

Kairos: A Compute-Efficient Hybrid MoE Diffusion Architecture for Multimodal Edge Foundation Models

A Proof-of-Concept Report

Fabien Furfaro

August 31, 2026

Abstract

Sparse Mixture-of-Experts (MoE) routing is usually sold as an almost-free lunch: keep the total parameter count high, activate only a small fraction per token, and pay only for what you use. We show this promise quietly breaks down during *training* on memory-bandwidth-limited hardware. Kairos is a proof-of-concept multimodal model combining bidirectional gated DeltaNet linear attention with bidirectional sliding-window attention through shared projections (LiZAttention2, TPTT-inspired), a block-wise Attention-Residual aggregator, a sparse MoE feed-forward layer, and a masked-diffusion training objective over a shared byte-level latent space spanning text, image, audio, video, lidar and robot-control modalities. The current, runnable configuration is a $\sim 14\text{M}$ -parameter sanity-check network (4 layers, $d_{\text{model}} = 64$, 7 experts, top-1 routing) used to validate the pipeline end-to-end via overfitting and per-modality loss checks; a separate, not-yet-trained 200M-total/25M-active design point is used only to size the target compute budget. Using a roofline model, we show that this target design point sits deep in the memory-bound regime ($\mathcal{I}_{\text{train}} = 1.5$ FLOP/byte against ridge points of 156–295 FLOP/byte on A100/H100-class hardware), and that its MoE backward pass is bandwidth-gated by the *total* parameter count rather than the *active* one – a distinction absent from FLOP-centric treatments of MoE efficiency. We give the exact, code-derived algorithms for the hybrid attention block, the Block-AttnRes aggregator, and the masked-diffusion curriculum, and we specify – ahead of running them – the tables the codebase’s own instrumentation (`summary`, `memory_report`, `profile`, `overfit_test`, `check_per_modality_loss`) is built to produce. No full training run has been performed; this report is a design and compute-budget document, not a benchmark paper.

1 Introduction

1.1 Background

Small, edge-deployable language models (MobileLLM [Liu et al., 2024], SmolLM2 [Allal et al., 2025], Gemma-3 [Gemma Team, 2025]) are now routinely over-trained well past the Chinchilla-optimal token-to-parameter ratio [Hoffmann et al., 2022], on the premise that inference cost dominates total cost of ownership. That premise says nothing about training cost when the training hardware is *itself* constrained – the situation this report is concerned with. Three architectural levers target training-time efficiency in largely separate literatures: (i) sub-quadratic sequence mixing – local windowed attention [Beltagy et al., 2020, Jiang et al., 2023] and linear state-compressing mixers derived from state-space models (Mamba [Gu and Dao, 2023]) and the delta rule [Yang et al., 2024b,a]; (ii) sparse capacity scaling via MoE routing [Shazeer et al., 2017, Fedus et al., 2021, Dai et al., 2024, DeepSeek-AI, 2024]; (iii) non-autoregressive diffusion training [Austin et al., 2021, Lou et al., 2023, Sahoo et al.,

2024, Nie et al., 2025], which generalizes masked language modeling [Devlin et al., 2018] into a full generative objective.

1.2 Problem statement

Each lever reports efficiency in its own currency – asymptotic sequence-length complexity, FLOPs-per-token, or generation parallelism – rarely all three, and rarely against the memory-bandwidth limits of the edge-class GPUs that motivate combining them. FLOP-centric MoE accounting implicitly assumes a bandwidth-favorable forward pass implies a bandwidth-favorable *training step*; Section 4 shows this fails once the backward pass is accounted for, because gradients flow through every expert reachable across a batch, not the subset any one token selected. **Research question:** for a hybrid linear-attention/MoE/diffusion model targeting edge-class training hardware, what governs achievable throughput – FLOPs or bandwidth – and which parameter count, active or total, is the correct lever to reduce?

1.3 Contributions

1. The Kairos architecture, specified directly from its implementation: LiZAttention2 (Algorithm 1), Block-AttnRes (Algorithm 2), sparse MoE (§3.4), and the masked-diffusion curriculum (Algorithm 3).
2. An explicit split between the *running* proof-of-concept configuration ($\sim 14\text{M}$ parameters, used for pipeline sanity checks) and the *target* design point (200M total / 25M active, used only for compute-budget sizing) – Table 1.
3. A roofline-based compute analysis of the target design point (§4) showing it is memory-bound and that its MoE backward pass scales with total, not active, parameters.
4. Result tables the codebase’s instrumentation is built to fill (§5), reported here as structure, not numbers, since no full run has been executed.

2 Related Work

Efficient attention. The delta rule [Yang et al., 2024b,a] improves on plain linear attention with an error-correcting state update; Kairos runs it *bidirectionally* (forward pass plus a time-reversed pass, concatenated) rather than causally, matching the bidirectional context a masked-diffusion objective requires. Sliding-window attention [Beltagy et al., 2020, Jiang et al., 2023] is likewise run bidirectionally here, with rotary position embeddings [Su et al., 2021] and grouped-query attention [Ainslie et al., 2023]. Coupling a delta-rule branch and a window-attention branch through shared Q/K/V/O projections follows the TPTT construction [Furfaro, 2025].

Sparse MoE. MoE architectures [Shazeer et al., 2017, Fedus et al., 2021, Dai et al., 2024, DeepSeek-AI, 2024] decouple capacity from per-token compute. Kairos’s MoE layer subclasses Qwen2’s dense MLP [Qwen Team et al., 2024] for the non-routed path and DeepSeek-V3’s routed-expert layer [DeepSeek-AI, 2024] directly, rather than reimplementing either. The training-time memory-bandwidth cost of the router’s backward pass on bandwidth-starved hardware – central to §4 – is not, to our knowledge, separately quantified in this literature.

Diffusion-style language modeling. Kairos follows the MDLM [Sahoo et al., 2024] masking/reweighting formulation but adds a three-stage MAE $\rightarrow$ transition $\rightarrow$ diffusion curriculum (Algorithm 3), motivated by optimization instability observed when sampling the full masking distribution from a random initialization; the cited diffusion-LM literature typically does not stage this.

Scaling laws. Chinchilla [Hoffmann et al., 2022] is fitted on 70M–16B *dense* models; extrapolating to a 4M-active-parameter MoE model is out of range. Joint MoE scaling laws [Ludziejewski et al., 2025] instead parameterize loss in active parameters, tokens, and expert count, and report that the marginal value of tokens grows with expert count – relevant to §4’s token budget. A separate line analyzes the FLOPs-vs-sparsity trade-off directly [Abnar et al., 2025]; we build on it to isolate the total-vs-active distinction specifically in the backward pass.

3 Architecture

3.1 Two configurations, not one

The codebase runs at two distinct scales that this report is careful not to conflate. The **PoC configuration** is the default in the project’s marimo notebook (`notebook/kairos_pretraining.py`): a ~ 14 – 15 M-parameter network used to sanity-check the pipeline via memorization (`overfit_test`) and per-modality loss checks. Its 7-expert, top-1 MoE is deliberately small – the notebook notes that top-1 routing converges slowly on tiny overfit runs, so the expert count is kept low to make the sanity check tractable at all. The **target configuration** is the 200M-total / 25M-active design point used in §4 purely to size a future training run; it has not been instantiated or trained. Table 1 gives both.

Parameter	PoC (running)	Target (design point, untrained)
d_{model}	64	–
Attention heads (head dim)	4 (16)	–
Backbone layers	4	–
Codec scales / stride	4 / 3	–
AttnRes block size	4	–
Routed experts E / active k	7 / 1	32 / 4
Shared experts	1	–
FFN intermediate size	544	–
Backbone sharing across scales	yes ($\sim 75\%$ param saving)	–
Codec mode	patch (dense per-scale)	–
Cross-session memory bank	enabled	–
N_{total}	~ 14 – 15 M	200M
N_{act}	not yet logged (§5)	25M (4M for §4)
Target training tokens D	16 examples $\times$ 200 steps (overfit)	30B

Table 1: The two configurations this report distinguishes. Dashes mean the target design point has not been fixed at that granularity – only the parameter/token budget that drives §4 has been specified. N_{act} for the PoC config is computed by the same routine used at target scale (Eq. 1) but has not yet been logged from an instantiated model; it is listed as a pending measurement, not estimated here, to avoid quoting an unverified number.

Figure 1 shows the shared data flow; both configurations instantiate the same modules, differing

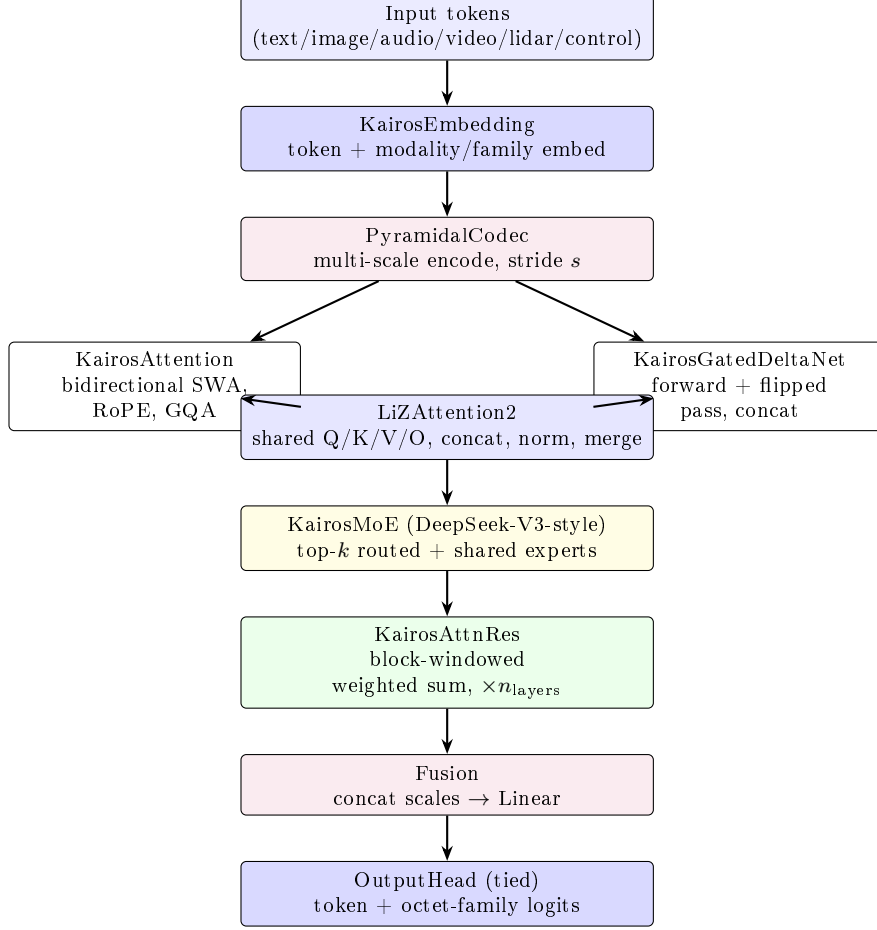


Figure 1: Kairos data flow ($\text{DiffusionBlock} \times n_{\text{layers}}$, wrapped by $\text{KairosDiffusionBackbone}$). Both attention branches see the same input and are bidirectional; the Block-AttnRes aggregator determines what each layer actually reads.

only in the sizes in Table 1.

3.2 LiZAttention2: shared-basis bidirectional hybrid attention

Each layer runs a bidirectional sliding-window branch (KairosAttention , window mask $|q - k| \leq w$ in both directions) and a bidirectional gated-DeltaNet branch ($\text{KairosGatedDeltaNet}$, run once forward and once on the time-reversed sequence, then flipped back and concatenated) over the *same* input. The DeltaNet branch’s Q/K/V/output projections are deleted and replaced by references to the SWA branch’s, so both mixers read the same linear projections of x ; only their mixing rule differs.

3.3 Block-AttnRes aggregation

Rather than a plain residual $x_{i+1} = x_i + f_i(x_i)$, each layer’s *input* is a learned, softmax-weighted mix of the embedding, finalized block-sums of earlier layers, and a running partial sum of the current block (Algorithm 2); the layer’s own attention/FFN sublayers still use ordinary residual connections around h . With `attnres_block_size` = 1 this degenerates to a DenseNet-style sum over all prior

Algorithm 1 LiZAttention2 forward (one layer)

Require: hidden states $x \in \mathbb{R}^{B \times L \times d}$, RoPE table, attention mask

- 1: $q, k, v \leftarrow \text{swa.q_proj}(x), \text{swa.k_proj}(x), \text{swa.v_proj}(x)$ $\triangleright$ shared with DeltaNet
 - 2: $\text{swa_out} \leftarrow \text{BidirWindowAttn}(q, k, v, \text{window } w, \text{RoPE})$
 - 3: $\text{delta_out}_f \leftarrow \text{GatedDeltaRule}(q, k, v)$
 - 4: $\text{delta_out}_b \leftarrow \text{flip}(\text{GatedDeltaRule}(\text{flip}(q, k, v)))$
 - 5: $\text{delta_out} \leftarrow \text{swa.out_proj}(\text{delta_out}_f + \text{delta_out}_b \text{ combination})$
 - 6: $\text{out} \leftarrow \text{concat}(\text{swa_out}, \text{delta_out})$ $\triangleright 2d$ -wide
 - 7: $\text{out} \leftarrow \text{RMSNorm}(\text{out})$
 - 8: **return** $\text{Linear}_{2d \rightarrow d}(\text{out})$
-

individual layer outputs; with block size = `n_layers` it degenerates to a single running sum (closer to a plain residual stream). The PoC configuration uses block size 4 over 4 layers, i.e. one block covering the whole network.

Algorithm 2 Block-AttnRes backbone forward

Require: embedding x_0 , layers $f_1, \dots, f_L$, block size S , learned $w \in \mathbb{R}^d$ per aggregator

- 1: $\text{completed} \leftarrow []$; $\text{partial} \leftarrow \text{None}$; $\text{count} \leftarrow 0$
 - 2: **for** $i = 1, \dots, L$ **do**
 - 3: $\text{sources} \leftarrow [x_0] \cup \text{completed} \cup (\{\text{partial}\} \text{ if } \text{partial} \neq \text{None})$
 - 4: $V \leftarrow \text{stack}(\text{sources})$; $K \leftarrow \text{RMSNorm}(V)$
 - 5: $\text{logits} \leftarrow w^\top K$ (over the stacked/source dimension)
 - 6: $h \leftarrow \sum_{\text{sources}} \text{softmax}(\text{logits}) \cdot V$
 - 7: $x_i \leftarrow f_i(h)$ $\triangleright$ ordinary residual attention + FFN inside f_i
 - 8: $\text{partial} \leftarrow x_i$ if partial is None else $\text{partial} + x_i$; $\text{count} += 1$
 - 9: **if** $\text{count} = S$ **then**
 - 10: $\text{completed.append}(\text{partial})$; $\text{partial} \leftarrow \text{None}$; $\text{count} \leftarrow 0$
 - 11: **end if**
 - 12: **end for**
 - 13: **return** $\text{RMSNorm}(x_L)$
-

3.4 Sparse Mixture-of-Experts

KairosMoE subclasses DeepSeek-V3’s routed-expert layer [DeepSeek-AI, 2024] directly (only the expert-weight initialization is patched, since the upstream implementation leaves expert weights uninitialized); **KairosFFN**, used when MoE is disabled, subclasses Qwen2’s dense MLP [Qwen Team et al., 2024]. Each token is routed to k of E routed experts plus 1 or more always-on shared experts. The PoC configuration ($E = 7, k = 1$) is deliberately under-sparse relative to the target ($E = 32, k = 4$): the notebook records that top-1 routing on a wide expert bank converges too slowly to be useful as a fast sanity check.

The active-parameter count used throughout this report is computed directly, not estimated after the fact:

$$N_{\text{act}} = N_{\text{total}} - N_{\text{expert}} + N_{\text{expert}} \cdot \frac{k}{E}, \quad (1)$$

where N_{expert} sums only parameters under modules whose name contains `".experts."` – shared experts, the router gate, the attention/DeltaNet path, and the codec are treated as fully dense

(every token activates them), matching how `count_active_parameters()` walks the module tree in the codebase.

3.5 Masked-diffusion training objective

Given a clean sequence x_0 , each row samples $p \sim U(\epsilon, p_{\max})$ and a Bernoulli(p) mask over non-prompt, non-pad positions; masked positions are replaced with random tokens (and, where present, random octet-family ids) to form x_t . The model predicts x_0 at masked positions from x_t ; the loss is cross-entropy, optionally reweighted by a continuous blend factor $\alpha \in [0, 1]$ between plain CE ($\alpha = 0$) and full $1/p$ reweighting ($\alpha = 1$), following the MDLM formulation [Sahoo et al., 2024]. Self-conditioning (probability 0.5 in the current config) occasionally feeds a detached, no-grad warm-up prediction back in as extra context, matching how `generate()` uses it at inference time.

Algorithm 3 Masked-diffusion training step (`compute_loss`)

Require: batch x_0 , prompt lengths, pad mask, stage schedule $(p_{\max}^{(t)}, \alpha^{(t)})$ at step t

- 1: $p \leftarrow (p_{\max}^{(t)} - \epsilon) \cdot U(0, 1)^B + \epsilon$ $\triangleright$ one p per row
- 2: $\text{mask} \leftarrow \text{Bernoulli}(p)$ restricted to non-prompt, non-pad positions
- 3: $x_t \leftarrow x_0$ with $x_t[\text{mask}] \leftarrow$ random tokens
- 4: $\text{logits} \leftarrow \text{model}(x_t, \text{logits_mask} = \text{mask})$ $\triangleright$ project only masked positions
- 5: $\ell \leftarrow \text{CE}(\text{logits}, x_0[\text{mask}]); \quad \ell \leftarrow \ell \cdot (1 + \alpha^{(t)}(1/p - 1))$ if $\alpha^{(t)} > 0$
- 6: **return** $\text{mean}(\ell)$ $\triangleright$ + octet-family CE if family ids present

The stage schedule $(p_{\max}^{(t)}, \alpha^{(t)})$ is a pure function of the global step (`stage_mask_schedule`): flat at $(p_{\max}^{\text{MAE}} = 0.3, \alpha = 0)$ for the MAE stage, linearly ramped to $(p_{\max} = 1.0, \alpha = 1)$ over the transition stage, then flat at the diffusion target – chosen so that resuming from a checkpoint depends only on the persisted step count, not on which pipeline call produced it.

4 Compute Budget Analysis (Target Design Point)

This section applies only to the 200M/25M **target** design point of Table 1; it is a sizing exercise for a training run that has not been executed, not a measurement of the running PoC.

4.1 Total training FLOPs

For a dense transformer, $C \approx 6ND$ [Kaplan et al., 2020]; for MoE, compute scales with active, not total, parameters [Ludziejewski et al., 2025]:

$$C_{\text{Kairos}} = 6 \times 4\text{M} \times 30\text{B} = 7.2 \times 10^{17} \text{ FLOPs},$$

roughly $720\times$ cheaper in raw FLOPs than an estimated dense Gemma-3 270M [Gemma Team, 2025] run ($\sim 5 \times 10^{20}$ FLOPs). This comparison alone overstates the win, as the roofline analysis below shows.

4.2 Roofline model

Achievable throughput is $\min(\pi, \mathcal{I} \cdot b_s)$, with ridge point $\mathcal{I}^* = \pi/b_s$ [Williams et al., 2009] (Table 2, Figure 2). Kairos’s training step has $\mathcal{I}_{\text{train}} = 6N_{\text{act}}/4N_{\text{act}} = 1.5 \text{ FLOP/byte}$ – independent of model size, deep in the memory-bound region on every GPU class considered.

GPU	π (TFLOPS)	b_s (TB/s)	Ridge point $\mathcal{I}^*$ (FLOP/byte)
A100	312	2.0	156
H100 SXM5	989	3.35	295

Table 2: Peak compute, HBM bandwidth, ridge points.

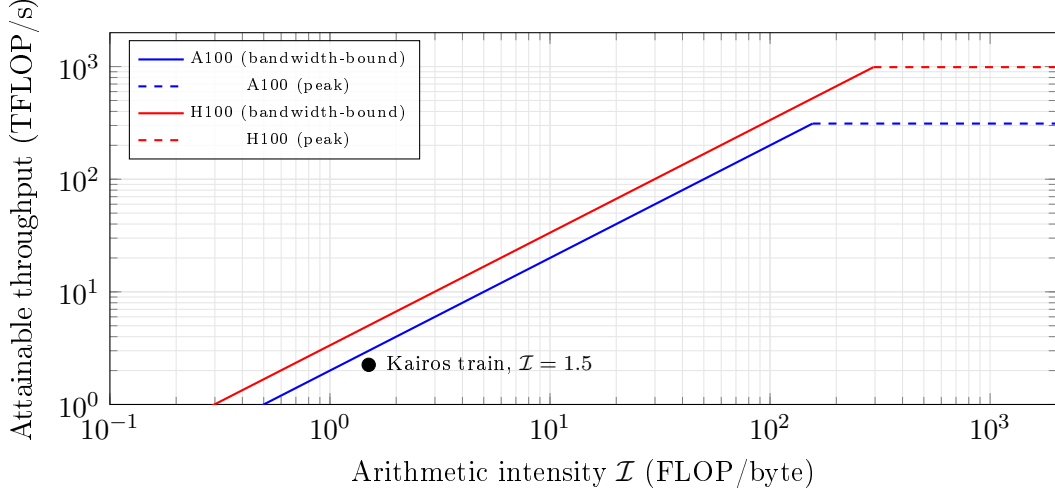


Figure 2: Kairos’s training arithmetic intensity vs. A100/H100 ridge points – deep in the memory-bound regime on both.

4.3 The MoE bandwidth bottleneck

Pass	Bytes/token	Driving factor
Forward	$4N_{\text{act}} = 16\text{MB}$	only active experts + attention read
Backward	$4N_{\text{total}} = 800\text{MB}$	gradients flow through <i>all</i> experts via router
Adam	$8N_{\text{total}}/B$	amortized; negligible for $B \gtrsim 32$

Table 3: Bytes moved per token by pass, target design point.

The backward pass reads N_{total} , not N_{act} , because the router’s gradient depends on every expert’s contribution to the batch loss [Abnar et al., 2025]. As batch size grows,

$$B_{\text{eff}} = 4N_{\text{total}} + \frac{8N_{\text{total}}}{B} \xrightarrow{B \rightarrow \infty} 4N_{\text{total}}.$$

The backward-pass bandwidth cost of a MoE model is governed by its *total*, not active, parameter count – capping the achievable training-time speedup from sparsity even though it is fully realized at inference time.

4.4 Throughput and wall-clock estimates

None of the four classes reach a practical wall-clock budget at the current N_{total} ; T4/RTX 5060Ti – the stated low-compute target – are orders of magnitude over any reasonable budget. The fastest lever is reducing N_{total} (e.g. to $\sim 50\text{M}$) at fixed N_{act} , a $4\times$ cut in backward-pass bytes, before any kernel-level work.

GPU	b_s (GB/s)	tok/s (theory)	tok/s (MFU 25%)	Days for 30B tokens
T4	300	375	~ 94	$\sim 3\,700$
RTX 5060Ti	600	750	~ 187	$\sim 1\,850$
A100 40G	2\,000	2\,500	~ 625	~ 555
H100 SXM5	3\,350	4\,188	$\sim 1\,047$	~ 332

Table 4: Theoretical-upper-bound throughput at $N_{\text{total}} = 200\text{M}$. Not measured: no run has reached this scale. §5 specifies the benchmark that would replace this table with real numbers.

4.5 Scaling-law context

Chinchilla [Hoffmann et al., 2022]’s fit is out of range for a 4M-active-parameter model; joint MoE scaling laws [Ludziejewski et al., 2025] instead predict the marginal value of tokens *grows* with expert count, softening the concern. Table 5 places the target token ratio ($D/N_{\text{act}} = 7,500\times$) against small dense-model precedents – none of which use sparse routing, so they bound plausibility of coherent output at this active-parameter scale but say nothing about the bandwidth bottleneck above.

Model	N_{total}	N_{act}	Tokens	D/N	Notes
Pythia-14M [Biderman et al., 2023]	14M	14M	$\sim 300\text{B}$	$\sim 21,000\times$	dense, $6\text{L}\times\text{d}128$
Pythia-31M [Biderman et al., 2023]	31M	31M	$\sim 300\text{B}$	$\sim 9,700\times$	dense, $6\text{L}\times\text{d}256$
TinyStories-1M [Eldan and Li, 2023]	$\sim 1\text{M}$	$\sim 1\text{M}$	$\sim 3\text{B}$	$\sim 3,000\times$	domain-scoped
TinyStories-33M [Eldan and Li, 2023]	$\sim 33\text{M}$	$\sim 33\text{M}$	$\sim 3\text{B}$	$\sim 90\times$	domain-scoped
Kairos PoC (running)	$\sim 14\text{--}15\text{M}$	TBD	overfit-scale	n/a	MoE, sanity check only
Kairos target (design)	200M	4M	30B	$7,500\times$	MoE, multimodal, untrained

Table 5: Small-model precedents next to both Kairos configurations.

5 Anticipated Results

No training run backs this report. What follows is the exact structure the codebase’s own instrumentation is built to fill – specified here so the eventual results have a pre-committed table to land in, rather than being chosen post hoc.

5.1 Pipeline sanity check (overfit_test)

`overfit_test` trains the PoC network on 16 held-in examples for 200 steps through the compressed MAE $\rightarrow$ transition $\rightarrow$ diffusion curriculum and reports first/last/tail-mean loss. A working pipeline should show near-monotonic decrease within the MAE stage, a visible bump when reweighting switches on during the transition, and convergence toward a low tail loss during diffusion; a pipeline bug typically shows as a plateau at the random-guess cross-entropy ($\ln|\mathcal{V}|$) or a non-finite loss (caught and logged by the circuit breaker in `KairosDiffusionTrainer`).

5.2 Per-modality loss (check_per_modality_loss)

5.3 Instrumented compute report (summary, memory_report, profile)

The pipeline already measures, rather than estimates, step time and memory: `summary(benchmark=True)` runs n real forward + backward + optimizer steps and reports

Stage	Steps	First loss	Tail-mean loss	Status
MAE ($p_{\max} = 0.3, \alpha = 0$)	TBD	TBD	TBD	TBD
Transition (ramp)	TBD	TBD	TBD	TBD
Diffusion ($p_{\max} = 1.0, \alpha = 1$)	TBD	TBD	TBD	TBD

Table 6: Overfit sanity-check table, to be filled from `pipe.overfit_test(n_examples=16, steps=200)`.

Modality	Mean masked CE loss	# batches
Text	TBD	TBD
Image	TBD	TBD
Audio	TBD	TBD
Video	TBD	TBD
Lidar	TBD	TBD
Control (state/action)	TBD	TBD

Table 7: Per-modality loss breakdown, to be filled from `pipe.check_per_modality_loss()`. A large, isolated gap on one modality would flag a codec-scale or tokenizer mismatch for that stream rather than a global optimization problem.

mean step time and derived tokens/s; `memory_report()` reports unique parameter bytes, optimizer-state bytes, and process RSS before/after a step; `profile()` reports per-module wall time. Table 8 is the structure these calls fill; Table 4’s theoretical figures are what this table would need to approach for the target design point to be trainable in practice.

Config	Measured tok/s	Peak mem (GB)	Step time (ms)	Hardware
PoC ($\sim 14\text{--}15\text{M}$)	TBD	TBD	TBD	TBD
Target (200M, projected)	TBD	TBD	TBD	not yet instantiated

Table 8: Real, instrumented benchmark table, to be filled from `pipe.summary(benchmark=True)` and `pipe.memory_report()` once the PoC has run on target hardware.

6 Discussion

For the target design point, bandwidth – not FLOPs – governs achievable training throughput, and N_{total} , not N_{act} , is the lever that sets it, because N_{total} drives the backward pass’s byte traffic. MoE efficiency claims reported on data-center hardware, where bandwidth is generous relative to N_{total} , do not transfer cleanly to bandwidth-starved edge hardware for this reason. Separately, the PoC’s under-sparse routing ($E = 7, k = 1$, versus a $E = 32, k = 4$ target) is itself a data point for §4: it was chosen specifically because wider top-1 routing failed to converge fast enough for a sanity check, which is a qualitative, code-level echo of the router-optimization difficulty discussed in the MoE literature, not yet a quantitative one.

6.1 Limitations

This is a design and compute-budget report, not a benchmark paper: the PoC has not been run to produce Tables 6–8, and Table 4’s figures are theoretical upper bounds for an untrained, uninstantiated target configuration. The Chinchilla extrapolation is out of its fitted range. The roofline model ignores kernel launch overhead, activation recomputation, and multi-GPU communication, all of which would lower real throughput further. Readers are encouraged to check any figure or citation here against its primary source; this analysis is a planning tool for the project, not a peer-reviewed benchmark.

7 Conclusion

Kairos is currently a $\sim 14\text{--}15\text{M}$ -parameter proof of concept exercising a hybrid bidirectional DeltaNet/SWA, MoE, masked-diffusion architecture end-to-end, with a separate 200M/25M design point used only to size a future run. The code-derived algorithms (1–3) and the roofline analysis together give a falsifiable next step ahead of that run: reduce N_{total} at fixed N_{act} , then fill Tables 6–8 with real, instrumented numbers from the PoC before committing to the target scale.

References

- Samira Abnar, Omid Saremi, Mostafa Dehghani, et al. Parameters vs flops: Scaling laws for optimal sparsity for mixture-of-experts language models. *arXiv preprint arXiv:2501.12370*, 2025.
- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.
- Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.
- Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *arXiv preprint arXiv:2107.03006*, 2021.
- Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O’Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*, 2023.
- Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.
- DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

- Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.
- William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.
- Fabien Furfaro. Tptt: Transforming pretrained transformer into titans. *arXiv preprint arXiv:2506.17671*, 2025.
- Gemma Team. Gemma 3 technical report. Google DeepMind, 2025.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*, 2024.
- Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.
- Jan Ludziejewski, Jakub Krajewski, Kamil Adamczewski, Maciej Pi  ro, Micha   Krutul, Szymon Antoniak, Kamil Ciebia, Krystian Kr  l, Tomasz Odrzyg  dz, Piotr Sankowski, and Marek Cygan. Joint moe scaling laws: Mixture of experts can be memory efficient. *arXiv preprint arXiv:2502.05172*, 2025.
- Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.
- Qwen Team, An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

- Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *arXiv preprint arXiv:2406.07524*, 2024.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.
- Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. In *Communications of the ACM*, volume 52, pages 65–76. ACM, 2009.
- Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024a.
- Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024b.