prior architectural variants. A controlled ablation and supporting theory show that momentum, not preconditioning, is the main source of the gain. We further show that TMMFormer and other momentum-based designs reach flatter minima than the vanilla Transformer, which leads to less forgetting and better generalization.

1 INTRODUCTION

The Transformer has improved mostly by redesigning its attention and MLP sublayers or by scaling them up. The *depth recurrence*—how each layer updates the residual stream—is almost always left as a plain additive update and is rarely treated as a design choice in its own right. A recent line of work suggests that a pre-norm Transformer layer can be read as one step of a first-order optimizer on a surrogate token energy, with attention and MLP acting as gradient oracles (Geshkovski et al., 2025; Zimin et al., 2026). Under this correspondence the vanilla residual block is plain gradient descent, and the recent YuriiFormer (Zimin et al., 2026) is Nesterov momentum. This invites a natural question: if a layer is an optimizer step, *which* optimizer should it implement, and how much does that choice matter? We study a family of optimizer-inspired Transformers built from this correspondence, and find that the choice matters substantially: stronger momentum templates yield markedly better residual dynamics, and a triple-momentum design, TMMFormer, is the strongest of the optimizers we try.

Optimizer-inspired Transformer design. Concretely, we treat the optimizer template as an architectural axis: we hold the attention and MLP sublayers fixed and vary only the update rule of the residual stream. Instantiating it with several classical optimizers yields a family of optimizer-inspired Transformers—TMMFormer, AdamFormer, AdamWFormer, MuonFormer, and SOAPFormer—which we pretrain under matched compute on TinyStories and OpenWebText and evaluate for downstream transfer. TMMFormer achieves the lowest validation loss in our main comparison, beating the vanilla residual stream and the prior YuriiFormer on both corpora. The advantage is architectural—robust to the parameter-training optimizer and learning rate—and is consistent with the optimization theory in which triple momentum is an optimal first-order method for strongly convex quadratics (Section 4).

Why momentum helps. To explain the source of the gain, we run a controlled momentum $\times$ preconditioning ablation. Adding a momentum stream recovers most of the improvement, whereas diagonal Adam-style preconditioning does not. The advantage therefore comes from the momentum stream itself. A full-block Jacobian analysis explains how: the auxiliary velocity turns the otherwise first-order residual map into a second-order recurrence in depth—a filter that changes how perturbations propagate through the residual stream. Across variants, weaker minimum-gain persistence tracks validation loss, and a simple local model shows the momentum recurrence is a strictly better forward filter than the vanilla first-order stream (Section 5).

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.

Loss landscape and fine-tuning behavior. To further explain the gains of optimizer-inspired Transformers, we analyze their loss landscape and fine-tuning behavior. The momentum variants converge to flatter minima than the vanilla Transformer, and these flatter minima translate into less forgetting after fine-tuning and better out-of-distribution generalization (Section 6).

Summary of Our Main Contribution

1. We study a family of optimizer-inspired Transformers and identify the triple-momentum TMMFormer as the best validation-loss model in our main comparison, beating both YuriiFormer and the vanilla Transformer.
2. With a controlled ablation and theory, we show that momentum, not preconditioning, is the main source of the gain.
3. We show that the momentum variants reach flatter minima than vanilla, with less forgetting and better out-of-distribution generalization.

2 Related Work

A line of work interprets deep learning architectures as discretizations of continuous-time dynamics or as numerical schemes for evolving representations. In the Transformer setting, Lu et al. (2019) view the model as a numerical solver for a multi-particle dynamical system and show that the standard pre-norm transformer block corresponds to a first-order Lie–Trotter splitting between an inter-token interaction term and a per-token potential term. This viewpoint also motivates alternative splitting schemes such as the Strang–Marchuk-inspired Macaron architecture, and provides early evidence that numerical-analysis principles can guide Transformer design. Together, this line of work argues that Transformer blocks are not immutable design primitives, but can instead be derived from broader algorithmic templates.

A second strand studies attention from a variational or interacting-particle perspective. Geshkovski et al. (2025) provide a mathematical framework in which Transformers are analyzed as interacting particle systems, connecting attention dynamics to gradient flows and long-time clustering over token configurations. This viewpoint justifies treating attention not as a heuristic token-mixing mechanism, but as implementing a structured gradient update on an interaction energy. It provides the conceptual bridge to import ideas of optimizer design into representation-space architecture rather than only into parameter-space training.

The most directly related work is YuriiFormer (Zimin et al., 2026), which unifies attention and MLP updates as gradient oracles for two complementary energies and reinterprets a standard GPT-style block as gradient descent on the resulting composite objective. Replacing this gradient-descent template with Nesterov-accelerated momentum (Nesterov, 1983; Polyak, 1964) yields a Transformer whose residual stream is augmented by a velocity stream and whose forward dynamics are second order in depth. Empirically, Nesterov momentum transformers improve validation loss under matched parameter and training budgets, with the improvements transferring to downstream accuracy on reasoning tasks. YuriiFormer thereby establishes both the conceptual validity and the practical promise of optimizer-informed Transformer design.

Our extension. We extend the optimizer-informed Transformer design along two axes. First, modern training optimization has developed a much richer toolkit than plain gradient descent or Nesterov. We broaden the architectural template to cover the Triple Momentum Method, AdamW, Muon, and SOAP, evaluated under identical pre-norm backbones and matched budgets. Second, we investigate *which* aspect of an optimizer survives translation into the model architecture. Our experimental and theoretical results attribute the pretraining gains to the momentum stream, while diagonal and matrix preconditioning prove largely redundant with LayerNorm and attention.

3 Preliminary

Notation. $X_\ell \in \mathbb{R}^{T\times d}$ is the residual stream at layer ℓ (T tokens, model dimension d); $\ell + \frac{1}{2}$ denotes the intermediate state after the attention substep, so a layer factorizes as $\ell \to \ell + \frac{1}{2} \to \ell + 1$ (attention

then MLP). Attn_ℓ and MLP_ℓ are the layer's two modules and $\mathrm{LN}(\cdot)$ the pre-norm LayerNorm applied before each. Optimizer-inspired blocks also propagate one or more auxiliary streams $\mathcal{S}_\ell$ (a velocity V, Adam moments M, S, or a covariance R), stabilized by dedicated auxiliary LayerNorms LN_v (velocity) and LN_u (update); auxiliary streams start from separate learned token+position embeddings, except $S_0 = \mathbf{1}$ and $R_0 = I_D$ for headwise channel covariances with head dimension $D = d/H$. Each block carries a few learned per-layer scalars: we suppress the layer index ℓ on them and use superscripts (a) and (m) for the attention- and MLP-substep copies, respectively (e.g. $\mu^{(a)}$ vs. $\mu^{(m)}$). Scalars constrained to $(0, 1)$ are parameterized as $\sigma(\cdot)$ of an unconstrained weight, positive scalars as $\mathrm{softplus}(\cdot)$.

3.1 Optimization View of Transformers

Write $X_\ell = (x_1, \ldots, x_T)^\top$ for the residual stream at layer ℓ, with token rows $x_i \in \mathbb{R}^d$. We associate to X a composite surrogate energy

$$\mathcal{J}(X) \;=\; \mathcal{E}(X) + \mathcal{F}(X),$$

where $\mathcal{E}(X) := \sum_{i,j} e^{\langle x_i, x_j \rangle}$ is a token–token *interaction* energy and $\mathcal{F}(X) := \sum_i U(x_i)$ is a per-token *potential* energy. The two sublayers of a Transformer block then act as learned negative-gradient oracles (Geshkovski et al., 2025; Zimin et al., 2026):

$$\mathrm{Attn}_\ell\big(X\big) \;\approx\; -\nabla\mathcal{E}_\ell(X), \quad \mathrm{MLP}_\ell\big(X\big) \;\approx\; -\nabla\mathcal{F}_\ell(X).$$

Any first-order descent method can be formulated as the template $x_{k+1} = \Phi\big(x_k, d_k\big)$, $d_k \approx -\nabla f(x_k)$. The optimizer template can be lifted to a pre-norm Transformer block by identifying $x_k \leftrightarrow X_\ell$, replacing the descent direction d_k by the two negative-gradient oracles $\mathrm{Attn}_\ell, \mathrm{MLP}_\ell$ applied to $\mathrm{LN}(X)$, and discretizing the joint flow on $\mathcal{E} + \mathcal{F}$ with an Lie–Trotter splitting so that the two oracles are applied sequentially. Concretely, a single Transformer layer becomes

$$X_{\ell+1/2} = \mathrm{Opt}_\ell\big(X_\ell, \mathcal{S}_\ell; \mathrm{Attn}_\ell(\mathrm{LN}(\cdot))\big), \quad X_{\ell+1} = \mathrm{Opt}_\ell\big(X_{\ell+1/2}, \mathcal{S}_{\ell+1/2}; \mathrm{MLP}_\ell(\mathrm{LN}(\cdot))\big), \quad (1)$$

where Opt_ℓ is the optimizer step written in terms of a descent direction, $\mathcal{S}_\ell$ denotes any auxiliary state the optimizer maintains alongside the iterate, such as moments. Different choices of Opt_ℓ thus give rise to different *optimizer-informed Transformers* sharing the same $\mathrm{Attn}_\ell, \mathrm{MLP}_\ell$ backbone. The vanilla pre-norm transformer block is the Lie–Trotter splitting of gradient descent:

$$X_{\ell+1/2} = X_\ell + \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)), \qquad X_{\ell+1} = X_{\ell+1/2} + \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})).$$

3.2 Optimizers Beyond Gradient Descent

We will instantiate the template (1) with five optimizer families that span the principal axes of modern first-order optimization. We give only the essential structure here. Full update rules are in Appendix A.

Heavy-Ball and Nesterov momentum. Polyak's heavy-ball method (Polyak, 1964) augments gradient descent with a velocity buffer v_k that filters past gradients. Nesterov's accelerated gradient (Nesterov, 1983) additionally evaluates the gradient at a *lookahead point* $x_k + \mu_k v_k$ and, on the strongly convex quadratic model with curvature range $[m, L]$, has contraction factor $(\sqrt{L}-\sqrt{m})/(\sqrt{L}+\sqrt{m})$. YuriiFormer (Zimin et al., 2026) applies the Nesterov method to template (1), leading to the attention layer

$$\begin{aligned} X_\ell^{\mathrm{in}} &= X_\ell + \mu\, V_\ell, \\ V_{\ell+1/2} &= \mathrm{LN}_v\Big(\beta\, V_\ell + \gamma\; \mathrm{Attn}_\ell\big(\mathrm{LN}(X_\ell^{\mathrm{in}})\big)\Big), \\ X_{\ell+1/2} &= X_\ell + V_{\ell+1/2}, \end{aligned}$$

where μ, β, γ are learned scalars and the auxiliary state is the velocity buffer, $\mathcal{S}_\ell = V_\ell$. We defer the discussion of other related works to Appendix 2.

Triple Momentum Method (TMM). TMM extends Nesterov with a second scalar ν_k that decouples the gradient-evaluation lookahead from the iterate update, $x_{k+1} = x_k + \nu_k v_{k+1}$. This gives a larger second-order update family; in the local analysis below, setting $\nu_k \equiv 1$ recovers the YuriiFormer/Nesterov-style update.

Adam and AdamW. Adam (Kingma & Ba, 2014) maintains EMAs of the first and second moments of the gradient, and rescales each coordinate to obtain a coordinate-wise adaptive step size. AdamW decouples weight decay from the adaptive update. Both methods produce updates that are *element-wise* rescaled.

Muon. Muon (Boreiko et al., 2025; Ma et al., 2026) treats a matrix-valued parameter as a single object and performs steepest descent under the spectral norm. It maintains a momentum buffer and replaces it by its orthogonal polar factor through Newton–Schulz iterations, producing updates whose singular values are all approximately one. This makes the update *spectrally isotropic* rather than coordinate-wise.

Shampoo and SOAP. Shampoo (Gupta et al., 2018) maintains separate left and right Gram matrix accumulators L_t, R_t for a matrix gradient $G \in \mathbb{R}^{m\times n}$ and applies a Kronecker-factored preconditioner $L_t^{-1/4} G R_t^{-1/4}$. SOAP (Vyas et al., 2025) improves and stabilizes Shampoo by performing Adam-style moment updates in the eigenbasis of the Kronecker factors. Both combine matrix preconditioning and adaptive updates.

4 Optimizer Inspired Transformer

In this section we describe the design of three optimizer-inspired transformer variants, and then present our main evaluation results and ablation studies. Building on the optimizer–as–architecture correspondence in Section 3, we instantiate TMMFormer, AdamFormer, and MuonFormer, and study their relative behavior under matched compute. We also build a SOAPFormer variant (right-factor Kronecker preconditioning), an AdamWFormer variant that augments AdamFormer with decoupled weight decay, and a factorial sweep of additional optimizer cells (Heavy-Ball, RMSProp, orthogonal, Shampoo). To keep the main text focused, the update rules for these six variants are deferred to Appendix C.

4.1 Optimizer Inspired Transformer Design

TMMFormer (Triple Momentum Method). A velocity stream $V_\ell \in \mathbb{R}^{T\times d}$ is propagated alongside the residual state X_ℓ, with eight learned scalars per layer: a lookahead coefficient μ, a velocity decay β, an oracle-output gain γ, and a velocity-reinjection gain ν (an attention and an MLP copy of each, superscripts (a) and (m)). Each substep applies a lookahead $\widetilde{X} = X + \mu V$, a velocity EMA (old velocity decayed by β, fresh oracle output scaled by γ, renormalized by LN_v), and an iterate update that moves X along the new velocity with gain ν; the full update rule is given in Appendix B.1. Setting $\nu \equiv 1$ recovers YuriiFormer (Zimin et al., 2026), which is our initialization for ν (Appendix E.2).

AdamFormer. Two auxiliary streams $(M_\ell, S_\ell) \in \mathbb{R}^{T\times d} \times \mathbb{R}^{T\times d}_{>0}$ track per-token first and second moments of the oracle output. Six learned scalars per layer: first- and second-moment decays β_1, β_2 and an update gain γ (an attention and an MLP copy of each). The oracle is queried at the current state X (no lookahead). M and S are EMAs of the oracle output and of its element-wise square, and X is updated along the Adam direction $M/(\sqrt{S}+\varepsilon)$, renormalized by LN_u and scaled by γ; the full update rule is given in Appendix B.2. AdamWFormer adds decoupled weight decay (Appendix C.1).

MuonFormer (orthogonalized momentum). A single momentum stream $M_\ell \in \mathbb{R}^{T\times d}$ is propagated; the update is orthogonalized by a per-token, head-wise Newton–Schulz operator $\mathrm{NS}(\cdot) : \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times d}$ before being added back into the residual (per-token reshape sizes and iteration count in Appendix E.2). Four learned scalars per layer: a momentum decay β and an update gain γ (an attention and an MLP copy of each). M is a momentum EMA of the oracle output. The update uses its orthogonal polar factor, computed per token and per head by the Newton–Schulz operator NS (driving singular values to 1) (Boreiko et al., 2025), which preserves causality for autoregressive training; the full update rule is given in Appendix B.3.

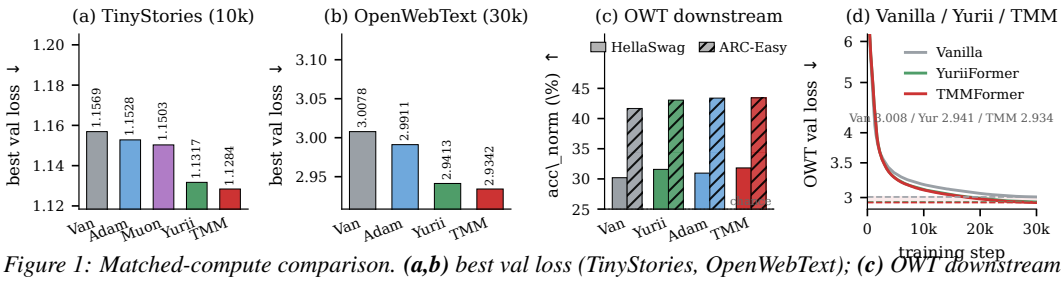

*Figure 1: Matched-compute comparison. **(a,b)** best val loss (TinyStories, OpenWebText); **(c)** OWT downstream* `acc_norm`*; **(d)** Vanilla, YuriiFormer, and TMMFormer OWT val-loss curves.*

4.2 Experimental Results

Experimental Setup. All variants, including the vanilla baseline of Section 3, share an identical 12-layer pre-norm Transformer backbone with 12 attention heads per layer and model dimension $d = 768$ (context 1024, GPT-2 BPE, weight-tied head) and differ *only* in the optimizer template of Section 4.1: vanilla has 124M parameters and every auxiliary-stream variant $\approx$ 163M. We pretrain from scratch on TinyStories (TS, 10k steps) and OpenWebText (OWT, 30k steps) with an effective batch of 480 sequences and a warmup-then-cosine schedule. Parameters are trained with two coupled optimizers—Muon on the 2D weight matrices and AdamW on embeddings, LayerNorms, and the learned per-layer scalars (at a higher learning rate)—so that the training optimizer and the architectural template stay strictly separate. We report best validation cross-entropy (nats/token) and downstream `acc_norm` on HellaSwag and ARC-Easy. Full hyperparameters are listed in Appendix E.4 (Tables 3–6).

Main results

	val loss ↓		acc_norm (%) ↑	
Variant	TS	OWT	HS	ARC
VanillaTransformer	1.1569	3.0078	30.20	41.67
AdamFormer	1.1528	2.9911	30.96	43.39
AdamWFormer	1.1472	2.9883	30.08	41.88
YuriiFormer	1.1317	2.9413	31.58	43.06
TMMFormer	**1.1284**	**2.9342**	**31.82**	**43.43**

Table 1: Best val loss (nats/token) and OWT downstream `acc_norm` (%); full results in Appendix E.7.

Main Results. **TMMFormer gives the best validation loss in our main comparison** (Table 1, Figure 1): it attains the lowest validation loss on both corpora (1.1284 TS, 2.9342 OWT) and the best downstream accuracy on HellaSwag and ARC-Easy, improving on vanilla by $\approx$0.029 nats on TS and $\approx$0.074 on OWT. Among the rows reported in Table 1, it is best in every column, and in particular surpasses YuriiFormer (Zimin et al., 2026), the prior momentum-stream architecture that it generalizes (the $\nu \equiv 1$ special case). This is consistent with the optimization view of Section 3: on the quadratic local model used to analyze the depth recurrence, triple momentum has a stronger first-order rate than the Nesterov and gradient-descent templates. TMMFormer also does not materially increase runtime: it and the vanilla transformer both train at $\approx$3 s/step on OpenWebText (no significant difference), with full per-variant wall-clock in Appendix E.6. Among the preconditioned variants, MuonFormer and SOAPFormer converge but trail every momentum-stream design—best TinyStories validation loss 1.1503 (Muon) and 1.1431 (SOAP), and on OpenWebText MuonFormer reaches only 3.0096, no better than the vanilla stream—consistent with preconditioning being a weak inductive bias for the residual stream (Section 5). Full numbers, training curves, and analysis are in Appendix E.7.

Ablation Study. To confirm the advantage is architectural, we run four ablations.
• *Peak learning rate.* We halve the Muon peak learning rate ($4\times10^{-3} \to 2\times10^{-3}$) with the rest of the recipe held fixed. On OpenWebText, Vanilla degrades from 3.008 to 3.029 (+0.021 nats) and TMMFormer from 2.934 to 2.963 (+0.029); the architectural gap shrinks only $\approx$11% (from 0.074 to 0.066), so the ordering is robust to LR perturbation.
• *Parameter-training optimizer.* We swap the optimizer on the 2D weight matrices (default Muon hybrid $\to$ a single pure AdamW for every parameter group), forming a full 2×2 with the two

architectures. The pure-AdamW peak LR is the standard nanoGPT / GPT-2-small default (6×10^{-4}), not retuned per architecture, so the comparison uses a community-accepted recipe for both sides. On OpenWebText, Vanilla barely moves ($3.008 \to 3.010$, $+0.002$ nats), while TMMFormer moves more ($2.934 \to 2.970$, $+0.035$); TMMFormer still beats Vanilla under both optimizers (gap 0.074 under Muon hybrid, 0.041 under pure AdamW).

• *Parameter-matched controls.* We train a Vanilla variant with the same parameter count as TMMFormer ($\approx$ 163M, achieved by widening d_{model} to 900). On TinyStories it closes only about 40% of the TMM–Vanilla gap (best val 1.1454, vs. TMMFormer's 1.1272 and the default Vanilla's 1.1578); the remaining ≈ 0.018 nats is roughly $9\times$ the per-seed standard deviation, so the bulk of TMMFormer's gain is not a parameter-count effect.

• *Multi-seed validation.* Separate three-seed TinyStories runs (10k steps) give Vanilla 1.1578 ± 0.0028 and TMMFormer 1.1272 ± 0.0013; these mean/std values need not match the single-run values in Table 1. The gap of 0.0306 is $\approx 15\times$ the pooled seed standard deviation, so it is not seed luck. We run this on TinyStories because the smaller corpus has both a tighter architectural gap and less data, so seed noise could plausibly be larger; since the noise on TS is already very small, OWT (with a larger gap) should be at least as stable. Across all four ablations the noise is small (a few thousandths of a nat) and the architectural effect is the dominant signal; full per-ablation numbers are in Appendix E.7.

5 Momentum

The previous section suggests that momentum is the useful ingredient: YuriiFormer (Zimin et al., 2026) and TMMFormer outperform the other optimizer-inspired variants, but they also include lookahead or learned velocity reinjection. To isolate momentum from preconditioning, we introduce three controlled variants: HBFormer keeps only a heavy-ball velocity stream, RMSPropFormer keeps only diagonal second-moment preconditioning, and OrthoFormer keeps only the Muon-style spectral preconditioner. Together with Vanilla, AdamFormer, and MuonFormer, these variants form a momentum$\times$preconditioning ablation.

Momentum $\times$ preconditioning (OWT val loss)

	No precond.	Diag. precond.	Spectral precond.
No momentum	Vanilla: 3.008	RMSProp: 3.015	Ortho: 3.033
Momentum	HB: 2.945	AdamFormer: 2.991	Muon: 3.010

Table 2: Momentum–preconditioning ablation. AdamFormer isolates diagonal preconditioning without AdamW weight decay; OrthoFormer is MuonFormer without momentum.

5.1 Momentum Is Sufficient for Most of the Gain

The cleanest evidence comes from the no-preconditioning column of Table 2. Adding only a heavy-ball momentum stream improves OpenWebText validation loss from 3.008 to 2.945. This comparison does not add Adam-style second moments, orthogonalization, or matrix preconditioning, so it isolates the effect of momentum itself.

The broader no-preconditioning ordering is then: 3.008 (Vanilla) $>$ 2.945 (HB) $>$ 2.941 (Yurii) $>$ 2.934 (TMM), where lower validation loss is better. A minimal heavy-ball stream already recovers most of the vanilla-to-Yurii gap; lookahead and learned reinjection then add smaller gains, with TMMFormer lowest in this comparison.

5.2 Preconditioning Does Not Explain the Improvement

The same ablation shows that preconditioning—diagonal or spectral—does not explain the gains in this setting. Without momentum, RMSPropFormer is slightly worse than Vanilla (3.015 versus 3.008) and OrthoFormer worse still (3.033). With momentum, AdamFormer (2.991) and MuonFormer (3.010) are both worse than the plain heavy-ball HBFormer (2.945), so in these runs the preconditioned momentum variants underperform the unpreconditioned momentum variant. The momentum effect, by contrast, holds in every column—each momentum row beats its no-momentum counterpart. Thus the main benefit comes from momentum rather than from adaptive or spectral rescaling of token-space update directions.

For diagonal preconditioning, Appendix F gives a supporting explanation, summarized informally below.

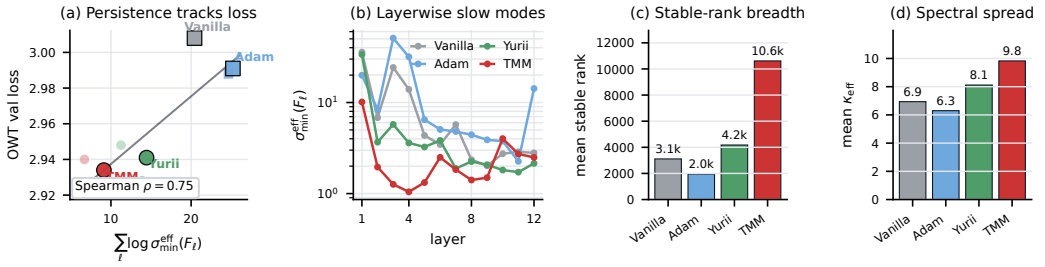

Figure 2: Full-block Jacobian spectra: (a) minimum-gain persistence versus loss, (b) layerwise minimum-gain persistence, (c) stable rank, and (d) spectral spread.

Diagonal preconditioning is redundant

Theorem 5.1. *If the Adam- or RMSProp-style second-moment stream is approximately coordinate-balanced, then its diagonal preconditioner is close to a scalar multiple of the identity. Because AdamFormer and RMSPropFormer apply an update LayerNorm after this preconditioned update, the scalar component is absorbed and cannot create a new residual direction.*

Thus, in the diagonal case, any useful preconditioning effect must come from non-scalar deviations that survive the update LayerNorm.

5.3 What Does Momentum Change?

We next ask what momentum changes inside the forward computation. Our main finding is that momentum changes how perturbations propagate across depth: momentum variants have lower minimum-gain persistence than Vanilla while preserving a broad transition spectrum. This is a forward-propagation diagnostic, not an optimization condition-number claim. To measure it, we analyze the full layer transition, not the raw attention/MLP oracle, because momentum blocks also include auxiliary streams, update LayerNorms, and learned scalar gates. The diagnostic below summarizes the full-block Jacobian along trained trajectories.

Full-Block Jacobian Diagnostic

For each layer, let $F_\ell : X_\ell \mapsto X_{\ell+1}$ be the complete block map, including auxiliary streams, update LayerNorms, and learned scalar gates, and let

$$J_\ell = \partial F_\ell / \partial X_\ell \,\big|_{X_\ell = \bar{X}_\ell}$$

be its Jacobian at trained activations, with auxiliary streams fixed to their trajectory values. We write $\sigma^{\text{eff}}_{\max}(J_\ell)$ and $\sigma^{\text{eff}}_{\min}(J_\ell)$ for the largest and smallest singular values kept by the same fixed numerical cutoff across all variants. From these singular values, we report:

$$\begin{aligned} \text{min-gain persist.:} \quad & \mathcal{P} = \textstyle\sum_\ell \log \sigma^{\text{eff}}_{\min}(J_\ell), \\ \text{stable rank:} \quad & r_{\text{st}}(J_\ell) = \|J_\ell\|_F^2 / \|J_\ell\|_2^2, \\ \text{spread:} \quad & \kappa_{\text{eff}}(J_\ell) = \sigma^{\text{eff}}_{\max}(J_\ell) / \sigma^{\text{eff}}_{\min}(J_\ell). \end{aligned}$$

Minimum-gain persistence sums the weakest layerwise gain across depth; lower values mean less amplification along the most contracted measured directions. *Stable rank* measures how broadly the Jacobian spectrum is used; higher values mean a less concentrated transition. *Spread* is the ratio between the largest and smallest effective gains; higher values mean a wider range of forward amplification, not a worse optimizer condition number.

Figure 2 summarizes the result. Across the analyzed OpenWebText variants, lower minimum-gain persistence tracks lower validation loss: the momentum variants move to the low-persistence, low-loss region, whereas AdamFormer does not. At the same time, the momentum variants have higher stable rank and larger spectral spread, indicating that they do not simply collapse the transition spectrum. Instead, momentum reshapes the full-block Jacobian into a broader forward filter while reducing

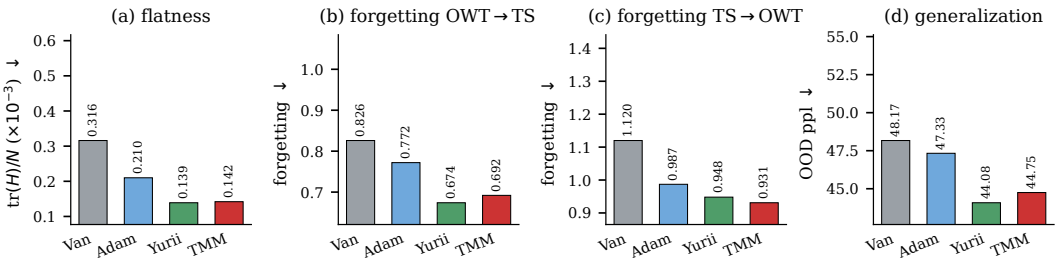

Figure 3: Flatness, forgetting, and generalization (lower is better). ***(a)*** $\mathrm{tr}(H)/N$ $(\times 10^{-3})$*;* ***(b,c)*** *forgetting (OWT→TS, TS→OWT);* ***(d)*** *out-of-distribution perplexity.*

the persistence of the most contracted measured directions. Together, these results suggest that momentum improves the residual stream by changing how perturbations propagate across layers, rather than merely changing the scale of individual updates.
Appendix G gives complementary theoretical support for this mechanism in a simplified local model; we summarize the implication informally here.

Momentum yields second-order filtering

Theorem 5.2. *In a local linearized residual-stream model around a task-relevant hidden representation, a vanilla residual Transformer implements a first-order polynomial filter over token-feature modes, whereas a momentum-stream Transformer implements a second-order filter. For a nontrivial local spectrum with condition number $\kappa > 1$, there exist stable momentum coefficients whose worst-case contraction factor is $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$, strictly smaller than the best fixed-step first-order factor $(\kappa-1)/(\kappa+1)$.*

This result is local, not a global language-modeling guarantee, but it explains why an auxiliary velocity gives a richer finite-depth filter than the vanilla first-order stream. Appendix G also shows that TMMFormer contains YuriiFormer as the $\nu_\ell = 1$ special case, supporting the empirical ordering Vanilla $<$ HBFormer $<$ YuriiFormer $<$ TMMFormer.

6 Loss Landscape of Optimizer Based Transformer

In this section, we study the loss-landscape sharpness of the optimizer-inspired transformers introduced in Section 4. We first measure the loss landscape of the vanilla transformer and of the optimizer-inspired variants, and show that the latter exhibit a flatter landscape than the vanilla transformer. We then measure the forgetting and generalization behavior of the momentum variants, and show that they generalize better and forget less than the vanilla transformer. Finally, we show that adding a learning-rate schedule further improves TMMFormer, while additionally mitigating forgetting and improving generalization. Together, these results indicate that a flatter loss landscape is one of the reasons the optimizer-inspired transformers outperform the vanilla transformer.

6.1 Loss Landscape

Setup. For each variant, we probe the curvature of the validation cross-entropy loss around the trained parameters θ at its best checkpoint, using three standard sharpness diagnostics: the top Hessian eigenvalue $\lambda_{\max}$ (via power iteration on Hessian–vector products), the Hessian trace $\mathrm{tr}(H)$ (via the Hutchinson estimator), and the loss range along a filter-normalized one-dimensional perturbation (Li et al., 2018), together with the scale-invariant trace $\mathrm{tr}(H)/N$. Every variant is probed on the same fixed validation batches, and a lower value of each diagnostic indicates a flatter minimum. Full estimator definitions and hyperparameters are given in Appendix H.

Results. Figure 3(a) reports the parameter-normalized Hessian trace $\mathrm{tr}(H)/N$ $(\times 10^{-3})$. The vanilla transformer sits in the sharpest minimum (0.316). AdamFormer already flattens it substantially (0.210), and the momentum variants reach the flattest minima by a wide margin—0.139 for YuriiFormer and 0.142 for TMMFormer, roughly $2.2\times$ flatter than vanilla and about a third lower than AdamFormer. YuriiFormer is slightly flatter than TMMFormer by this diagnostic, even though TMMFormer has the lower validation loss. Thus the landscape measurements should be read as supporting the broader momentum effect—optimizer-inspired gains are accompanied by substantially flatter minima than vanilla—rather than as a strict ordering among the two best momentum variants.

6.2 Forgetting and Generalization

The flatter minima found above are classically linked to better generalization and greater robustness to distribution shift (Foret et al., 2020). We test whether this link holds for the optimizer-inspired variants by measuring how much each one forgets under sequential fine-tuning and how well it generalizes out of distribution.

Setup. We probe two complementary axes. For *forgetting*, we fine-tune each pretrained checkpoint on a second corpus and measure how much it degrades on its original one: from an OpenWebText (resp. TinyStories) model we fine-tune on TinyStories (resp. OpenWebText) and report *forgetting*, the rise in the original corpus's loss. To isolate the architecture, every variant is fine-tuned with the *same* AdamW optimizer and schedule, regardless of its pretraining optimizer. For *generalization*, we evaluate the OpenWebText checkpoint zero-shot (no fine-tuning) on three out-of-distribution corpora—WikiText-103, LAMBADA, and C4—and report the average perplexity. Lower forgetting and lower out-of-distribution perplexity are better. Full fine-tuning hyperparameters, the evaluation protocol, and corpus details are given in Appendix I.

Results. Both axes follow the broad flatness pattern of Figure 3(b–d). Forgetting decreases monotonically from the vanilla transformer to the momentum variants in *both* transfer directions: for OpenWebText→TinyStories it drops from 0.83 (vanilla) to 0.77 (AdamFormer) to 0.69 (TMMFormer) and 0.67 (YuriiFormer), and TinyStories→OpenWebText shows the same ordering ($1.12 \to 0.99 \to 0.93/0.95$). Out-of-distribution generalization shows the same broader pattern: the average perplexity over WikiText-103, LAMBADA, and C4 falls from 48.2 for vanilla and 47.3 for AdamFormer to 44.8 for TMMFormer and 44.1 for YuriiFormer. YuriiFormer is slightly better on these robustness diagnostics, whereas TMMFormer is better on validation loss. The main conclusion is that the momentum variants sit in flatter minima, forget less, and transfer better than vanilla. Per-corpus numbers are reported in Appendix I.

6.3 Learning-Rate Schedule and Sharpness-Aware Minimization

We test two low-cost training interventions on TMMFormer, changing only the parameter-training recipe (the architecture is unchanged): a warmup–stable–decay (WSD) learning-rate schedule in place of the default warmup–cosine, and Sharpness-Aware Minimization (Foret et al., 2020) (SAM), which takes an extra ascent step toward the worst-case loss in a ρ-ball before each update. We also combine them (*SAWD*: the WSD schedule with SAM applied only during the decay phase). Schedule and SAM details are in Appendix J.

Both interventions flatten the TMMFormer minimum. WSD lowers $\mathrm{tr}(H)/N$ from 0.142 to 0.106×10^{-3} and improves OWT validation loss from 2.934 to 2.924. SAM gives a faltter minimum (0.062×10^{-3}, $\approx 2.3\times$ below cosine) but does not improve validation loss ($2.934 \to 2.940$).

7 Conclusion

Viewing a pre-norm Transformer layer as one step of a first-order optimizer on a surrogate token energy turns the choice of optimizer into an architectural design axis. Among the resulting optimizer-inspired Transformers, TMMFormer—the triple-momentum template—achieves the lowest validation loss in our main comparison, beating the vanilla stream and the prior YuriiFormer on both corpora; this gain is robust to the training optimizer and learning rate, and is consistent with the local theory of triple-momentum depth recurrences. Controlled experiments further show that the improvement comes from the optimizer's momentum design, while its preconditioning design does not explain the gain. Finally, analyzing the loss landscape and fine-tuning behavior of optimizer-inspired Transformers, we find that the momentum variants reach flatter minima than vanilla, which in turn yields less forgetting and better generalization; YuriiFormer is slightly better than TMMFormer on several of these robustness diagnostics, so these results should be interpreted as evidence for the broader momentum effect rather than a strict win for TMMFormer on every metric.

AI use statement

We did not use AI to plan or design the experiments in this work. However, we used AI tools for coding and Claude for writing assistance. We have reviewed all AI-assisted work and take responsibility for the final content of this paper.

Reproducibility statement

Every architecture is specified by its full update rule (Appendices B and C). Appendix E lists the backbone, the optimization budget, the per-group optimizer settings, and the evaluation protocol (Tables 3–6). Appendices H, I, and J give the sharpness, forgetting/generalization, and schedule/SAM protocols. Appendices D, F, and G contain the proofs. Code and pretrained checkpoints for all variants will be linked in the de-anonymized version.

References

Valentyn Boreiko, Zhiqi Bu, and Sheng Zha. Towards understanding of orthogonalization in muon. In *Tiny Titans: The next wave of On-Device Learning for Foundational Models (TTODLer-FM)*, 2025.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

Borjan Geshkovski, Cyril Letrouit, Yury Polyanskiy, and Philippe Rigollet. A mathematical perspective on transformers. *Bulletin of the American Mathematical Society*, 62(3):427–479, 2025.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. Understanding and improving transformer from a multi-particle dynamic system point of view. *arXiv preprint arXiv:1906.02762*, 2019.

Jianhao Ma, Yu Huang, Yuejie Chi, and Yuxin Chen. Preconditioning benefits of spectral orthogonalization in muon. *arXiv preprint arXiv:2601.13474*, 2026.

Y Nesterov. A method of solving a convex programming problem with convergence rate (1/k2). *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. Soap: Improving and stabilizing shampoo using adam for language modeling. In *International Conference on Learning Representations*, volume 2025, pp. 93423–93444, 2025.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*, pp. 581–590. IEEE, 2020.

Aleksandr Zimin, Yury Polyanskiy, and Philippe Rigollet. Yuriiformer: A suite of nesterov-accelerated transformers. *arXiv preprint arXiv:2601.23236*, 2026.

A Optimizer Update Rules

This appendix collects the parameter-space update rules for the optimizer templates summarised in Section 3. Each template lifts to a Transformer block by the recipe of that section: replace ∇f by the two oracles $\mathrm{Attn}_\ell(\mathrm{LN}(\cdot))$ and $\mathrm{MLP}_\ell(\mathrm{LN}(\cdot))$, maintain the optimizer's auxiliary state as a parallel residual stream, and learn the per-layer scalars.

A.1 Heavy-Ball and Nesterov Accelerated Gradient

Polyak's heavy-ball method (Polyak, 1964) adds a momentum buffer v_k to gradient descent. Nesterov's accelerated gradient (NAG) (Nesterov, 1983) evaluates the gradient at a lookahead point. With state x_k, velocity v_k, lookahead μ_k, momentum $\beta_k \in (0, 1)$, and step size $\gamma_k > 0$:

$$\begin{aligned} \tilde{x}_k &= x_k + \mu_k\, v_k, \\ v_{k+1} &= \beta_k\, v_k - \gamma_k\, \nabla f(\tilde{x}_k), \\ x_{k+1} &= x_k + v_{k+1}. \end{aligned}$$

Heavy-ball is the special case $\mu_k = 0$. For the strongly convex quadratic model with curvature range $[m, L]$, NAG has contraction factor $(\sqrt{L} - \sqrt{m})/(\sqrt{L} + \sqrt{m})$.

A.2 Triple Momentum Method

The Triple Momentum Method (TMM) introduces a second scalar ν_k that decouples the iterate update from the gradient lookahead:

$$\begin{aligned} \tilde{x}_k &= x_k + \mu_k\, v_k, \\ v_{k+1} &= \beta_k\, v_k - \gamma_k\, \nabla f(\tilde{x}_k), \\ x_{k+1} &= x_k + \nu_k\, v_{k+1}. \end{aligned}$$

NAG corresponds to $\nu_k \equiv 1$. In the layer-indexed lift, this containment is the main architectural point: the TMM template can recover the Nesterov/YuriiFormer update while learning a larger second-order filter class.

A.3 Adam and AdamW

Adam (Kingma & Ba, 2014) maintains exponential moving averages of the first and second moments of the gradient $g_k = \nabla f(x_k)$:

$$\begin{aligned} m_k &= \beta_1\, m_{k-1} + (1 - \beta_1)\, g_k, \\ s_k &= \beta_2\, s_{k-1} + (1 - \beta_2)\, g_k \odot g_k, \\ \hat{m}_k &= m_k/(1 - \beta_1^k),\ \hat{s}_k = s_k/(1 - \beta_2^k), \\ x_{k+1} &= x_k - \gamma_k \frac{m_k}{\sqrt{s_k} + \varepsilon}. \end{aligned}$$

Here $\beta_1, \beta_2 \in [0, 1)$ are decay rates and $\varepsilon > 0$ is a numerical floor. Bias correction by $1 - \beta_j^k$ is omitted in the layer-indexed lift to AdamFormer.
AdamW differs from Adam by decoupling weight decay $\lambda_k \in (0, 1)$ from the adaptive update, shrinking the iterate before applying the Adam step:

$$x_{k+1} \;=\; (1 - \lambda_k \gamma_k)\, x_k \;-\; \gamma_k \frac{m_k}{\sqrt{s_k} + \varepsilon}.$$

A.4 Muon

For a matrix-valued parameter $W \in \mathbb{R}^{m \times n}$ with gradient $G_k = \nabla_W f(W_k)$, Muon (Boreiko et al., 2025; Ma et al., 2026) replaces the momentum buffer by its orthogonal polar factor before applying it:

$$\begin{aligned} G^m_{k+1} &= \beta_k\, G^m_k + (1 - \beta_k)\, G_k, \\ U_{k+1} &= \mathrm{NS}_K\big(G^m_{k+1}\big), \\ W_{k+1} &= W_k - \gamma_k\, U_{k+1}, \end{aligned}$$

where $\mathrm{NS}_K(\cdot)$ denotes K steps of the quintic Newton–Schulz iteration applied to the Frobenius-normalized buffer $Y_0 = G^m_{k+1}/\|G^m_{k+1}\|_F$:

$$Y_{j+1} \;=\; Y_j\big(a\, I + b\, Y_j^\top Y_j + c\, (Y_j^\top Y_j)^2\big).$$

Y_K approximates the polar factor of G^m_{k+1}, so all singular values of U_{k+1} are approximately one. The resulting update is steepest descent under the spectral norm.

A.5 SHAMPOO AND SOAP

For a matrix iterate $W \in \mathbb{R}^{m\times n}$ with gradient G_k, Shampoo (Gupta et al., 2018) maintains Kronecker-factored gram matrix accumulators

$$L_{k+1} = L_k + G_k G_k^\top \in \mathbb{R}^{m\times m},$$
$$R_{k+1} = R_k + G_k^\top G_k \in \mathbb{R}^{n\times n},$$
$$W_{k+1} = W_k - \gamma_k L_{k+1}^{-1/4} G_k R_{k+1}^{-1/4}.$$

The matrix inverse fourth roots realize a structured preconditioner of the tensor.
SOAP (Vyas et al., 2025) improves and stabilizes Shampoo by carrying out Adam-style first- and second-moment updates in the eigenbasis of the Kronecker factors. Letting $L_{k+1} = Q_L \Lambda_L Q_L^\top$ and $R_{k+1} = Q_R \Lambda_R Q_R^\top$, SOAP rotates the gradient as $\widetilde{G}_k = Q_L^\top G_k Q_R$, runs Adam in the rotated coordinate system on $\widetilde{G}_k$, and rotates the resulting update back. Compared to plain Shampoo, this adds per-direction adaptivity at modest extra cost.
A common simplification used in practice, and adopted in our Shampoo/SOAP-inspired Transformer variant, is to keep only the right factor R_{k+1}.

B UPDATE RULES OF TMMFORMER, ADAMFORMER, AND MUONFORMER

This appendix gives the full per-layer update rules of the three optimizer-inspired Transformers described in Section 4.1. The notation $\ell \to \ell+1/2 \to \ell+1$, the (a) and (m) superscripts on substep scalars, and the auxiliary LayerNorms $\mathrm{LN}_v, \mathrm{LN}_u$ are as in Section 3.

B.1 TMMFORMER

TMMFormer

The attention substep $\ell \to \ell + 1/2$ is

$$\widetilde{X}_\ell = X_\ell + \mu^{(a)} V_\ell,$$
$$V_{\ell+1/2} = \mathrm{LN}_v(\beta^{(a)} V_\ell + \gamma^{(a)} \mathrm{Attn}_\ell(\mathrm{LN}(\widetilde{X}_\ell))),$$
$$X_{\ell+1/2} = X_\ell + \nu^{(a)} V_{\ell+1/2},$$

and the MLP substep $\ell + 1/2 \to \ell + 1$ is

$$\widetilde{X}_{\ell+1/2} = X_{\ell+1/2} + \mu^{(m)} V_{\ell+1/2},$$
$$V_{\ell+1} = \mathrm{LN}_v(\beta^{(m)} V_{\ell+1/2} + \gamma^{(m)} \mathrm{MLP}_\ell(\mathrm{LN}(\widetilde{X}_{\ell+1/2}))),$$
$$X_{\ell+1} = X_{\ell+1/2} + \nu^{(m)} V_{\ell+1}.$$

B.2 ADAMFORMER

AdamFormer

The attention substep is

$$G_\ell = \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),$$
$$M_{\ell+1/2} = \beta_1^{(a)} M_\ell + (1-\beta_1^{(a)}) G_\ell,$$
$$S_{\ell+1/2} = \beta_2^{(a)} S_\ell + (1-\beta_2^{(a)}) G_\ell \odot G_\ell,$$
$$X_{\ell+1/2} = X_\ell + \gamma^{(a)} \mathrm{LN}_u\left(\frac{M_{\ell+1/2}}{\sqrt{S_{\ell+1/2}}+\varepsilon}\right),$$

and the MLP substep is analogous with MLP_ℓ replacing Attn_ℓ on input $X_{\ell+1/2}$:

$$G_{\ell+1/2} = \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),$$
$$M_{\ell+1} = \beta_1^{(m)} M_{\ell+1/2} + (1-\beta_1^{(m)}) G_{\ell+1/2},$$
$$S_{\ell+1} = \beta_2^{(m)} S_{\ell+1/2} + (1-\beta_2^{(m)}) G_{\ell+1/2} \odot G_{\ell+1/2},$$
$$X_{\ell+1} = X_{\ell+1/2} + \gamma^{(m)} \mathrm{LN}_u\left(\frac{M_{\ell+1}}{\sqrt{S_{\ell+1}}+\varepsilon}\right).$$

B.3 MUONFORMER

MuonFormer

The attention substep is

$$
\begin{aligned}
G_\ell &= \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),\\
M_{\ell+1/2} &= \beta^{(a)}\, M_\ell + (1-\beta^{(a)})\, G_\ell,\\
X_{\ell+1/2} &= X_\ell + \gamma^{(a)}\ \mathrm{LN}_u(\mathrm{NS}(M_{\ell+1/2})),
\end{aligned}
$$

and the MLP substep is

$$
\begin{aligned}
G_{\ell+1/2} &= \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),\\
M_{\ell+1} &= \beta^{(m)}\, M_{\ell+1/2} + (1-\beta^{(m)})\, G_{\ell+1/2},\\
X_{\ell+1} &= X_{\ell+1/2} + \gamma^{(m)}\ \mathrm{LN}_u(\mathrm{NS}(M_{\ell+1})).
\end{aligned}
$$

C ADDITIONAL OPTIMIZER-INSPIRED TRANSFORMER VARIANTS

We collect here the update rules for the optimizer-inspired transformer variants whose details we deferred from Section 4: AdamWFormer (which adds decoupled weight decay on top of AdamFormer) and four *factorial ablation* cells—HBFormer, RMSPropFormer, OrthoFormer, ShampooFormer—each of which keeps exactly one of {momentum, preconditioner} and removes the other from one of the four main-text architectures. The notation $\ell \to \ell+1/2 \to \ell+1$, the (a) and (m) superscripts on substep scalars, and the auxiliary LayerNorms $\mathrm{LN}_v, \mathrm{LN}_u$ are as in Section 4.1.

C.1 ADAMWFORMER

AdamWFormer keeps the AdamFormer first/second-moment streams (M_ℓ, S_ℓ) and adds two scalars per substep, $\lambda^{(a)}, \lambda^{(m)} \in (0,1)$, implementing the AdamW-style decoupled weight decay: before the adaptive update is applied, the residual is contracted by $1-\lambda$. Eight learned scalars per layer $(\beta_1^{(a)}, \beta_2^{(a)}, \gamma^{(a)}, \lambda^{(a)}, \beta_1^{(m)}, \beta_2^{(m)}, \gamma^{(m)}, \lambda^{(m)})$.

AdamWFormer

The attention substep is

$$
\begin{aligned}
G_\ell &= \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),\\
M_{\ell+1/2} &= \beta_1^{(a)}\, M_\ell + (1-\beta_1^{(a)})\, G_\ell,\\
S_{\ell+1/2} &= \beta_2^{(a)}\, S_\ell + (1-\beta_2^{(a)})\, G_\ell \odot G_\ell,\\
X_{\ell+1/2} &= (1-\lambda^{(a)})\, X_\ell + \gamma^{(a)}\ \mathrm{LN}_u\!\left(\frac{M_{\ell+1/2}}{\sqrt{S_{\ell+1/2}}+\varepsilon}\right),
\end{aligned}
$$

and the MLP substep is

$$
\begin{aligned}
G_{\ell+1/2} &= \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),\\
M_{\ell+1} &= \beta_1^{(m)}\, M_{\ell+1/2} + (1-\beta_1^{(m)})\, G_{\ell+1/2},\\
S_{\ell+1} &= \beta_2^{(m)}\, S_{\ell+1/2} + (1-\beta_2^{(m)})\, G_{\ell+1/2} \odot G_{\ell+1/2},\\
X_{\ell+1} &= (1-\lambda^{(m)})\, X_{\ell+1/2} + \gamma^{(m)}\ \mathrm{LN}_u\!\left(\frac{M_{\ell+1}}{\sqrt{S_{\ell+1}}+\varepsilon}\right).
\end{aligned}
$$

We initialize $\lambda^{\mathrm{raw}} = -5$ so $\sigma(\lambda^{\mathrm{raw}}) \approx 0.007$: the model starts with near-trivial decay and learns where to push it up. Setting $\lambda^{(a)} \equiv \lambda^{(m)} \equiv 0$ recovers AdamFormer (Appendix B.2).

C.2 SOAPFORMER (RIGHT-FACTOR KRONECKER PRECONDITIONING)

A first-moment stream $M_\ell \in \mathbb{R}^{T\times H\times D}$ in head space and a per-token right covariance $R_\ell \in \mathbb{R}^{T\times D\times D}$ are propagated alongside the residual; the update is preconditioned by $R^{-1/2}$ on the channel side. Six learned scalars per layer: a first-moment decay β_1, a covariance decay β_R, and an update gain γ (an attention and an MLP copy of each). Below $G_\ell \in \mathbb{R}^{T\times H\times D}$ is the oracle output

viewed as a per-token (H, D) matrix, and $\mathrm{reshape} : \mathbb{R}^{T\times H\times D} \to \mathbb{R}^{T\times d}$ collapses the head and head-dim axes back to the residual-stream layout.

SOAPFormer

The attention substep is

$$
\begin{aligned}
G_\ell &= \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),\\
M_{\ell+1/2} &= \beta_1^{(a)}\, M_\ell + (1-\beta_1^{(a)})\, G_\ell,\\
R_{\ell+1/2} &= \beta_R^{(a)}\, R_\ell + (1-\beta_R^{(a)})\, G_\ell^\top G_\ell,\\
X_{\ell+1/2} &= X_\ell + \gamma^{(a)}\ \mathrm{LN}_u(\mathrm{reshape}(M_{\ell+1/2}\, R_{\ell+1/2}^{-1/2})),
\end{aligned}
$$

and the MLP substep is

$$
\begin{aligned}
G_{\ell+1/2} &= \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),\\
M_{\ell+1} &= \beta_1^{(m)}\, M_{\ell+1/2} + (1-\beta_1^{(m)})\, G_{\ell+1/2},\\
R_{\ell+1} &= \beta_R^{(m)}\, R_{\ell+1/2} + (1-\beta_R^{(m)})\, G_{\ell+1/2}^\top G_{\ell+1/2},\\
X_{\ell+1} &= X_{\ell+1/2} + \gamma^{(m)}\ \mathrm{LN}_u(\mathrm{reshape}(M_{\ell+1}\, R_{\ell+1}^{-1/2})).
\end{aligned}
$$

M is an Adam-style first-moment EMA of G and R is an EMA of the per-token outer product $G^\top G$; the update $MR^{-1/2}$ rescales each token's head matrix by its accumulated channel covariance. We drop the SOAP left covariance because it would precondition the token axis, where attention already learns token mixing; keeping only the right/channel covariance also avoids an additional token-side matrix factor (Appendix D). $R^{-1/2}$ is computed by $K = 10$ Newton iterations on matmuls; $R_0 = I_D$ per token, where $D = d/H$ is the head dimension. Although the identity initialization keeps the EMA covariance positive definite, each new outer-product update has rank at most $H = 12$ in $D = 64$, making the estimate poorly conditioned in this low-rank regime. We therefore avoid eigendecomposition; empirically SOAPFormer fails to converge (Section 4.2).

C.3 Factorial-Ablation Variants

The four core architectures (TMMFormer, AdamFormer, MuonFormer, SOAPFormer) populate the *momentum*×*preconditioner* table at the cells (Nesterov+TMM, none), (heavy-ball, per-coord), (heavy-ball, spectral), (heavy-ball, full-matrix). The four variants below populate four additional cells of the same table, each obtained by either (i) keeping momentum but removing the preconditioner (HBFormer) or (ii) keeping a preconditioner but removing the momentum (RMSPropFormer, OrthoFormer, ShampooFormer). All four share the attention/MLP Lie–Trotter splitting and the auxiliary-LayerNorm convention of the main text; only their auxiliary streams and update directions change.

C.3.1 HBFormer (Heavy-Ball Momentum, No Preconditioner)

HBFormer keeps the velocity stream of TMM/YuriiFormer but removes lookahead and fixes the ν iterate-gain scalar to $\nu \equiv 1$—i.e., classical Polyak heavy-ball. A velocity stream $V_\ell \in \mathbb{R}^{T\times d}$ is propagated, with four learned scalars per layer $(\beta^{(a)}, \gamma^{(a)}, \beta^{(m)}, \gamma^{(m)})$.

HBFormer

The attention substep is

$$
\begin{aligned}
G_\ell &= \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),\\
V_{\ell+1/2} &= \mathrm{LN}_v(\beta^{(a)}\, V_\ell + \gamma^{(a)}\, G_\ell),\\
X_{\ell+1/2} &= X_\ell + V_{\ell+1/2},
\end{aligned}
$$

and the MLP substep is

$$
\begin{aligned}
G_{\ell+1/2} &= \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),\\
V_{\ell+1} &= \mathrm{LN}_v(\beta^{(m)}\, V_{\ell+1/2} + \gamma^{(m)}\, G_{\ell+1/2}),\\
X_{\ell+1} &= X_{\ell+1/2} + V_{\ell+1}.
\end{aligned}
$$

HBFormer is the special case of YuriiFormer with $\mu^{(a)} \equiv \mu^{(m)} \equiv 0$ (no lookahead), and the special case of TMMFormer with $\mu^{(a)} \equiv \mu^{(m)} \equiv 0$ and $\nu^{(a)} \equiv \nu^{(m)} \equiv 1$. It isolates the contribution of pure momentum to the residual stream—no gradient lookahead, no learnable iterate gain.

C.3.2 RMSPROPFORMER (PER-COORDINATE PRECONDITIONER, NO MOMENTUM)

RMSPropFormer keeps AdamFormer's second-moment stream S_ℓ and removes the first-moment stream M_ℓ—i.e., the update direction is the raw oracle divided by $\sqrt{S}$. A single auxiliary stream $S_\ell \in \mathbb{R}^{T\times d}_{>0}$ is propagated, with four learned scalars per layer $(\beta_2^{(a)}, \gamma^{(a)}, \beta_2^{(m)}, \gamma^{(m)})$.

RMSPropFormer

The attention substep is

$$
\begin{aligned}
G_\ell &= \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),\\
S_{\ell+1/2} &= \beta_2^{(a)}\, S_\ell + (1-\beta_2^{(a)})\, G_\ell \odot G_\ell,\\
X_{\ell+1/2} &= X_\ell + \gamma^{(a)}\ \mathrm{LN}_u\!\left(\frac{G_\ell}{\sqrt{S_{\ell+1/2}+\varepsilon}}\right),
\end{aligned}
$$

and the MLP substep is

$$
\begin{aligned}
G_{\ell+1/2} &= \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),\\
S_{\ell+1} &= \beta_2^{(m)}\, S_{\ell+1/2} + (1-\beta_2^{(m)})\, G_{\ell+1/2} \odot G_{\ell+1/2},\\
X_{\ell+1} &= X_{\ell+1/2} + \gamma^{(m)}\ \mathrm{LN}_u\!\left(\frac{G_{\ell+1/2}}{\sqrt{S_{\ell+1}+\varepsilon}}\right).
\end{aligned}
$$

$S_0 = \mathbf{1}$ as in AdamFormer. RMSPropFormer is the special case of AdamFormer with $\beta_1^{(a)} \equiv \beta_1^{(m)} \equiv 0$, so that the first moment M collapses to the raw oracle G. It isolates the contribution of coordinate-wise second-moment preconditioning, without any first-moment smoothing.

C.3.3 ORTHOFORMER (SPECTRAL PRECONDITIONER, NO MOMENTUM)

OrthoFormer is MuonFormer with the momentum EMA removed: the oracle output is orthogonalized directly. There is no auxiliary stream; only two learned scalars per layer $(\gamma^{(a)}, \gamma^{(m)})$.

OrthoFormer

The attention substep is

$$
\begin{aligned}
G_\ell &= \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),\\
X_{\ell+1/2} &= X_\ell + \gamma^{(a)}\ \mathrm{LN}_u(\mathrm{NS}(G_\ell)),
\end{aligned}
$$

and the MLP substep is

$$
\begin{aligned}
G_{\ell+1/2} &= \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),\\
X_{\ell+1} &= X_{\ell+1/2} + \gamma^{(m)}\ \mathrm{LN}_u(\mathrm{NS}(G_{\ell+1/2})).
\end{aligned}
$$

NS denotes the per-token, head-wise Newton–Schulz polar-factor operator defined in Section 4.1. OrthoFormer is the special case of MuonFormer with $\beta^{(a)} \equiv \beta^{(m)} \equiv 0$ (no EMA, fresh oracle every step). It isolates the contribution of the spectral / isotropy bias of Newton–Schulz, separate from any momentum-style smoothing of the update.

C.3.4 SHAMPOOFORMER (FULL-MATRIX PRECONDITIONER, NO MOMENTUM)

ShampooFormer is SOAPFormer with the first-moment stream removed: the raw oracle is preconditioned by the running channel-side covariance. A single per-token right covariance $R_\ell \in \mathbb{R}^{T\times D\times D}$ is propagated, with four learned scalars per layer $(\beta_R^{(a)}, \gamma^{(a)}, \beta_R^{(m)}, \gamma^{(m)})$. With G_ℓ the per-token (H, D) reshape of the oracle output and $\mathrm{reshape}$ the inverse of that reshape (as in SOAPFormer, Appendix C.2),

ShampooFormer

the attention substep is

$$G_\ell = \mathrm{Attn}_\ell(\mathrm{LN}(X_\ell)),$$
$$R_{\ell+1/2} = \beta_R^{(a)}\, R_\ell + (1-\beta_R^{(a)})\, G_\ell^\top G_\ell,$$
$$X_{\ell+1/2} = X_\ell + \gamma^{(a)}\ \mathrm{LN}_u(\mathrm{reshape}(G_\ell\, R_{\ell+1/2}^{-1/2})),$$

and the MLP substep is

$$G_{\ell+1/2} = \mathrm{MLP}_\ell(\mathrm{LN}(X_{\ell+1/2})),$$
$$R_{\ell+1} = \beta_R^{(m)}\, R_{\ell+1/2} + (1-\beta_R^{(m)})\, G_{\ell+1/2}^\top G_{\ell+1/2},$$
$$X_{\ell+1} = X_{\ell+1/2} + \gamma^{(m)}\ \mathrm{LN}_u(\mathrm{reshape}(G_{\ell+1/2}\, R_{\ell+1}^{-1/2})).$$

$R_0 = I_D$ per token; $R^{-1/2}$ is computed by Newton iterations on matmuls as in SOAPFormer. ShampooFormer is the special case of SOAPFormer with $\beta_1^{(a)} \equiv \beta_1^{(m)} \equiv 0$, so the first moment M collapses to the raw oracle G. It isolates the contribution of full-matrix channel-side preconditioning, without first-moment smoothing.

Summary. Together with the three main-text variants, the six additional variants of this appendix fill in the rest of the (momentum × preconditioner) factorial table that we use to attribute the generalization gap between TMM/Yurii and Adam(W) to its momentum vs. preconditioner components. The results corresponding to each cell are reported in Section 4.2 (factorial ablation).

D Token-Side Redundancy in Matrix Preconditioning

SOAP- or Shampoo-style matrix preconditioning often uses both a token-side factor and a channel-side factor. SOAPFormer deliberately drops the token-side factor and keeps only the right, channel-side covariance. The reason is that the token-side factor acts on the same axis as attention: it mixes token positions. The next proposition gives the linearized motivation for this design choice.

Token-side preconditioning Redundancy

Proposition D.1. *Consider a linearized attention oracle*

$$\mathrm{Attn}(X) = AXW,$$

where $X \in \mathbb{R}^{T\times d}$, $A \in \mathbb{R}^{T\times T}$ is a token-mixing matrix, and $W \in \mathbb{R}^{d\times d}$ is a channel map. Let a token-side preconditioner act as

$$G \mapsto PG, \qquad P = L^{-1/2} \in \mathbb{R}^{T\times T}.$$

Then there exists another linear token-mixing matrix

$$\widetilde{A} = PA$$

such that

$$P\,\mathrm{Attn}(X) = \widetilde{A}XW.$$

Thus, in the linearized regime, a token-side preconditioner can be represented as a replacement of the attention mixing matrix by another linear token-mixing matrix.

Proof. By direct substitution,

$$P\,\mathrm{Attn}(X) = PAXW.$$

Defining $\widetilde{A} = PA$ gives

$$P\,\mathrm{Attn}(X) = \widetilde{A}XW.$$

Therefore the preconditioned operation has the form of another linear token-mixing map. □

Remark D.2 (Interpretation). The proposition is a linearized representational statement. The matrix $\widetilde{A} = PA$ need not be a valid softmax attention matrix: it may fail to be nonnegative, row-stochastic, or causal. Thus the result should not be read as an exact identity inside standard softmax attention, nor as a proof that token-side preconditioning is algebraically unnecessary in the full model. Rather, it motivates why a token-side SOAP/Shampoo factor is less compelling as an architectural intervention: its role overlaps with attention's learned token mixing, while the right factor acts on channel correlations that attention does not directly precondition.

E Experimental Details

This appendix gives the full training and evaluation configuration used for every optimizer-inspired Transformer variant in Section 4.2. Unless noted otherwise, all variants share every value in Tables 3–6; they differ only in the optimizer template of Section 4.1 and in the auxiliary streams it propagates. All numbers are taken directly from the training code.

E.1 Backbone and Tokenization

Every model uses a 12-layer, 12-head, $d = 768$ pre-norm Transformer with a context length of 1024 tokens. Tokenization is the GPT-2 byte-pair encoding (`tiktoken gpt2`, $|\mathcal{V}| = 50{,}304$), and the output projection is weight-tied to the token embedding. The vanilla model has 124M parameters; every auxiliary-stream variant has $\approx$ 163M, the extra $\approx$ 39M coming entirely from the separate learned token+position embeddings that initialize the auxiliary stream(s) (V_0, M_0, ...). The per-layer learned scalars ω_ℓ are a negligible parameter count (≤ 8 per layer) but are trained on a separate, higher learning rate (see below).

E.2 Architectural Constants and Initialization

The per-layer scalars are parameterized as in the Notation paragraph of Section 4.1: scalars in $(0, 1)$ as $\sigma(\cdot)$ of an unconstrained raw weight, and positive scalars as $\mathrm{softplus}(\cdot)$. TMMFormer's velocity-reinjection gain is initialized so that $\mathrm{softplus}(\nu^{\mathrm{raw}}) \approx 1$; training therefore begins in the $\nu \equiv 1$ YuriiFormer regime and learns where to deviate. MuonFormer reshapes each token's d-dimensional update into an $(H, D) = (12, 64)$ matrix (heads $\times$ head dimension) and applies the quintic Newton–Schulz iteration independently per token for $K = 5$ steps to recover the orthogonal polar factor, then reshapes back to d.

E.3 Two-Optimizer Parameter Training

Network parameters are split into four groups, each with its own optimizer and learning rate (Table 5):

- 2**D weight matrices** (attention/MLP projections): **Muon**, Nesterov momentum 0.95, weight decay 0, peak learning rate 0.02 (TS) / 0.004 (OWT).
- **Token/position embeddings**: AdamW, lr 6×10^{-4}, weight decay 0.1.
- **LayerNorm gains**: AdamW, lr 6×10^{-4}, weight decay 0.
- **Learned per-layer scalars** ω_ℓ (the raw parameters): AdamW, lr 3×10^{-3}, weight decay 0.

The Muon learning rate is the only optimization hyperparameter that differs between corpora; everything else is identical for TS and OWT. The single global schedule multiplier (linear warmup, then cosine decay to $0.1\times$ peak) is applied to all four groups simultaneously, so the ratio between the four learning rates is held fixed throughout training.

E.4 Hyperparameter Table

Backbone (shared by all variants)

Hyperparameter	Value
Layers / heads / d	12 / 12 / 768
Context length	1024
Tokenizer / $\lvert\mathcal{V}\rvert$	GPT-2 BPE / 50,304
Output head	weight-tied
Params (vanilla / aux-stream)	124M / $\approx$ 163M

Table 3: Backbone configuration, identical for all variants and both corpora.

Optimization budget		
Hyperparameter	**TS**	**OWT**
Total steps	10,000	30,000
Warmup steps	1,000	3,000
LR schedule	warmup $\rightarrow$ cosine	
Min-LR ratio	0.1	
Micro-batch	8	
Grad accumulation	60	
Effective batch (seqs)	480	
Tokens / step	$\approx 4.9 \times 10^5$	

Table 4: Optimization budget. Only total and warmup steps differ between TS and OWT.

Optimizers (per parameter group)		
Hyperparameter	**TS**	**OWT**
Muon (2D weights), lr	0.02	0.004
momentum / Nesterov / wd	0.95 / yes / 0	
AdamW (embeddings), lr / wd	6×10^{-4} / 0.1	
AdamW (LayerNorm), lr / wd	6×10^{-4} / 0	
AdamW (scalars), lr / wd	3×10^{-3} / 0	

Table 5: Optimizer routing and hyperparameters per parameter group. The Muon peak LR is the only TS/OWT difference.

System and evaluation	
Hyperparameter	**Value**
Precision	`bfloat16` autocast
Compilation	`torch.compile`
Data parallel	2-GPU DDP
Seeds	single (42)
Val interval / #batches	100 steps / 160
Checkpoint	best val CE
Downstream harness	`lm-eval` v0.4.3
HellaSwag / ARC-Easy	10-shot / 25-shot
Downstream metric	`acc_norm`

Table 6: System and evaluation settings, identical for all variants and both corpora.

E.5 Pretraining and Downstream Evaluation

During training we evaluate the validation cross-entropy every 100 steps on 160 fixed batches and keep the checkpoint with the lowest value (`best.pt`); pretraining quality in the main text is this best validation loss in nats/token. Downstream transfer is measured only on the best *OWT* checkpoint of each variant—TS-pretrained checkpoints sit at chance on these benchmarks because the TinyStories distribution is too narrow to transfer—using `lm-evaluation-harness` v0.4.3 with HellaSwag (10-shot) and ARC-Easy (25-shot). We report length-normalized accuracy (`acc_norm`), the harness default for these multiple-choice tasks. All runs use a single seed (42); differences within $\sim$0.01 `acc_norm` on these benchmarks at this model scale are within seed noise and are not interpreted as signal.

E.6 Wall-Clock

Per-step wall-clock for the OpenWebText runs, as printed by the training loop (hardware varies across SLURM jobs, so values are indicative). The vanilla transformer and the momentum-stream variants

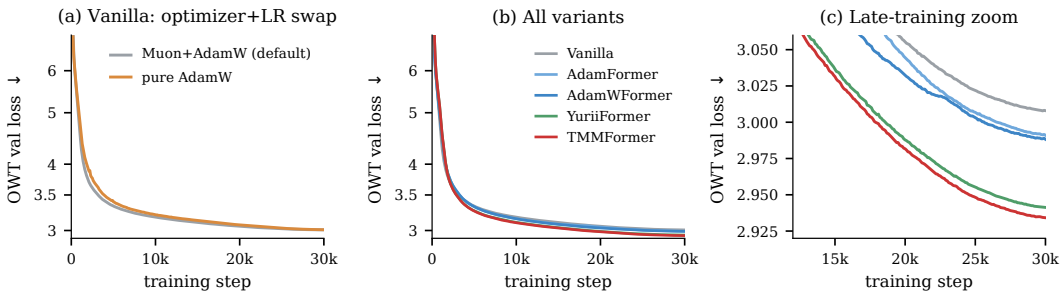

Figure 4: OWT validation-loss training curves (real per-step values from the canonical SLURM runs). ***(a)*** *Vanilla under the default Muon+AdamW recipe vs. a single pure-AdamW optimizer (optimizer+LR swap).* ***(b)*** *All optimizer-inspired variants.* ***(c)*** *The same, zoomed to late training, where the ordering TMMFormer $<$ YuriiFormer $<$ AdamFormer $\approx$ AdamWFormer $<$ Vanilla is clear.*

are all comparable: VanillaTransformer $\approx$3.3–3.4 s/step, TMMFormer $\approx$2.7–3.9 s/step, YuriiFormer $\approx$2.8 s/step, and AdamFormer $\approx$3.4 s/step. The TMMFormer auxiliary velocity stream and per-layer scalars add negligible cost relative to attention and the MLP, so TMMFormer does not materially increase runtime over the vanilla baseline. The only variant with a large penalty is MuonFormer, whose per-token Newton–Schulz iteration runs at $\approx$10–16 s/step ($\approx$3–5$\times$ slower).

E.7 Detailed Results and Ablation

Full results

	val loss ↓		acc_norm (%) ↑	
Variant	TS	OWT	HS	ARC
VanillaTransformer	1.1569	3.0078	30.20	41.67
AdamFormer	1.1528	2.9911	30.96	43.39
AdamWFormer	1.1472	2.9883	30.08	41.88
TMMFormer	**1.1284**	**2.9342**	**31.82**	**43.43**

Table 7: Full results: best val loss (nats/token) and OWT downstream `acc_norm` (%); best per column bold. Only variants with results on every task are listed; partial-task variants (MuonFormer, SOAPFormer) are reported in the text below.

Full results. Table 7 and Figure 1 report all variants against the vanilla baseline of Section 3; per-step OWT training curves for every variant are in Figure 4. TMMFormer attains the lowest validation loss on both corpora (1.1284 on TS, 2.9342 on OWT) and the best downstream transfer (31.8% HellaSwag, 43.4% ARC-Easy); the momentum-stream design improves on vanilla by $\approx$0.029 nats on TS and $\approx$0.074 on OWT, and the pretraining ordering (TMMFormer $>$ AdamFormer $\approx$ AdamWFormer $>$ Vanilla) transfers *exactly* to downstream accuracy, so the gain is genuine generalization rather than a pretraining-loss artifact. AdamFormer and AdamWFormer recover part of the gap—an adaptive per-coordinate update is better than none—but the diagonal $\sqrt{S}$ preconditioner is largely absorbed by the subsequent LN_u (Appendix F), so they fall well short of the second-order momentum recurrence. MuonFormer and SOAPFormer do not have results on every task and are therefore omitted from Table 7; we report them here. MuonFormer converges on both corpora but its spectral preconditioning yields no gain (1.1503 on TS, between the two Adam variants, and 3.0096 on OWT, no better than vanilla). SOAPFormer converges on TinyStories (1.1431, the weakest matrix-preconditioned variant) but its per-token full-matrix $R^{-1/2}$ is poorly conditioned in the per-token covariance estimate and did not complete the OpenWebText run. These two negative results reinforce the central finding: among updates that *do* train stably inside the residual stream, the Nesterov-style and triple-momentum second-order recurrences are the most effective Transformer blocks, consistent with the optimization view of Section 3 (triple momentum has an optimal rate on the strongly-convex quadratic model used to study the depth recurrence) and analyzed architecturally in Appendix G.

Optimizer-on-2D ablation (Vanilla, OWT, step-aligned)

step	pure AdamW	Muon+AdamW	Δ
1k	4.844	4.638	+0.206
2k	3.923	3.734	+0.189
3k	3.651	3.528	+0.122
5k	3.397	3.334	+0.063
7k	3.294	3.251	+0.043
10k	3.213	3.178	+0.035
15k	3.132	3.107	+0.025
20k	3.074	3.055	+0.018
25k	3.030	3.022	+0.009
30k	3.011	3.008	+0.002

Table 8: Step-aligned OWT val loss for *Vanilla* under the two parameter-training optimizers (Muon hybrid vs. pure AdamW). The Muon-hybrid early-training advantage decays monotonically to within one val-eval by step 30k.

Optimizer and learning-rate ablation. To check that the TMMFormer advantage is architectural and not an artifact of the Muon+AdamW training recipe, we run two independent OWT ablations.
Optimizer on 2D weights. We replace the Muon optimizer that updates the 48 2D matrix weights (qkv, out, w1, w2 across the 12 blocks) with AdamW, holding the schedule, batch, and the embedding and LayerNorm optimizer fixed at AdamW @ 6×10^{-4}. Combined with the two architectures this is a full 2×2: Vanilla goes from 3.0078 (Muon hybrid) to 3.0103 (pure AdamW), Δ=0.0025, while TMMFormer goes from 2.9342 to 2.9696, Δ=0.0354. TMMFormer beats Vanilla under both optimizers (architectural gap 0.074 under the Muon hybrid, 0.041 under pure AdamW). The two optimizers are essentially indistinguishable on Vanilla at the final step (within one val-eval) but distinguishable on TMMFormer ($\approx 14\times$ larger gap); the step-aligned Vanilla curves (Table 8, Figure 4a) show that the Muon-hybrid early-training lead on Vanilla decays monotonically and disappears by 30k, so the converged Vanilla numbers are essentially on top of each other. The architectural ordering survives in either column.
Learning-rate sweep (partial). Holding the Muon+AdamW recipe fixed, we halve the Muon peak learning rate from 4×10^{-3} to 2×10^{-3}. Vanilla degrades from 3.0078 to 3.0288 (+0.021 nats) and TMMFormer from 2.9342 to 2.9634 (+0.029); the architectural gap shrinks from 0.074 to 0.066 ($\approx$11% compression). The sweep is partial: only the $\times0.5$ Muon-LR cell ran for each architecture; the $\times2$ Muon-LR cells and the four pure-AdamW LR cells of the originally designed $2\times2\times2$ grid were submitted and then cancelled, and seed count is N=1. The robustness claim is therefore that the TMMFormer>Vanilla ordering is preserved under halving the Muon LR, not across a full LR grid.
In both ablations the architectural effect (TMMFormer vs. Vanilla, $\approx$0.07 nats on OWT) is an order of magnitude larger than the optimizer- or LR-induced effect on either architecture, and consistent in sign across both perturbations.

Parameter-matched controls. TMMFormer carries $\approx$39M more parameters than the vanilla 12L/768d backbone (163.8M vs. 124.4M); over 99% of this is the duplicate token+position embedding table that initializes the velocity stream, and the within-block additions (two extra LayerNorms per block) are negligible in both parameter count and per-step FLOPs. To rule out a pure parameter-count effect we train a Vanilla variant whose parameter count matches TMMFormer's, with the optimization recipe of Section E.3 held fixed (TinyStories, 10k steps, seed 42; Muon on 2D weights with peak lr 2×10^{-2} and AdamW on embeddings and LayerNorms with lr 6×10^{-4}; warmup 1k steps followed by cosine decay to a floor of $0.1\times$ peak; gradient clip 1.0; effective batch of 480 sequences of length 1024). The match is single-axis: *width*-900 (12L, d_{model}=900, 12 heads, d_{head}=75, 162.86M—0.6% below TMMFormer) is the clean control. A secondary *depth*-18 run (18L, d_{model}=768, 166.85M) was disrupted by partition-walltime requeues and is reported only as suggestive.
Width-900 Vanilla reaches a best validation loss of 1.1454 on TinyStories, versus 1.1578 for the default Vanilla and 1.1272 for TMMFormer, closing $(1.1578-1.1454)/(1.1578-1.1272)\approx 40\%$ of the TMMFormer–Vanilla gap. The remaining $\approx$0.0182 nats is roughly $9\times$ the per-seed standard deviation reported in the seed-variance arm (≤ 0.0028 at the default sizes), so it is well outside seed

noise. The depth-18 Vanilla appears to close a larger fraction of the gap (best validation ≈1.130); pending a clean rerun we do not draw any conclusion from the width–depth contrast. The headline conclusion stands either way: the bulk of TMMFormer's advantage on TinyStories is *not* explained by parameter count—giving Vanilla TMMFormer's parameter budget, spent as extra width, recovers under half the gap. A full isolation of velocity *dynamics* from the duplicate velocity *embeddings* would require pairing this control with a TMM variant that drops the embedding table but keeps the dynamics (and vice versa); we leave that to future work.

F Preconditioning Redundancy in Pre-Norm Transformers

Adam- or RMSProp-style diagonal preconditioning is most useful when different coordinates of an update have substantially different scales. In a pre-norm Transformer, however, each attention or MLP oracle receives normalized token representations. This does not make every diagonal preconditioner algebraically redundant: in general, $\mathrm{LN}(Dx) \neq \mathrm{LN}(x)$ for a non-scalar positive diagonal matrix D. Rather, LayerNorm motivates a weaker and more useful condition: the oracle outputs may already have nearly balanced coordinate-wise second moments. Under this condition, the Adam- or RMSProp-style token-space preconditioner collapses to an approximate scalar step-size rescaling. For $x \in \mathbb{R}^d$, define the zero-mean LayerNorm map without learned gain or bias by

$$\mathrm{LN}(x) = \frac{x - \bar{x}\mathbf{1}}{\|x - \bar{x}\mathbf{1}\|_2/\sqrt{d}}, \quad \bar{x} = \frac{1}{d}\mathbf{1}^\top x.$$

Balanced moments flatten gains

Theorem F.1. *Consider an Adam- or RMSProp-style diagonal preconditioner*

$$D_s = \mathrm{diag}\left(\frac{1}{\sqrt{s_i} + \delta}\right)_{i=1}^{d},$$

with $\delta \geq 0$, applied to an update direction before the update LayerNorm LN_u used in AdamFormer and RMSPropFormer. Let

$$\rho_i = \sqrt{s_i}, \qquad \rho_{\min} = \min_i \rho_i, \qquad \rho_{\max} = \max_i \rho_i.$$

Assume the second-moment stream is nearly coordinate-balanced:

$$\frac{\rho_{\max}^2}{\rho_{\min}^2} \leq 1 + \epsilon \quad \textit{for some } \epsilon \in [0, 1], \quad \rho_{\min} > 0.$$

Then there exists a scalar $\alpha > 0$ such that

$$\left\| \frac{D_s}{\alpha} - I \right\|_2 \leq \frac{\sqrt{1+\epsilon} - 1}{1 + \delta/\rho_{\max}} \leq \frac{\epsilon}{2}.$$

Consequently, the diagonal Adam- or RMSProp-style preconditioner in this Transformer substep is approximately a scalar multiple of the identity. Since LN_u is invariant to positive scalar rescaling, the scalar part of the preconditioner is absorbed by the update LayerNorm. Thus any nontrivial contribution of diagonal preconditioning must come from the $O(\epsilon)$ non-scalar deviation of D_s from a scalar matrix.

Proof. The diagonal entries of D_s are

$$d_i = \frac{1}{\rho_i + \delta}.$$

Since d_i is decreasing in ρ_i, the largest and smallest diagonal entries are

$$d_{\max} = \frac{1}{\rho_{\min} + \delta}, \qquad d_{\min} = \frac{1}{\rho_{\max} + \delta}.$$

Choose $\alpha = d_{\max}$. Then every normalized diagonal entry satisfies

$$\frac{d_i}{\alpha} = \frac{\rho_{\min} + \delta}{\rho_i + \delta}$$

$$\in \left[\frac{\rho_{\min}+\delta}{\rho_{\max}+\delta}, 1\right].$$

Because $D_s/\alpha - I$ is diagonal,

$$\left\|\frac{D_s}{\alpha} - I\right\|_2 = 1 - \frac{\rho_{\min}+\delta}{\rho_{\max}+\delta} = \frac{\rho_{\max}-\rho_{\min}}{\rho_{\max}+\delta}.$$

The balance assumption gives

$$\rho_{\max} \le \sqrt{1+\epsilon}\,\rho_{\min},$$

or equivalently

$$\rho_{\max} - \rho_{\min} \le \rho_{\max}\left(1 - \frac{1}{\sqrt{1+\epsilon}}\right).$$

Therefore

$$\begin{aligned}\left\|\frac{D_s}{\alpha} - I\right\|_2 &\le \frac{1 - \frac{1}{\sqrt{1+\epsilon}}}{1+\delta/\rho_{\max}} \\ &\le 1 - \frac{1}{\sqrt{1+\epsilon}} \\ &\le \sqrt{1+\epsilon} - 1 \\ &\le \frac{\epsilon}{2},\end{aligned}$$

where the final inequality holds for $\epsilon \in [0,1]$ by concavity of the square-root function. This proves that D_s differs from a scalar multiple of the identity by at most $O(\epsilon)$ in spectral norm.
Finally, both AdamFormer and RMSPropFormer apply LN_u after the diagonal preconditioner. For any positive scalar c, $\mathrm{LN}_u(cz) = \mathrm{LN}_u(z)$ up to the fixed numerical epsilon in the LayerNorm denominator. Thus the scalar component of D_s cannot provide an independent update direction after LN_u. □

Remark F.2 (Interpretation). The balance assumption is natural in this setting because the Adam- or RMSProp-style second-moment stream is driven by oracle outputs $q = \mathcal{O}_\ell(\mathrm{LN}(x))$, rather than raw, unnormalized token states. The oracle input is normalized, the coordinate projections are dense and learned, and the final adaptive update is again normalized by LN_u before entering the residual stream. These architectural features make large persistent coordinate-scale disparities less central than in parameter-space optimization, where Adam is most useful.
The theorem shows that, under this balanced-second-moment condition, the Adam- or RMSProp-style diagonal preconditioner in token space is nearly a scalar matrix. Since LN_u removes positive scalar rescalings of the update, the scalar part of the preconditioner does not create a new residual direction. Any useful effect must therefore come from the small non-scalar deviation of D_s from a scalar multiple of the identity.

G Momentum as Second-Order Residual-Stream Filtering

We now give a local explanation for why momentum-stream Transformer variants can outperform a vanilla pre-norm Transformer at matched depth. The point is architectural: a momentum stream changes the forward residual dynamics from a first-order recurrence to a second-order recurrence. In a linearized local model, this gives a richer polynomial filter over token-feature modes.

G.1 Local Linearized Sandbox

Let the hidden state at layer ℓ be

$$X_\ell \in \mathbb{R}^{T\times d},$$

and let $X^\star \in \mathbb{R}^{T\times d}$ be a task-relevant hidden representation. We assume a local quadratic surrogate energy

$$\mathcal{F}(X) = \frac{1}{2}\langle X - X^\star,\, H(X - X^\star)\rangle,$$

where H is self-adjoint positive definite on token-embedding space, with

$$0 < \mu \le \lambda_i(H) \le L, \qquad \kappa = \frac{L}{\mu}.$$

In the clean sandbox case, the combined Transformer oracle is aligned with the negative gradient of this surrogate:

$$G(X) = -H(X - X^\star). \tag{2}$$

The language-modeling logits are produced by the shared output head

$$Z(X) = XW^\top,$$

where $W \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the token embedding/output matrix and $|\mathcal{V}|$ is the vocabulary size. Thus $Z(X) \in \mathbb{R}^{T\times|\mathcal{V}|}$. and the language-modeling loss is

$$\mathcal{L}_{\mathrm{LM}}(X) = \mathrm{CE}(Z(X), Y).$$

Assume that, near $X^\star$, this loss is C-smooth as a function of the hidden representation and that $X^\star$ is a local minimizer. Then

$$\mathcal{L}_{\mathrm{LM}}(X) - \mathcal{L}_{\mathrm{LM}}(X^\star) \le \frac{C}{2}\|X - X^\star\|_F^2. \tag{3}$$

G.2 Vanilla as a First-Order Filter

Let $E_\ell = X_\ell - X^\star$. In the sandbox, a vanilla residual step is

$$X_{\ell+1} = X_\ell - \eta H(X_\ell - X^\star),$$

so

$$E_{\ell+1} = (I - \eta H)E_\ell.$$

If $H = Q\Lambda Q^\top$, each eigenmode evolves independently:

$$e_{\ell+1,i} = (1 - \eta\lambda_i)e_{\ell,i}.$$

After N layers,

$$E_N = p_N(H)E_0, \qquad p_N(\lambda) = (1 - \eta\lambda)^N.$$

Thus vanilla implements a first-order polynomial filter over the local token-feature spectrum.

Best uniform vanilla contraction

Lemma G.1. *For $\lambda \in [\mu, L]$, the best fixed scalar step size for the vanilla update is*

$$\eta^\star = \frac{2}{L + \mu}.$$

The resulting worst-case contraction factor is

$$\rho_{\mathrm{vanilla}} = \max_{\lambda\in[\mu,L]} |1 - \eta^\star\lambda| = \frac{\kappa - 1}{\kappa + 1}.$$

Consequently,

$$\|E_N^{\mathrm{vanilla}}\|_F \le \rho_{\mathrm{vanilla}}^N \|E_0\|_F.$$

Proof. For fixed η, the worst-case contraction over $[\mu, L]$ is

$$\max\{|1 - \eta\mu|, |1 - \eta L|\}.$$

The optimum equalizes the endpoint magnitudes:

$$1 - \eta\mu = -(1 - \eta L),$$

which gives $\eta^\star = 2/(L + \mu)$. Substitution yields

$$1 - \eta^\star\mu = \frac{L - \mu}{L + \mu} = \frac{\kappa - 1}{\kappa + 1}.$$

Since H is self-adjoint positive definite, the operator norm of $I - \eta^\star H$ is the maximum absolute eigenvalue over the interval. Iterating gives the result. □

G.3 Momentum as a Second-Order Filter

Consider a linearized single-oracle momentum substep:

$$\begin{aligned}\widetilde{X}_\ell &= X_\ell + aV_\ell,\\ V_{\ell+1} &= bV_\ell - \eta H(\widetilde{X}_\ell - X^\star),\\ X_{\ell+1} &= X_\ell + cV_{\ell+1}.\end{aligned}$$

This family includes heavy-ball updates ($a = 0$), the YuriiFormer Nesterov-style lookahead update from prior work ($a > 0, c = 1$) (Zimin et al., 2026), and TMM-style updates ($a > 0$) with learned c. Because V_ℓ stores information from previous residual updates, each eigenmode follows a second-order recurrence. Indeed, for an eigenmode with eigenvalue λ, write the scalar error and velocity as e_ℓ and v_ℓ. Then

$$\begin{aligned}v_{\ell+1} &= bv_\ell - \eta\lambda(e_\ell + av_\ell)\\ &= -\eta\lambda e_\ell + (b - a\eta\lambda)v_\ell,\\ e_{\ell+1} &= e_\ell + cv_{\ell+1}.\end{aligned}$$

When $c > 0$, $v_\ell = (e_\ell - e_{\ell-1})/c$, and therefore

$$\begin{aligned}e_{\ell+1} &= (1 - c\eta\lambda)e_\ell + (b - a\eta\lambda)(e_\ell - e_{\ell-1})\\ &= \alpha(\lambda)e_\ell - \theta(\lambda)e_{\ell-1},\end{aligned}$$

where

$$\begin{aligned}\alpha(\lambda) &= 1 + b - (a + c)\eta\lambda,\\ \theta(\lambda) &= b - a\eta\lambda.\end{aligned}$$

Thus, in general,

$$e_{\ell+1,i} = \alpha(\lambda_i)e_{\ell,i} - \theta(\lambda_i)e_{\ell-1,i},$$

for coefficients determined by (a, b, c, η). Therefore

$$E_N = q_N(H)E_0,$$

where q_N is generated by a second-order recurrence. This is a richer filter family than the vanilla filter $(1 - \eta\lambda)^N$.

Momentum improves contraction

Lemma G.2. *If $\kappa > 1$, there exist stable momentum coefficients such that*

$$\rho_{\text{mom}} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} < \frac{\kappa - 1}{\kappa + 1} = \rho_{\text{vanilla}}.$$

Thus a second-order momentum recurrence can have a strictly better finite-depth worst-case contraction factor than the vanilla recurrence.

Proof. It suffices to exhibit one stable momentum choice. For the quadratic energy $\mathcal{F}(x) = \frac{1}{2}x^\top Hx$, choose the classical heavy-ball parameters

$$\begin{aligned}\eta_{\text{HB}} &= \frac{4}{(\sqrt{L} + \sqrt{\mu})^2},\\ \beta_{\text{HB}} &= \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}\right)^2.\end{aligned}$$

For each eigenmode $\lambda \in [\mu, L]$, the recurrence is

$$e_{\ell+1} = (1 - \eta_{\text{HB}}\lambda + \beta_{\text{HB}})e_\ell - \beta_{\text{HB}}e_{\ell-1}.$$

The characteristic polynomial is

$$r^2 - (1 - \eta_{\text{HB}}\lambda + \beta_{\text{HB}})r + \beta_{\text{HB}} = 0.$$

Classical Chebyshev semi-iterative analysis gives a mode-wise bound with rate

$$\sqrt{\beta_{\mathrm{HB}}} = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}} = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}.$$

Thus, for a constant C_{mom} depending on the initial velocity,

$$\|E_N^{\mathrm{mom}}\|_F \le C_{\mathrm{mom}} \rho_{\mathrm{mom}}^N \|E_0\|_F.$$

It remains to compare the factors. Let $s = \sqrt{\kappa} > 1$. Then

$$\begin{aligned}\rho_{\mathrm{vanilla}} - \rho_{\mathrm{mom}} &= \frac{s^2-1}{s^2+1} - \frac{s-1}{s+1} \\ &= \frac{2s(s-1)}{(s^2+1)(s+1)} > 0.\end{aligned}$$

Hence $\rho_{\mathrm{mom}} < \rho_{\mathrm{vanilla}}$. □

Corollary G.3 (TMM contains the YuriiFormer update in the linearized class)**.** *In the linearized eigenmode model, the polynomial family realizable by TMM contains the polynomial family realizable by the YuriiFormer update. Therefore*

$$\inf_{q\in\mathcal{Q}_N^{\mathrm{TMM}}} \max_{\lambda\in[\mu,L]} |q(\lambda)| \le \inf_{q\in\mathcal{Q}_N^{\mathrm{Yurii}}} \max_{\lambda\in[\mu,L]} |q(\lambda)|.$$

Proof. YuriiFormer fixes the residual reinjection coefficient to $c = 1$, while TMMFormer learns $c = \nu_\ell$. Setting $\nu_\ell = 1$ in TMMFormer recovers the YuriiFormer update. Thus every linearized polynomial filter attainable by YuriiFormer is also attainable by TMMFormer. Taking the infimum over a larger class cannot increase the worst-case spectral error. □

G.4 From Representation Error to Local LM Loss

In this subsection, N counts full Transformer layers. Each layer is modeled by its effective linearized residual update, abstracting over the internal attention–MLP Lie–Trotter substeps.

Momentum lowers the local loss bound

Theorem G.4. *Under the local linearized assumptions above, let X_N^V be the representation produced by a vanilla Transformer after N layers. Let X_N^M be the representation produced by a momentum Transformer after N layers, where one layer is represented by the corresponding effective linearized residual update. If both models start from the same X_0, then*

$$\begin{aligned}&\mathcal{L}_{\mathrm{LM}}(X_N^V) - \mathcal{L}_{\mathrm{LM}}(X^\star) \le \frac{C}{2}\rho_{\mathrm{vanilla}}^{2N}\|X_0 - X^\star\|_F^2, \\ &\mathcal{L}_{\mathrm{LM}}(X_N^M) - \mathcal{L}_{\mathrm{LM}}(X^\star) \\ &\le \frac{C}{2}C_{\mathrm{mom}}^2\rho_{\mathrm{mom}}^{2N}\|X_0 - X^\star\|_F^2.\end{aligned}$$

Since $\rho_{\mathrm{mom}} < \rho_{\mathrm{vanilla}}$, for any fixed $C_{\mathrm{mom}} < \infty$ there exists

$$N_0 = \max\left\{0, \left\lceil \frac{\log C_{\mathrm{mom}}}{\log(\rho_{\mathrm{vanilla}}/\rho_{\mathrm{mom}})} \right\rceil\right\}$$

such that for all $N \ge N_0$, the momentum upper bound is strictly lower than the vanilla upper bound.

Proof. By the smoothness assumption in (3),

$$\mathcal{L}_{\mathrm{LM}}(X) - \mathcal{L}_{\mathrm{LM}}(X^\star) \le \frac{C}{2}\|X - X^\star\|_F^2.$$

Because N counts full layers, applying the vanilla layer contraction from Lemma G.1 for N iterations gives

$$\|X_N^V - X^\star\|_F \le \rho_{\mathrm{vanilla}}^N \|X_0 - X^\star\|_F.$$

Substituting this representation-error bound into (3) yields the vanilla LM-loss bound.
Similarly, applying the effective momentum layer contraction from Lemma G.2 for N layers gives

$$\|X_N^M - X^\star\|_F \le C_{\mathrm{mom}}\rho_{\mathrm{mom}}^N \|X_0 - X^\star\|_F.$$

Substituting this into (3) yields the momentum LM-loss bound. The momentum bound is lower than the vanilla bound whenever

$$C_{\mathrm{mom}}^2 \rho_{\mathrm{mom}}^{2N} < \rho_{\mathrm{vanilla}}^{2N},$$

equivalently

$$C_{\mathrm{mom}} < \left(\frac{\rho_{\mathrm{vanilla}}}{\rho_{\mathrm{mom}}}\right)^N.$$

Because $\rho_{\mathrm{vanilla}}/\rho_{\mathrm{mom}} > 1$, the threshold N_0 above is sufficient for this inequality to hold. □

Remark G.5 (Interpretation). The assumptions are natural as a local model of residual-stream dynamics. First, a Transformer used at inference follows a fixed trajectory of hidden states, so linearizing each layer around the states it actually visits is the standard first approximation to its local behavior. The resulting Jacobian has token-feature modes with different effective rates, which is exactly the situation where finite-depth residual updates can leave slow modes under-corrected. Second, replacing this local operator by a self-adjoint positive definite surrogate isolates the aligned component of the layer oracle: the part that moves the representation toward a task-relevant state $X^\star$. This is the favorable case for vanilla; if momentum improves even there, the advantage is architectural rather than an artifact of a hostile oracle. Third, the smooth readout assumption is natural because the final logits are linear in the hidden state and cross-entropy is smooth on bounded-logit neighborhoods. Near a local minimizer $X^\star$, smaller representation error therefore gives a smaller local upper bound on language-modeling loss.
The theorem explains the empirical advantage of momentum-stream architectures, including the prior YuriiFormer baseline and TMMFormer, over vanilla as a forward-architecture effect. Momentum does not merely change how parameters are trained; it changes the residual stream from a first-order map $X_\ell \mapsto X_{\ell+1}$ to a second-order map $(X_\ell, V_\ell) \mapsto (X_{\ell+1}, V_{\ell+1})$. In the local linearized regime, this gives a faster filter for slow token-feature modes. The TMM-vs-YuriiFormer statement is an expressivity containment result: TMMFormer can recover YuriiFormer by setting $\nu_\ell = 1$, while learning ν_ℓ gives a larger second-order filter class.

H Loss-Landscape Sharpness Measurement

This appendix gives the precise definitions of the sharpness diagnostics summarized in the *Loss Landscape* setup. For a variant with trained parameters $\theta \in \mathbb{R}^N$ we write $\mathcal{L}(\theta)$ for the mean token-level cross-entropy on a fixed set of B held-out validation minibatches, and $H = \nabla_\theta^2 \mathcal{L}(\theta)$ for its Hessian. The matrix H is never instantiated; every quantity below uses only Hessian–vector products (HVPs). All quantities are evaluated at the best (lowest validation loss) checkpoint, on the corpus the model was trained on (OpenWebText unless stated otherwise). The B minibatches are sampled once with a fixed random seed and reused for every variant and every probe, so all models are compared on identical inputs.

Hessian–vector products. For a vector $v \in \mathbb{R}^N$ the HVP is computed by double backward (the Pearlmutter trick):

$$Hv \;=\; \nabla_\theta\big(\langle\, \nabla_\theta \mathcal{L}(\theta),\, v\,\rangle\big), \tag{4}$$

i.e. a first backward pass produces $g = \nabla_\theta \mathcal{L}$ with the computation graph retained, and a second backward pass through the scalar $\langle g, v\rangle$ yields Hv. The loss is evaluated with the math attention kernel, because the fused/flash attention kernels do not support the required double backward. Each HVP is averaged over the B fixed validation minibatches.

Top Hessian eigenvalue. The dominant curvature $\lambda_{\max}(H)$ is estimated by power iteration on the HVP operator (Yao et al., 2020):

$$v_0 \sim \mathcal{N}(0, I_N), \quad \lambda^{(k)} = v_k^\top H v_k, \quad v_{k+1} = \frac{Hv_k}{\|Hv_k\|},$$

run for at most T_{pow} iterations and stopped early once $|\lambda^{(k)} - \lambda^{(k-1)}|/|\lambda^{(k)}| < \tau$. A large $\lambda_{\max}$ indicates a sharp direction in parameter space.

Hessian trace. The trace is estimated with Hutchinson's estimator using Rademacher probes $v^{(j)} \in \{-1, +1\}^N$ (Yao et al., 2020):

$$\mathrm{tr}(H) \;\approx\; \frac{1}{P}\sum_{j=1}^{P} v^{(j)\top} H\, v^{(j)}, \qquad v_i^{(j)} \sim \mathrm{Unif}\{-1, +1\},$$

which is unbiased because $\mathbb{E}\big[vv^\top\big] = I_N$ for Rademacher v. We report the mean and standard deviation over P probes. The quantity compared across variants is the scale-normalized trace $\mathrm{tr}(H)/N$, i.e. the mean curvature per parameter.

Filter-normalized loss curve. Following Li et al. (2018), we probe a random direction that is normalized per parameter tensor, which removes the spurious scale invariance of (pre-norm) networks. For each weight tensor $\theta^{(l)}$ we draw $d^{(l)} \sim \mathcal{N}(0, I)$ and rescale

$$d^{(l)} \;\leftarrow\; d^{(l)}\,\frac{\|\theta^{(l)}\|}{\|d^{(l)}\|}, \tag{5}$$

then evaluate the loss along $\phi(\alpha) = \mathcal{L}\big(\theta + \alpha\, d\big)$ on an evenly spaced grid of G values $\alpha \in [-\alpha_{\max}, \alpha_{\max}]$, restoring θ afterwards. We summarize flatness by the loss range $\max_\alpha \phi(\alpha) - \min_\alpha \phi(\alpha)$; because d is filter-normalized, this range is comparable across models of different scale.

Hyperparameters. Unless stated otherwise we use power iteration with $T_{\mathrm{pow}} = 15$ and tolerance $\tau = 10^{-3}$, $P = 10$ Hutchinson probes, and a curve grid of $G = 11$ points over $\alpha \in [-0.5, 0.5]$. The B validation minibatches are drawn with a fixed seed and shared across all variants. Lower $\lambda_{\max}$, lower $\mathrm{tr}(H)/N$, and a smaller curve range each indicate a flatter loss landscape.

I Forgetting and Generalization Measurement

This appendix gives the precise protocol for the forgetting and generalization diagnostics summarized in the *Forgetting and Generalization* setup. All losses are mean token-level cross-entropy, evaluated on a fixed set of validation minibatches drawn with a shared random seed so that every variant is scored on identical inputs.

Forgetting via sequential fine-tuning. Let $\mathcal{L}_S$ be the validation loss on the *source* corpus (the pretraining corpus). We fine-tune the pretrained checkpoint on a *target* corpus and measure the source-corpus loss before fine-tuning (T_0) and after (T_1), and define

$$\text{forgetting} = \mathcal{L}_S(T_1) - \mathcal{L}_S(T_0), \tag{6}$$

the rise in the source-corpus loss caused by adapting to the target; lower is better retention. We run both directions, OpenWebText→TinyStories and TinyStories→OpenWebText.

To isolate the *architecture* from the pretraining optimizer, every variant is fine-tuned with the same fixed AdamW optimizer ($\mathrm{lr} = 10^{-4}$, weight decay 0.01, $\beta = (0.9, 0.95)$) for 1000 steps, regardless of the optimizer used during pretraining. The learning rate follows a linear warmup over W steps and then a cosine decay to a floor of 0.1 of the peak,

$$\mathrm{lr}(t) = \begin{cases} \dfrac{t}{W}, & t < W, \\[2ex] 0.1 + 0.9 \cdot \frac{1}{2}\left(1 + \cos\dfrac{\pi(t-W)}{T-W}\right), & t \geq W, \end{cases} \tag{7}$$

with T the total number of fine-tuning steps. Source and target losses are evaluated every 100 steps on 50 fixed validation minibatches (seed 0) under `bfloat16` autocast.

Zero-shot cross-corpus generalization. Generalization is measured without any fine-tuning: the OpenWebText checkpoint is evaluated directly on out-of-distribution corpora—WikiText-103 (validation), LAMBADA (OpenAI test split), and C4 (English validation)—as well as on in-distribution OpenWebText validation as a reference. For a corpus with mean cross-entropy $\overline{\mathrm{ce}}$ the perplexity is

$$\mathrm{ppl} = \exp\big(\overline{\mathrm{ce}}\big), \tag{8}$$

computed over a fixed set of minibatches (seed 0, identical across variants). The out-of-distribution score reported in the main text is the mean perplexity over the three out-of-distribution corpora; lower is better.

Detailed results. Table 9 reports the per-direction forgetting and per-corpus zero-shot perplexity behind Figure 3(b–d), for the four variants discussed in the main text. The broad ordering is consistent across columns: the momentum variants (TMMFormer, YuriiFormer) forget the least and attain the lowest perplexity on the out-of-distribution corpora, AdamFormer is intermediate, and the vanilla transformer is worst, matching the flatness pattern of Section H. Within the two momentum variants, YuriiFormer is slightly better on most robustness columns, while TMMFormer is better on pretraining validation loss in the main results.

Forgetting and zero-shot generalization

Variant	Forgetting ↓		Perplexity ↓				
	O→T	T→O	OWT	WT-103	LMB	C4	OOD
Vanilla	0.83	1.12	20.11	61.58	43.19	39.73	48.17
AdamFormer	0.77	0.99	19.78	59.31	43.13	39.56	47.33
YuriiFormer	0.67	0.95	18.81	51.78	41.55	38.92	44.08
TMMFormer	0.69	0.93	18.69	53.77	41.78	38.70	44.75

Table 9: Forgetting (source-corpus loss increase, both transfer directions; O=OpenWebText, T=TinyStories) and zero-shot perplexity on in-distribution OpenWebText and the out-of-distribution corpora (WikiText-103, LAMBADA, C4) with their average (OOD). Lower is better throughout.

J Learning-Rate Schedule and Sharpness-Aware Minimization

The interventions of the corresponding subsection change only TMMFormer's parameter-training recipe; the architecture, data, and total step budget ($T = 30{,}000$ on OpenWebText) are held fixed.

Warmup–stable–decay (WSD) schedule. The default recipe is warmup–cosine. The WSD alternative keeps a constant peak learning rate through most of training and decays only at the end. With warmup $W = 3{,}000$ steps, decay start $D = 25{,}000$, and floor $\eta_{\min} = 0.1$ of the peak, the learning-rate multiplier is

$$\mathrm{lr}(t) = \begin{cases} t/W, & t < W, \\ 1, & W \le t < D, \\ 1 - (1 - \eta_{\min}) \dfrac{t - D}{T - D}, & t \ge D. \end{cases} \tag{9}$$

Sharpness-Aware Minimization (SAM). SAM (Foret et al., 2020) replaces the gradient at θ with the gradient at the worst-case point in a ρ-ball. Each step uses two forward/backward passes on the *same* minibatch: the first yields $g = \nabla_\theta \mathcal{L}(\theta)$ and the ascent perturbation

$$\epsilon^\star = \rho \frac{g}{\|g\|_2}, \qquad \rho = 0.05, \tag{10}$$

and the second evaluates $\nabla_\theta \mathcal{L}(\theta + \epsilon^\star)$, which is used for the parameter update before $\epsilon^\star$ is undone. This is true SAM (the minibatch is shared across both passes), not the m-sharpness variant, and roughly doubles the per-step cost.

SAWD. SAWD uses the WSD schedule of (9) and turns SAM on only during the decay phase ($t \ge D$). Since SAM then runs for the final $\approx 1/6$ of training, the overhead is $\approx 17\%$ rather than $\approx 2\times$. All sharpness statistics for these runs use the diagnostic of Appendix H.