Transformer Encoder Frankenstein: Library, CLI, and Research-Grounded Design Notes

Erick F. Merino M.
This work is not affiliated
erickfmm@gmail.com

February 2026

Abstract

Transformer Encoder Frankenstein presents a unified configuration-driven toolkit for systematic experimentation with modern encoder architectures, spanning seventeen sequence mixer variants and twenty optimizer families. The research contributions are threefold: (i) a strict schema-based configuration contract that enables reproducible experimentation across diverse attention mechanisms, including standard softmax attention, sigmoid attention, retentive networks, selective state-space models, continuous-depth transformers, memory-augmented attention, sparse attention patterns, and gated mechanisms; (ii) a comprehensive optimizer routing framework supporting variance-reduction methods (MARS, Adan, AdEMAMix), memory-efficient variants (Adafactor, GaLore, Lion), schedule-free approaches, and second-order preconditioners (Shampoo, SOAP, Sophia); and (iii) end-to-end workflows spanning quantized deployment via ternary weight packing and sentence-embedding training inspired by SBERT. The toolkit implements a web-based configuration interface that provides schema-driven form rendering with inline documentation and real-time validation. This technical reference document includes architectural diagrams, execution-flow visualizations, decision tables, and comprehensive appendices synthesizing literature on transformer architectures, sparse attention mechanisms, gated attention variants, and optimization algorithms. The system enables rapid iteration while maintaining reproducible experimental conditions through its schema-first design philosophy.

Contents

1 Introduction

1.1 Motivation and Problem Statement

Transformer architectures have fundamentally transformed the landscape of sequence modeling across natural language processing [38], computer vision, and computational biology. The success of models such as BERT [10], GPT, and their variants has established the transformer as the de facto standard for representation learning in deep learning. However, the rapid proliferation of architectural innovations presents significant practical challenges for researchers and practitioners.

Recent years have witnessed an explosion of alternative attention and sequence-mixing mechanisms, each addressing specific limitations of standard softmax attention: quadratic computational complexity [8], memory-efficient inference requirements [36, 11], content-based selective state management [2], and hardware-aware optimizations [30]. Simultaneously, the optimization literature has diversified beyond classical AdamW, introducing variance-reduction techniques [42, 26, 48], memory-efficient variants [33, 53], schedule-free approaches [9], and second-order preconditioning methods [13, 39].

The practical consequence is a fragmented research ecosystem where experimental comparison across architectures and optimizers requires significant engineering effort. Researchers must implement and debug multiple variants from scratch, ensure consistent training pipelines, and manage complex hyperparameter spaces. This fragmentation hampers reproducibility, slows scientific progress, and increases the barrier to entry for new researchers.

This work addresses these challenges through a unified, configuration-driven experimentation toolkit that provides:

1. **Schema-First Design**: A strict, validated configuration contract that enforces reproducibility while supporting seventeen distinct sequence mixer architectures and twenty optimizer families.

2. **Architecture Agnostic Training**: Common training infrastructure supporting dense attention baselines (standard, sigmoid), recurrent alternatives (RetNet, Mamba, ODE-style blocks), sparse attention patterns (Sparse Transformer, Longformer, BigBird, SparseK, NSA, SpargeAttn, FASA), and gated mechanisms (GLA, DeltaNet, Gated DeltaNet, HGRN2, FoX, Gated Softmax).

3. **Optimizer Routing Framework**: Prefixed hyperparameter groups enabling fine-grained control over embeddings, normalization layers, recurrent blocks, attention blocks, and other parameter subsets across diverse optimizers.

4. **End-to-End Workflows**: Integrated deployment via quantization (ternary weight packing, INT8 activations) and sentence-embedding capabilities inspired by SBERT [31].

5. **Interactive Configuration**: Web-based interface providing schema-driven form rendering, real-time validation, and CLI command generation.

The project command surface is:

`frankestein-transformer`

with subcommands: `train`, `deploy`, `quantize`, `infer`, `sbert-train`, `sbert-infer`, `web-server`.

The `web-server` command launches a Streamlit-based configuration builder that provides:

- Schema-driven form fields with parameter titles and detailed descriptions
- Real-time tooltips and help text for each configuration option

- Live YAML preview and download functionality
- Generated CLI commands for training, deployment, inference, and SBERT workflows

This interactive interface serves as an alternative to manual YAML editing, improving usability for users exploring available configuration options and understanding their impact on model behavior and training dynamics.

1.2 Contributions

This work makes the following primary contributions:

1. **Unified Configuration Schema**: A YAML-based schema contract with strict validation that supports seventeen distinct sequence mixer architectures across four categories (dense baselines, recurrent/retentive blocks, sparse attention patterns, and gated mechanisms) while enforcing reproducibility through `additionalProperties: false` constraints.
2. **Comprehensive Architecture Support**: Implementation of modern transformer variants including standard softmax attention [38], sigmoid attention [30], RetNet [36], Mamba [11], ODE-style continuous transformers [51], Titans memory-augmented attention [2], sparse attention mechanisms [8, 3, 49, 23, 47, 52, 41], and gated attention architectures [44, 45, 46, 28, 19, 29].
3. **Optimizer Routing Framework**: Prefixed hyperparameter system enabling per-parameter-group control across twenty optimizers, including variance-reduction methods (MARS, Adan, AdEMAMix, Cautious AdamW), memory-efficient variants (Adafactor, GaLore, Lion), schedule-free approaches (Schedule-Free AdamW), curvature-aware methods (Sophia), second-order preconditioners (Shampoo, SOAP), and orthogonality-oriented optimizers (Muon, Turbo-Muon).
4. **Quantization and Deployment**: Integrated deployment pipeline supporting ternary weight packing and INT8 activation quantization with size estimates following 1.58-bit storage approximation.
5. **Sentence-Embedding Workflows**: SBERT-inspired training and inference pipelines supporting similarity scoring, retrieval, clustering, and persistent embedding export.
6. **Interactive Configuration Interface**: Streamlit-based web server providing schema-driven form generation, real-time validation, inline documentation, and CLI command generation.

1.3 Reading Guide

This document is organized as a technical reference addressing four operational concerns:

1. **Configuration Contract**: Section 3.1 describes the YAML schema that enforces valid experiments and Section 3.2 explains validation rules.
2. **Architecture Selection**: Section 3.7 covers normalization options; Section 4.1 provides comprehensive comparison of sequence mixer families; Appendices B, C, and D synthesize supporting literature.
3. **Optimization Dynamics**: Section 5 details optimizer routing and training dynamics; Appendix A provides comprehensive optimizer family analysis.
4. **Deployment and Inference**: Sections 6 and 7 describe quantized deployment and sentence-embedding workflows.

1.4 Web-Based Configuration Builder

In addition to direct YAML editing, this project provides a Streamlit-based web interface (accessed via `web-server` command) that improves configuration accessibility and discoverability. The interface presents schema fields with:

- **Schema-driven form rendering** — All fields are dynamically generated from the authoritative schema, ensuring consistency and validation.
- **Inline parameter documentation** — Each form field displays a *title* from the schema as its label, with the *description* shown as a help tooltip on hover.
- **Real-time configuration preview** — Users see live YAML output as they modify form fields, enabling immediate validation feedback.
- **CLI command generation** — The interface generates complete CLI commands for training, deployment, inference, and SBERT workflows based on current configuration.
- **Accessibility improvements** — Tooltips and structured forms make configuration options easier to understand, especially for users new to the project or exploring novel architectures.

This web-based approach addresses common usability barriers in configuration-driven experimentation:

- Reduces need to memorize YAML structure and field names
- Prevents typos through schema validation
- Provides educational context through inline documentation
- Enables rapid experimentation with guided parameter tuning
- Serves as both a configuration tool and a learning resource

The web interface implementation uses Streamlit's form widgets with:

- `st.checkbox()` for binary toggles with help text
- `st.number_input()` for numeric fields with step size and format
- `st.selectbox()` for enum choices with options display
- `st.multiselect()` for array selections from defined options
- `st.text_input()` for string fields
- `st.info()`, `st.caption()` for supplementary information

Schema metadata (title and description fields) are extracted and rendered systematically across all form sections, including:

- Model architecture parameters (hidden size, layers, attention heads, etc.)
- Training runtime settings (batch size, accumulation, scheduler)
- Optimizer configuration with per-parameter-group hyperparameters
- Deployment and quantization options
- SBERT-specific training and inference parameters

2 Related Work

This work sits at intersection of three active research areas: alternative attention architectures, sparse attention mechanisms, and advanced optimization algorithms. This section synthesizes relevant literature across these domains to contextualize contributions.

2.1 Sequence Mixer Architectures

The trajectory of sequence modeling in artificial intelligence has been shaped by continuous tension between computational expressivity and resource efficiency. The advent of standard transformer attention mechanism fundamentally transformed natural language processing, computer vision, and computational biology by prioritizing global contextualization over sequential recurrence. However, as ambition of foundation models scales toward multimillion-token context windows, the quadratic computational complexity $\mathcal{O}(n^2)$ of traditional self-attention has emerged as severe bottleneck [38].

2.1.1 Dense Attention Baselines

Standard Softmax Attention. The standard multi-head self-attention mechanism established paradigm shift by entirely dispensing with sequential recurrence and localized convolutions. By permitting every token in a sequence to directly attend to every other token, standard transformer achieved unparalleled success in capturing long-range dependencies. The formulation computes a weighted sum of value vectors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

For autoregressive decoding, this mechanism requires Key-Value (KV) caching to circumvent $\mathcal{O}(n^2)$ recomputation. However, this theoretical efficiency comes at cost of linearly growing memory footprint requiring $\mathcal{O}(n \cdot d)$ space, which can consume hundreds of gigabytes of High Bandwidth Memory (HBM) for large models [38].

Sigmoid Attention. Sigmoid attention replaces row-wise softmax normalization with element-wise sigmoid activation:

$$\text{SigmoidAttn}(Q, K, V) = \sigma\left(\frac{QK^\top}{\sqrt{d_k}} + b\right)V$$

Theoretical analyses leveraging mixture-of-experts perspective reveal critical advantages in sample complexity. For models utilizing ReLU experts, softmax gating converges at $\mathcal{O}(n^{-0.24})$, while sigmoid version achieves significantly faster $\mathcal{O}(n^{-0.51})$ convergence. Element-wise nature circumvents cross-thread synchronization overhead, enabling 17% inference kernel speedup via FlashSigmoid on NVIDIA H100 GPUs [30]. However, empirical scaling revealed training instabilities requiring "hybrid-norm" stabilization layer for large-scale models.

2.1.2 Recurrent and Retentive Architectures

RetNet (Retentive Network). RetNet was explicitly engineered to resolve "impossible triangle" of sequence modeling: achieving parallelizable training, low-cost $\mathcal{O}(1)$ inference, and high performance simultaneously. The retention mechanism introduces fixed causal exponential decay matrix D:

$$\text{Retention}(X) = (QK^\top \odot D)V$$

with recurrent form:

$$S_n = \gamma S_{n-1} + k_n^\top v_n, \quad o_n = q_n S_n$$

This dual representation enables parallel training with $\mathcal{O}(n^2 \cdot d)$ complexity and recurrent inference with $\mathcal{O}(1)$ time per step, eliminating KV cache entirely [36].

Mamba (Selective State Space Model). Mamba addresses limitations of Linear Time-Invariant (LTI) state-space models by introducing selectivity to fundamental parameters. The continuous-time system:

$$h'(t) = Ah(t) + Bx(t), \quad y(t) = Ch(t)$$

is discretized with input-dependent parameters:

$$\bar{A}_t = \exp(\Delta_t A), \quad \bar{B}_t = \Delta_t B_t, \quad \Delta_t = \text{softplus}(\text{Linear}(x_t))$$

The hardware-aware scan algorithm enables linear $\mathcal{O}(n \cdot d)$ complexity with efficient GPU utilization, making linear recurrence practical on accelerators [11].

ODE-style Continuous Transformers. The ODE transformer treats depth as numerical integration over continuous dynamical system:

$$\frac{dh(t)}{dt} = f_\theta(h(t), t)$$

with Runge-Kutta refinement reducing truncation error and improving sequence generation quality. This provides more expressive intra-block trajectory with weight sharing, though at higher computational cost [51].

Titans Memory-Augmented Attention. Titans introduces test-time neural memorization: instead of storing only activations, model updates internal memory using surprise-driven learning signals during inference. This addresses context beyond what fixed-state recurrent models typically support, enabling extremely long contexts and associative recall. The systems cost is higher implementation complexity because inference now includes memory adaptation logic [2].

2.1.3 Sparse Attention Mechanisms

Standard attention's quadratic complexity becomes prohibitive for sequences exceeding hundreds of thousands of tokens. Sparse attention mechanisms address this by restricting set of token interactions, exploiting observation that most learned attention weights are near zero.

Sparse Transformer. Introduces factorized sparse attention reducing $\mathcal{O}(n^2)$ complexity to $\mathcal{O}(n\sqrt{n})$. Uses two complementary sparse patterns: strided attention and fixed attention, each attending to $\mathcal{O}(\sqrt{n})$ positions [8].

Longformer. Introduces sliding window attention with optional global tokens, achieving linear $\mathcal{O}(n \cdot w)$ scaling where w is fixed window size. Combines sliding windows, dilated patterns, and global tokens to maintain long-range connectivity with sparse computation [3].

BigBird. Combines local windows, random links, and global tokens to preserve strong long-context connectivity with sparse computation. Randomized sparse mask plus local/global paths maintains theoretical approximation properties [49].

SparseK. Uses differentiable top-k style projection over importance scores before attention, so only selected KV pairs participate in expensive dot-product path [23].

NSA (Native Sparse Attention). Implements three-branch sparse design: compressed branch, selected branch, and local window branch, then combines them with learned gates [47].

SpargeAttn. Two-stage training-free block filtering: first predicts negligible block interactions, then applies softmax-aware pruning to remove low-contribution blocks [52].

FASA (Frequency-Aware Sparse Attention). Uses dominant RoPE frequency chunks for token importance prediction, then applies full attention only on selected tokens. Eval-only in this repository due to training-free nature [41].

2.1.4 Gated Attention Mechanisms

The unifying idea across gated architectures is that gating controls *what information survives*. Gates act on recurrent state updates (GLA, DeltaNet variants, HGRN2) or modify full attention path (FoX, Gated Softmax), making them especially useful when model must trade off recall, recency, and bounded memory.

Gated Linear Attention (GLA). Augments vanilla linear attention with data-dependent gating mechanism. Linear attention formulates recurrence with matrix-valued hidden state $S_t = S_{t-1} + v_t k_t^\top$, but this purely additive accumulation leads to "memory overload". GLA introduces diagonal gating matrix G_t:

$$S_t = G_t \odot S_{t-1} + v_t k_t^\top, \quad o_t = S_t q_t$$

with G_t parameterized via outer-product structure $G_t = \mathbf{1}\boldsymbol{\alpha}_t^\top$ [44].

DeltaNet. Applies classical delta learning rule to linear attention. State update performs difference between predicted value and target value:

$$S_t = S_{t-1}(I - \beta_t k_t k_t^\top) + \beta_t v_t k_t^\top$$

This is mathematically equivalent to single step of stochastic gradient descent on online MSE loss at each timestep, connecting DeltaNet to test-time training paradigm [45].

Gated DeltaNet. Synthesizes gating mechanism from Mamba2/GLA and delta rule from DeltaNet. Gated delta rule state update:

$$S_t = S_{t-1}\left(\alpha_t(I - \beta_t k_t k_t^\top)\right) + \beta_t v_t k_t^\top$$

where $\alpha_t \in (0, 1)$ is decay gate and $\beta_t \in (0, 1)$ is writing strength. These mechanisms are complementary: gating enables rapid memory erasure, while delta rule enables targeted updates [46].

HGRN2. Introduces outer-product-based state expansion mechanism with hierarchical forget gates. State update:

$$S_t = \mathrm{diag}(g_t) \cdot S_{t-1} + v_t k_t^\top, \quad o_t = S_t q_t$$

where g_t has hierarchically lower-bounded values increasing monotonically in upper layers, enforcing multi-scale temporal modeling [28].

FoX (Forgetting Transformer). Embeds forget gate directly into softmax attention by adding data-dependent logit bias to attention scores. Preserves full expressiveness and quadratic-time computation of softmax attention while gaining benefits of recency control [19].

Gated Softmax Attention. Applies post-SDPA sigmoid gate:

$$Y' = \text{SDPA}(Q, K, V) \odot \sigma(XW_g)$$

which adds multiplicative channel gating without replacing softmax attention [29].

2.2 Optimization Algorithms

Optimization of deep neural networks, particularly highly parameterized transformer architectures, represents one of most mathematically complex challenges in modern computational learning theory. Loss landscapes of large language models are characterized by severe non-convexity, saddle points, and block heterogeneity, where Hessian spectrum varies dramatically across different parameter blocks.

2.2.1 Classical Baselines

SGD with Momentum. The classical update accumulates momentum buffer and applies fixed learning rate. Strengths are low memory overhead and strong generalization when tuned carefully. Main weakness in transformer workloads is poor robustness to heterogeneous curvature and high dependence on learning-rate schedules [27].

Adam and AdamW. Adam tracks first and second moments of gradient; AdamW decouples weight decay from adaptive step. AdamW is treated as practical baseline for transformer fine-tuning because it converges quickly and is relatively forgiving. Tradeoff is state cost, since both moment tensors must be stored for every parameter [16, 22].

RAdam. Introduces variance rectification in early training, motivated by observation that Adam's adaptive denominator is unreliable during initial steps. This reduces warmup sensitivity without abandoning Adam family [21].

2.2.2 Advanced Momentum and Variance Reduction (2024–2025)

Adan (Adaptive Nesterov Momentum). Reformulates Nesterov-style momentum to avoid extra gradient computation at extrapolation point. Uses Nesterov Momentum Estimation (NME) for both first- and second-order moment estimates, achieving fast convergence across CNNs, Transformers, and GANs. Requires three momentum buffers [42].

ADOPT. Fixes Adam's theoretical non-convergence by: (1) removing current gradient from second-moment estimate and (2) reordering momentum update and normalization. Achieves optimal $\mathcal{O}(1/\sqrt{T})$ rate with any choice of β_2, without bounded noise assumptions [37].

AdEMAMix. Replaces Adam's single EMA of gradients with mixture of two EMAs—one fast-decaying and one slow-decaying. Demonstrates that gradients remain relevant for tens of thousands of steps, significantly slowing model forgetting during training [26].

MARS (Make Variance Reduction Shine). Unified framework reconciling preconditioned gradient methods with variance reduction via scaled stochastic recursive momentum. Offers instances based on AdamW, Lion, and Shampoo, consistently outperforming AdamW by large margin on GPT-2 [48].

Cautious Optimizers. One-line modification to any momentum-based optimizer: update is masked so that only components where momentum and gradient agree in sign are applied. Preserves convergence guarantees while significantly speeding up training [18].

2.2.3 Memory-Efficient and Schedule-Free Optimizers

Adafactor. Factorizes second-moment statistics for matrix-shaped parameters, dramatically reducing optimizer-state memory. Most attractive when memory is bottleneck and some loss in optimizer simplicity is acceptable [33].

GaLore. Projects gradients into low-rank subspace before optimization. Complementary memory-saving path especially relevant when model is too large for full-rank optimizer state [53].

Prodigy. Parameter-free or distance-adaptive method estimating effective step sizes from optimization geometry, reducing need for explicit learning-rate tuning [24].

Schedule-Free AdamW. Removes explicit scheduler design from optimization recipe. Emphasizes operational simplicity: instead of investing effort in warmup and decay design, one can use optimizer whose update dynamics absorb part of that responsibility [9].

Lion. Uses sign momentum updates carrying much smaller state than Adam-like methods. Positioned as low-memory, high-throughput alternative rather than universally superior optimizer [7].

2.2.4 Second-Order and Curvature-Aware Optimizers

Shampoo. Computes matrix preconditioners from Kronecker-structured statistics. One of most explicit second-order methods, motivated by conditioning improvements rather than minimal implementation complexity [13].

SOAP. Keeps Adam-style tracking in eigenbasis of Shampoo-like preconditioner. Hybrid between full matrix preconditioning and adaptive first-order behavior [39].

Sophia. Uses approximate second-order information through diagonal Hessian estimates and clipped updates. Belongs to family of curvature-aware methods seeking better conditioning without paying full cost of dense second-order optimization [20].

Muon and Turbo-Muon. Orthogonality-oriented optimizers explicitly reshaping update geometry. Turbo-Muon adds faster Newton–Schulz-style orthogonalization, making same basic idea more practical at scale [34, 4].

3 System Design and Architecture

3.1 Configuration-Centric Architecture

The core design choice in this repository is that experimentation is *schema first*. Instead of exposing a large number of loosely checked flags, the project forces model topology, optimizer family, training limits, and telemetry options through a single validated configuration document. This reduces ambiguity when reproducing results and makes it possible to compare many architectures under a consistent operational interface.

The authoritative contract is `src/training/configs/schema.yaml`. It enforces three top-level objects:

- `model_class`
- `model`
- `training`

The `model.layer_pattern` supports legacy, sparse, and gated blocks:

- **Retentive Network (RetNet)** — internal reference: `sun_retentive_2023` — code name: `retnet`, `retnet_attn`
- **Mamba (Selective State Space Model)** — internal reference: `gu_mamba_2023` — code name: `mamba`
- **ODE-style Continuous Depth Block** — internal reference: `zhang_continuous_2021` — code name: `ode`
- **Titans Memory-Augmented Attention** — internal reference: `behrouz_titans_2025` — code name: `titan_attn`
- **Standard Softmax Attention** — internal reference: `vaswani_attention_2017` — code name: `standard_attn`
- **Sigmoid Self-Attention** — internal reference: `ramapuram_theory_2024` — code name: `sigmoid_attn`
- **Sparse Transformer** — internal reference: `child_sparse_transformer_2019` — code name: `sparse_transformer_attn`
- **Longformer** — internal reference: `beltagy_longformer_2020` — code name: `longformer_attn`
- **BigBird** — internal reference: `zaheer_bigbird_2020` — code name: `bigbird_attn`
- **SparseK Attention** — internal reference: `lou_sparsek_2024` — code name: `sparsek_attn`
- **Native Sparse Attention (NSA)** — internal reference: `yuan_nsa_2025` — code name: `nsa_attn`
- **SpargeAttn** — internal reference: `zhang_spargeattn_2025` — code name: `sparge_attn`
- **FASA (Frequency-aware Sparse Attention)** — internal reference: `wang_fasa_2026` — code name: `fasa_attn`
- **Gated Linear Attention (GLA)** — internal reference: `yang_gla_2023` — code name: `gla_attn`
- **DeltaNet** — internal reference: `yang_deltanet_2024` — code name: `deltanet_attn`
- **Gated DeltaNet** — internal reference: `yang_gated_deltanet_2024` — code name: `gated_deltanet_attn`
- **HGRN2** — internal reference: `qin_hgrn2_2024` — code name: `hgrn2_attn`
- **Forgetting Transformer (FoX)** — internal reference: `lin_forgetting_transformer_2025` — code name: `fox_attn`
- **Gated Softmax Attention** — internal reference: `qiu_gated_attention_2025` — code name: `gated_softmax_attn`

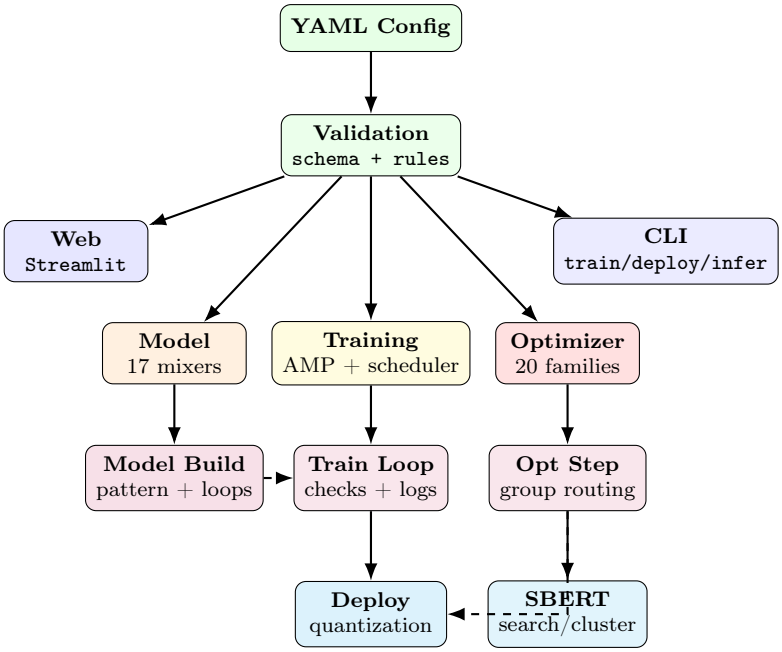

Figure 1: System architecture: configuration flows through validation, partitions into model/training/optimizer, executes runtime, produces deployment/SBERT artifacts.

which corresponds to current attention and sequence-mixer literature [36, 11, 51, 2, 30, 38, 8, 3, 49, 23, 47, 52, 41, 44, 45, 46, 28, 19, 29]

The `training.optimizer.optimizer_class` supports a broad optimizer family: `adamw`, `adafactor`, `radam`, `adan`, `adopt`, `ademamix`, `mars_adamw`, `cautious_adamw`, `schedulefree_adamw`, `lion`, `sophia`, `prodigy`, `muon`, `turbo_muon`, `shampoo`, `soap`, and others.

3.2 Schema Scope and Validation Rules

The schema is strict: top-level and nested objects set `additionalProperties: false`. This guarantees that unknown keys fail fast instead of being silently ignored. The `training.optimizer.parameters` object is additionally constrained by optimizer-specific prefix rules through `allOf`+`if/then` pattern checks.

Normalization values currently accepted by schema are:

$$\texttt{norm\_type} \in \{\texttt{layer\_norm}, \texttt{dynamic\_tanh}, \texttt{derf}\}$$

Thus, `rms_norm` is **not** a valid schema value in the current contract.

3.3 Complete Model Feature Inventory

Field	Type/Range	Meaning
`vocab_size`	int ≥ 1	Vocabulary size.
`hidden_size`	int ≥ 1	Hidden dimension.
`num_layers`	int ≥ 1	Physical layer count.
`num_loops`	int ≥ 1	Logical loop count (looped blocks).
`num_heads`	int ≥ 1	Attention heads.
`retention_heads`	int ≥ 1	Retention heads for RetNet-style mixers.
`num_experts`	int ≥ 1	MoE expert count.
`top_k_experts`	int ≥ 1	Top-k expert routing in MoE.
`dropout`	float $[0, 1]$	Global dropout.

Field	Type/Range	Meaning
layer_pattern	array enum	Ordered block list: legacy (retnet, retnet_attn, mamba, ode, titan_attn, standard_attn, sigmoid_attn), sparse (sparse_transformer_attn, longformer_attn, bigbird_attn, sparsek_attn, nsa_attn, sparge_attn, fasa_attn), and gated (gla_attn, deltanet_attn, gated_deltanet_attn, hgrn2_attn, fox_attn, gated_softmax_attn).
ode_solver	enum	rk4 or euler.
ode_steps	int ≥ 1	ODE integration steps.
use_bitnet	bool	Enable low-bit BitLinear path.
norm_type	enum	layer_norm, dynamic_tanh, derf.
use_factorized_embedding	bool	Enable factorized embeddings.
factorized_embedding_dim	int ≥ 1	Reduced embedding dimension for factorization.
use_embedding_conv	bool	Enable Conv1d over embedding stream.
embedding_conv_kernel	int ≥ 1	Conv1d kernel size.
hope_base	float ≥ 0	HoPE base value (optional in schema).
hope_damping	float ≥ 0	HoPE damping (optional in schema).
use_hope	bool	Apply HoPE in titan_attn.
use_moe	bool	Enable MoE FFN routing path.
ffn_hidden_size	int ≥ 1	FFN intermediate width.
ffn_activation	enum	silu or gelu.

Looped depth induced by schema is:

$$L_{\text{logical}} = \texttt{num\_layers} \times \texttt{num\_loops}$$

which is the configuration-level definition of looped blocks.

3.4 Complete Training Feature Inventory

Field	Type/Range	Meaning
batch_size	int ≥ 1	Loader batch size.
dataloader_workers	int ≥ 0	PyTorch dataloader workers.
max_length	int ≥ 1	Sequence length cap.
mlm_probability	float $[0, 1]$	MLM masking probability.
max_samples	int ≥ 1	Maximum streamed samples.
dataset_batch_size	int ≥ 1	Internal streaming dataset chunk size.
num_workers	int ≥ 0	Streaming dataset workers.
cache_dir	string	Dataset cache directory.
local_parquet_dir	string	Optional local parquet path.
prefer_local_cache	bool	Prefer local cache when available.
stream_local_parquet	bool	Stream from local parquet mode.
use_amp	bool	Mixed precision toggle.
gradient_accumulation_steps	int ≥ 1	Effective batch through accumulation.
optimizer	object	Contains optimizer_class and prefixed parameters.
scheduler_total_steps	int ≥ 1	Scheduler horizon.
scheduler_warmup_ratio	float $[0, 1]$	Warmup ratio.
scheduler_type	enum	cosine, constant, linear_warmup_then_constant.

Field	Type/Range	Meaning
grad_clip_max_norm	float ≥ 0	Global norm clipping threshold.
inf_post_clip_threshold	float ≥ 0	Exploding-gradient guard threshold after clipping.
max_nan_retries	int ≥ 0	Retry budget for NaN/Inf instability.
checkpoint_every_n_steps	int ≥ 1	Rolling checkpoint frequency.
max_rolling_checkpoints	int ≥ 1	Number of rolling checkpoints to keep.
num_best_checkpoints	int ≥ 1	Number of best checkpoints tracked.
nan_check_interval	int ≥ 1	NaN/Inf check cadence.
log_gradient_stats	bool	Enable gradient statistics logging.
gradient_log_interval	int ≥ 1	Gradient logging cadence.
csv_log_path	string	Step-level CSV output path.
csv_rotate_on_schema_change	bool	Rotate CSV if logging schema changes.
gpu_metrics_backend	enum	nvml or none.
nvml_device_index	int ≥ 0	Device index for NVML telemetry.
enable_block_grad_norms	bool	Include per-block gradient norm telemetry.
telemetry_log_interval	int ≥ 1	Heavy telemetry interval (optimizer steps).
use_galore	bool	Enable GaLore strategy.
galore_rank	int ≥ 1	GaLore low-rank projection dimension.
galore_update_interval	int ≥ 1	Projection refresh interval.
galore_scale	float ≥ 0	Gradient scaling in projected space.
galore_max_dim	int ≥ 1	Maximum tensor dimension for GaLore projection.

3.5 Optimizer Prefix Contract (Full)

Supported optimizer_class values are: sgd_momentum, adamw, adafactor, galore_adamw, prodigy, lion, sophia, muon, turbo_muon, radam, adan, adopt, ademamix, mars_adamw, cautious_adamw, lamb, schedulefree_adamw, shampoo, soap.

Shared per-group suffix families (all prefixed by optimizer name) are:

- LR groups: lr_embeddings, lr_norms, lr_ode, lr_retnet, lr_mamba, lr_attention, lr_other
- Weight decay groups: wd_embeddings, wd_norms, wd_ode, wd_retnet, wd_mamba, wd_attention, wd_other
- Beta groups: betas_embeddings, betas_norms, betas_ode, betas_retnet, betas_mamba, betas_attention, betas_other
- Epsilon groups: eps_embeddings, eps_norms, eps_ode, eps_retnet, eps_mamba, eps_attention, eps_other

Optimizer-specific global suffixes:

- sgd_momentum: momentum, nesterov
- adafactor: beta2_decay, clip_threshold, eps1, eps2
- galore_adamw: rank, update_proj_gap
- prodigy: d_coef
- sophia: rho, update_k
- muon / turbo_muon: momentum, nesterov, ns_steps, ns_eps
- cautious_adamw: cautious_clip

All other classes in the list above accept only prefixed shared groups.

Algorithm 1 Schema-Driven Training Step with Stability Controls

Require: Batch stream, config C

Initialize retry counter $r \leftarrow 0$
for each optimizer step **do**
    Accumulate gradients for $K = C.$`gradient_accumulation_steps` micro-batches
    Apply global norm clipping with $\tau = C.$`grad_clip_max_norm`
    if post-clip gradient exceeds $C.$`inf_post_clip_threshold` or NaN/Inf detected **then**
        if $r < C.$`max_nan_retries` **then**
            restore safe state / skip step; $r \leftarrow r + 1$
            continue
        else
            stop training with failure state
        end if
    end if
    run optimizer step selected by `optimizer_class`
    update scheduler (`cosine`, `constant`, or `linear_warmup_then_constant`)
    if step mod `checkpoint_every_n_steps`$= 0$ **then**
        save rolling checkpoint and prune to `max_rolling_checkpoints`
    end if
    update best checkpoints up to `num_best_checkpoints`
    emit CSV + telemetry following `gradient_log_interval` and `telemetry_log_interval`
end for

3.6 Training Safety and Runtime Semantics

Schema-level safety features include accumulation, clipping, post-clip explosion checks, and NaN retries:

$$g_{\text{acc}} = \frac{1}{K}\sum_{i=1}^{K} g_i, \quad K = \texttt{gradient\_accumulation\_steps}$$

$$g_{\text{clip}} = g_{\text{acc}} \cdot \min\left(1, \frac{\tau}{\|g_{\text{acc}}\|_2 + \epsilon}\right), \quad \tau = \texttt{grad\_clip\_max\_norm}$$

then overflow guards use `inf_post_clip_threshold` and retry logic bounded by `max_nan_retries`.

3.7 Normalization Variants: RMSNorm, Dynamic Tanh, and Dynamic Erf

Normalization determines how activation scale is controlled across depth. In this repository, normalization is not only a modeling choice but also a schema compatibility question, because only certain values are currently accepted by `norm_type`. The three formulations most relevant to this codebase are:

3.8 RMSNorm

RMSNorm removes mean-centering and only rescales by root mean square magnitude [50]:

$$\text{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon}, \qquad y_i = \gamma_i \frac{x_i}{\text{RMS}(x)}$$

Compared with LayerNorm, RMSNorm is computationally simpler (no subtraction of feature mean) and is often used when reducing normalization overhead is important.

3.9 Dynamic Tanh (DyT)

Dynamic Tanh proposes replacing explicit normalization with a bounded elementwise map [54]:

$$\mathrm{DyT}(x) = \tanh(\alpha x)$$

where α is learned. The core idea is that bounded nonlinear contraction can provide stable signal scaling without explicitly computing per-token normalization statistics.

3.10 Dynamic Erf (Derf)

Derf extends the same normalization-free direction by using an error-function based map [6]:

$$\mathrm{Derf}(x) = \mathrm{erf}(\alpha x + s)$$

with learnable scale/shift. Reported results in the cited work indicate stronger performance than DyT and common normalization baselines across multiple domains.

3.11 Schema Implications

Current configuration contract in this repository allows:

$$\texttt{norm\_type} \in \{\texttt{layer\_norm}, \texttt{dynamic\_tanh}, \texttt{derf}\}$$

so DyT and Derf are directly available in schema-driven runs, while RMSNorm is not currently an accepted enum value and would require code/schema extension.

Method	Formula	Stats Needed	Notes
RMSNorm	$\gamma_i x_i / \sqrt{\frac{1}{d}\sum_j x_j^2 + \epsilon}$	RMS only	Lower overhead; widely used baseline [50].
Dynamic Tanh	$\tanh(\alpha x)$	none	Normalization-free bounded transform; drop-in replacement [54].
Dynamic Erf	$\mathrm{erf}(\alpha x + s)$	none	Normalization-free alternative; improves over DyT [6].

4 Architecture Taxonomy and Implementation

4.1 Attention and Sequence-Mixer Families

This system implements seventeen distinct sequence mixer architectures organized into five functional categories reflecting research trends in sequence modeling design. The taxonomical organization reflects evolving understanding of how to balance expressivity, computational efficiency, and memory constraints.

1. **Dense Attention Baselines**: Standard softmax attention and sigmoid attention provide full global contextualization at quadratic computational cost, serving as reference baselines for comparison with more efficient alternatives.

2. **Recurrent and Retentive Architectures**: RetNet, Mamba, ODE-style blocks, and Titans maintain state representations enabling $\mathcal{O}(1)$ inference cost while preserving expressivity through recurrent dynamics, selective parameters, or test-time memory adaptation.

3. **Sparse Attention Patterns**: Seven sparse variants (Sparse Transformer, Longformer, BigBird, SparseK, NSA, SpargeAttn, FASA) reduce quadratic complexity through structured sparsity, token selection, or training-free pruning strategies.

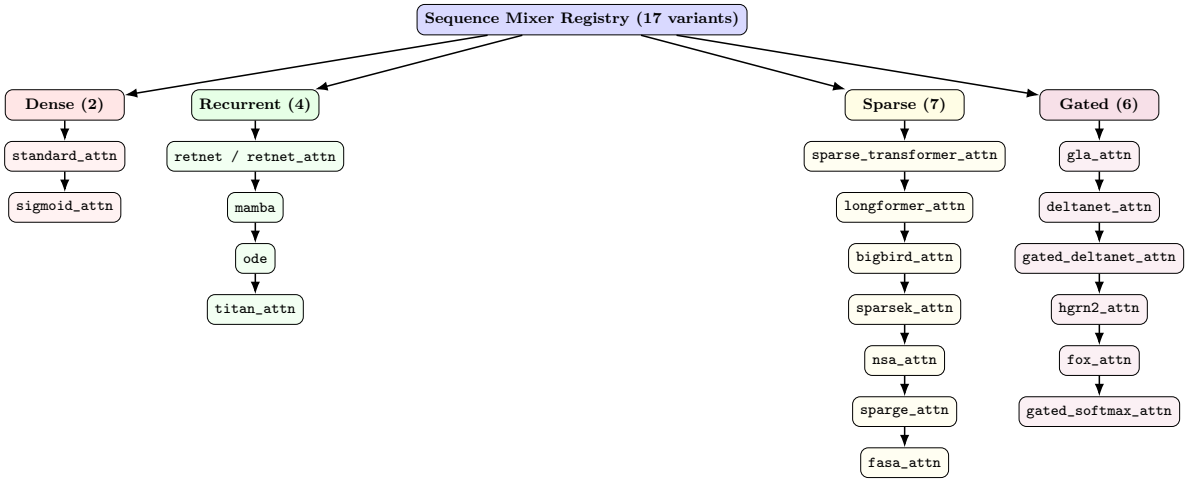

Figure 2: Comprehensive taxonomy of seventeen supported sequence mixer architectures. Dense baselines provide full global routing at quadratic cost. Recurrent architectures enable constant-time inference through state compression. Sparse variants reduce complexity through structured patterns or token selection. Gated mechanisms introduce data-dependent control over memory retention and forgetting.

4. **Gated Memory Mechanisms**: Six gated architectures (GLA, DeltaNet, Gated DeltaNet, HGRN2, FoX, Gated Softmax) introduce data-dependent control over memory retention, forgetting, and update strength.

4.2 Standard Attention

Given projected matrices (Q, K, V):

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

This is the baseline mechanism for content routing [38].

4.3 Sigmoid Attention

Sigmoid attention removes row-wise probability normalization:

$$\text{SigmoidAttn}(Q, K, V) = \sigma\left(\frac{QK^\top}{\sqrt{d_k}} + b\right)V$$

and has different training stability requirements, often with additional normalization [30].

4.4 Retentive Formulation

RetNet uses retention with decay matrix D:

$$\text{Retention}(Q, K, V) = \left(QK^\top \odot D\right)V$$

with recurrent form:

$$S_n = \gamma S_{n-1} + k_n^\top v_n, \qquad o_n = q_n S_n$$

enabling low-cost recurrent inference [36, 43].

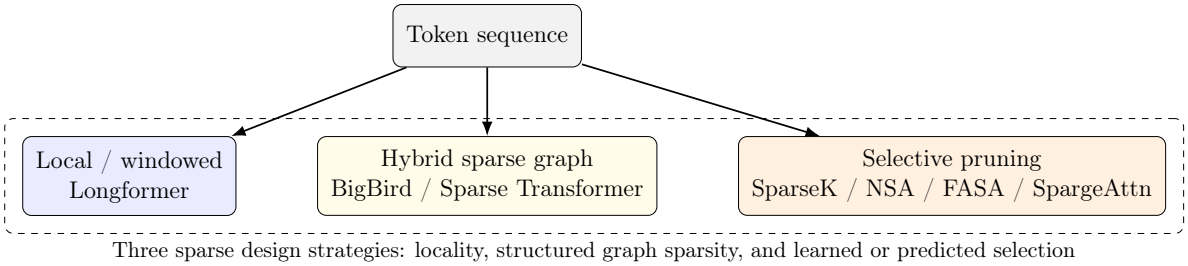

Figure 3: Conceptual map of sparse attention design choices used in the codebase. Different methods reduce cost by restricting neighborhoods, constructing sparse graphs, or selecting only high-value tokens/blocks.

4.5 Selective SSM (Mamba)

Discrete selective state-space recurrence is:

$$h_t = \bar{A}_t h_{t-1} + \bar{B}_t x_t, \qquad y_t = C_t h_t$$

where $(\bar{A}_t, \bar{B}_t, C_t)$ depend on input, preserving linear-time scaling with hardware-aware scan [11, 14].

4.6 ODE-style Continuous Updates

Continuous-depth framing:

$$\frac{dh(t)}{dt} = f_\theta(h(t), t)$$

with practical RK integrators for discrete execution [51].

4.7 Test-time Memory (Titans)

A memory-augmented update can be written:

$$M_t = (1 - \alpha_t) M_{t-1} + S_t, \qquad S_t = \eta_t S_{t-1} - \theta_t \nabla \ell(M_{t-1}; x_t)$$

to adapt memory at inference time [2, 1].

4.8 Sparse and Gated Extensions in the Current Codebase

The mixer registry now includes sparse blocks: `sparse_transformer_attn`, `longformer_attn`, `bigbird_attn`, `sparsek_attn`, `nsa_attn`, `sparge_attn`, and `fasa_attn`; and gated blocks: `gla_attn`, `deltanet_attn`, `gated_deltanet_attn`, `hgrn2_attn`, `fox_attn`, and `gated_softmax_attn`.

The implementation enforces an explicit execution policy for training-free sparse methods: `fasa_attn` and `sparge_attn` are eval/inference-only and raise runtime errors if used while the model is in training mode.

4.9 Implemented Sparse Attention Blocks (Detailed)

This codebase includes seven sparse attention families aligned with the sparse attention survey notes in `docs/bibliography/SPARSE_TRANSFORMER_TYPES.md` [8, 3, 49, 23, 47, 52, 41].

Sparse Transformer (`sparse_transformer_attn`). Uses factorized sparse masks (strided + fixed) to approximate dense connectivity at lower cost than full $\mathcal{O}(n^2)$ attention:

$$\text{Attn}_i = \text{softmax}\left(\frac{q_i K_{A_i}^\top}{\sqrt{d_k}}\right) V_{A_i}$$

where A_i is the sparse neighborhood induced by stride/fixed rules. [8]

Longformer (`longformer_attn`). Uses sliding-window locality with optional global tokens:

$$A_i = \{j : |i - j| \le w/2\} \cup \mathcal{G}$$

yielding linear scaling in sequence length for fixed window w. [3]

BigBird (`bigbird_attn`). Combines local windows, random links, and global tokens:

$$A_i = A_i^{\text{window}} \cup A_i^{\text{random}} \cup A_i^{\text{global}}$$

to preserve strong long-context connectivity with sparse computation. [49]

SparseK (`sparsek_attn`). Uses a differentiable top-k style projection over importance scores before attention, so only selected KV pairs participate in the expensive dot-product path. [23]

NSA (`nsa_attn`). Implements a three-branch sparse design: compressed branch, selected branch, and local window branch, then combines them with learned gates:

$$o_t = \sum_{c \in \{\text{cmp,sel,win}\}} g_t^c \, \text{Attn}(q_t, \tilde{K}_t^c, \tilde{V}_t^c)$$

[47]

SpargeAttn (`sparge_attn`). Two-stage training-free block filtering: first predicts negligible block interactions, then applies softmax-aware pruning to remove low-contribution blocks. [52]

FASA (`fasa_attn`). Frequency-aware training-free attention: uses dominant RoPE frequency chunks for token importance prediction, then applies full attention only on selected tokens. [41]

Block	Trainable	Asymptotic Trend	Primary Sparsity Unit	Current Integration Notes	Ref
Sparse Transformer	Yes	sub-quadratic	mask pattern (token-level)	Factorized strided/fixed masks inside SDPA pipeline.	[8]
Longformer	Yes	linear in n (fixed w)	sliding window + global tokens	Window mask with optional global indices.	[3]
BigBird	Yes	near-linear	window + random + global edges	Randomized sparse mask plus local/global paths.	[49]
SparseK	Yes	linear-like (selected KV)	differentiable top-k KV selection	Learned score net + SparseK projection + gathered KV attention.	[23]

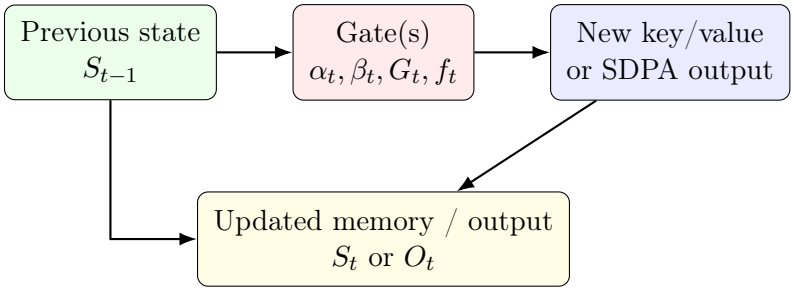

Figure 4: Generic gating template. A gate can decay existing memory, regulate write strength, or modulate dense attention outputs, depending on the block family.

Block	Trainable	Asymptotic Trend	Primary Sparsity Unit	Current Integration Notes	Ref
NSA	Yes	reduced-token multi-branch	compressed blocks + selected blocks + local window	Three sparse branches gated into one output tensor.	[47]
SpargeAttn	No (training-free)	sparse block dependent	block-level predicted sparsity	Eval-only in this repo; raises in training mode.	[52]
FASA	No (training-free)	selected-token dependent	dominant frequency chunks + selected tokens	Eval-only in this repo; raises in training mode.	[41]

4.10 Implemented Gated Attention Blocks (Detailed)

This codebase includes seven gated blocks aligned with `docs/bibliography/GATED_TRANSFORMER_TYPES.md` [44, 45, 46, 36, 28, 19, 29].

The unifying idea is that gating controls *what information survives*. Some gates act on recurrent state updates (GLA, DeltaNet variants, HGRN2), while others modify the full attention path itself (FoX and Gated Softmax). This makes gating especially useful when the model must trade off recall, recency, and bounded memory.

GLA (`gla_attn`). Gated Linear Attention applies data-dependent multiplicative decay in recurrent state updates:

$$S_t = G_t \odot S_{t-1} + v_t k_t^\top, \qquad o_t = S_t q_t$$

to control memory accumulation. [44]

DeltaNet (`deltanet_attn`). Uses a delta-rule error-correcting write with learned write strength β_t:

$$S_t = S_{t-1}(I - \beta_t k_t k_t^\top) + \beta_t v_t k_t^\top$$

which improves targeted memory replacement. [45]

Gated DeltaNet (`gated_deltanet_attn`). Adds decay gate α_t on top of delta-rule writes:

$$S_t = \alpha_t S_{t-1}(I - \beta_t k_t k_t^\top) + \beta_t v_t k_t^\top$$

for both global forgetting and local corrective updates. [46]

RetNet Attn Alias (`retnet_attn`). Provides an explicit gated-package alias wrapping multi-scale retention behavior for naming consistency in layer registries. [36]

HGRN2 (`hgrn2_attn`). Uses lower-bounded forget gates with outer-product state expansion:

$$S_t = \mathrm{diag}(g_t) S_{t-1} + v_t k_t^\top$$

to increase recurrent state expressiveness while remaining efficient. [28]

FoX (`fox_attn`). Injects token-wise forget bias directly into softmax logits:

$$O = \mathrm{softmax}(QK^\top + D)V$$

where D is derived from cumulative log-forget gates. [19]

Gated Softmax (`gated_softmax_attn`). Applies a post-SDPA sigmoid gate:

$$Y' = \mathrm{SDPA}(Q, K, V) \odot \sigma(XW_g)$$

which adds multiplicative channel gating without replacing softmax attention. [29]

Block	State Type	Gate Mechanism	Softmax Path	Current Integration Notes	Ref
GLA	matrix recurrent state	data-dependent multiplicative decay	No (linear recurrent)	Recurrent update with low-rank gate projection.	[44]
DeltaNet	matrix recurrent state	write gate (β)	No (linear recurrent)	Delta-rule correction with normalized Q/K.	[45]
Gated DeltaNet	matrix recurrent state	decay + write gates (α, β)	No (linear recurrent)	Combined forgetting and targeted writing.	[46]
RetNet Attn	matrix recurrent state	fixed multi-scale decay	No (retention)	Alias wrapper over existing RetNet mixer.	[36]
HGRN2	matrix recurrent state	lower-bounded forget gate	No (linear recurrent)	Hierarchical recurrent gating with outer products.	[28]
FoX	full attention matrix	logit-space forget gate	Yes	Forget bias added before softmax.	[19]
Gated Softmax	full attention matrix	post-attention sigmoid gate	Yes	Sigmoid gating applied after SDPA output.	[29]

5 Optimizer Families and Training Dynamics

Optimization of highly parameterized transformer architectures presents significant challenges due to non-convex loss landscapes, saddle points, and block heterogeneity across parameter groups. The Hessian spectrum varies dramatically between embeddings, attention weights, normalization layers, and feed-forward networks, creating optimization pathways where certain dimensions are exponentially sharper or flatter than others. This system addresses these challenges through a unified framework supporting twenty optimizer families spanning multiple algorithmic trajectories.

Algorithm 2 Pattern-Driven Mixer Forward (Conceptual)

Require: Hidden states H, pattern P, layer index ℓ

$m \leftarrow P[\ell \bmod |P|]$
if $m \in \{\texttt{fasa\_attn}, \texttt{sparge\_attn}\}$ and model is in training mode **then**
    raise configuration/runtime error (training-free block in train mode)
else if $m = \texttt{standard\_attn}$ **then**
    $H \leftarrow$ softmax-attention(H)
else if $m = \texttt{sigmoid\_attn}$ **then**
    $H \leftarrow$ sigmoid-attention(H)
else if $m \in \{\texttt{retnet}, \texttt{retnet\_attn}\}$ **then**
    $H \leftarrow$ retention(H)
else if $m = \texttt{mamba}$ **then**
    $H \leftarrow$ selective-ssm(H)
else if $m = \texttt{ode}$ **then**
    $H \leftarrow$ rk-step(H)
else if m is a sparse attention key **then**
    $H \leftarrow$ sparse-attention-family(H)
else if m is a gated attention key **then**
    $H \leftarrow$ gated-attention-family(H)
else
    $H \leftarrow$ memory-augmented-attn(H)
end if
return H

5.1 Taxonomic Organization of Supported Optimizers

The optimizer support is organized into six algorithmic categories reflecting modern research directions in deep learning optimization:

1. **Classical Baselines**: SGD with momentum, Adam, and AdamW define the comparison floor against which newer methods are evaluated. These methods are well-understood but exhibit limitations including sensitivity to learning-rate schedules, poor robustness to heterogeneous curvature (SGD), and high memory overhead for large models (Adam-family).

2. **Advanced Momentum and Variance Reduction**: Adan, ADOPT, AdEMAMix, MARS, and Cautious AdamW (2024–2025) address specific limitations of classical methods. Adan reformulates Nesterov acceleration, ADOPT fixes Adam's theoretical non-convergence, AdEMAMix introduces dual-EMA history mixing, MARS brings variance reduction to large-scale training, and Cautious variants mask updates to sign-consistent directions.

3. **Memory-Efficient Optimizers**: Adafactor, GaLore, and Lion reduce optimizer-state memory through factorization, low-rank projection, or sign-based updates. These methods are critical when VRAM is dominated by optimizer state rather than activations.

4. **Schedule-Free and Parameter-Free**: Schedule-Free AdamW and Prodigy remove explicit scheduler design or learning-rate tuning by making update dynamics absorb these responsibilities, reducing hyperparameter search complexity.

5. **Curvature-Aware and Second-Order**: Sophia, Shampoo, and SOAP incorporate approximate second-order information through diagonal Hessian estimates, Kronecker-structured statistics, or eigenbasis tracking, improving conditioning at cost of implementation complexity.

6. **Geometry-Oriented**: Muon and Turbo-Muon explicitly reshape update geometry through orthogonalization, useful when matrix structure matters for representation shaping.

5.2 Core Adaptive Formulation

Many supported optimizers share fundamental moment-tracking formulation. Adam-family optimizers track first and second moments of gradients:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

followed by preconditioned updates with bias correction:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad \theta_{t+1} = \theta_t - \eta_t \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)$$

where η_t is the learning rate schedule, ϵ is numerical stability constant, and λ is weight decay coefficient [22].

5.3 Detailed Optimizer Descriptions

- **RAdam**: variance rectification for early-step instability [21].
- **Adan**: adaptive Nesterov momentum for faster convergence [42].
- **ADOPT**: modified Adam order yielding stronger convergence guarantees [37].
- **AdEMAMix**: dual-EMA history mixing [26].
- **MARS**: variance reduction in preconditioned optimization [48].
- **Cautious optimizers**: sign-consistent masking of momentum updates [18].
- **Schedule-free**: remove explicit scheduler dependence [9].
- **Shampoo/SOAP**: matrix preconditioning families [13, 39].
- **Adafactor/GaLore**: memory reduction via factorization or low-rank projection [33, 53].
- **Prodigy/Lion/Sophia**: parameter-free adaptation, sign momentum, and clipped second-order scaling [24, 7, 20].
- **Muon/Turbo-Muon**: orthogonality-oriented updates with acceleration [34, 4].

6 Quantization and Deployment

The deploy stack uses ternary weight packing plus INT8 activation quantization for efficient artifacts.

6.1 Ternary Quantization

Given weight tensor W, a practical scaling is:

$$s = \text{mean}(|W|), \qquad \tilde{W} = \text{clip}\left(\text{round}\left(\frac{W}{s} \right), -1, 1 \right)$$

which approximates BitNet-style low-bit updates [40]. The packed mapping uses two bits per weight symbol for storage efficiency.

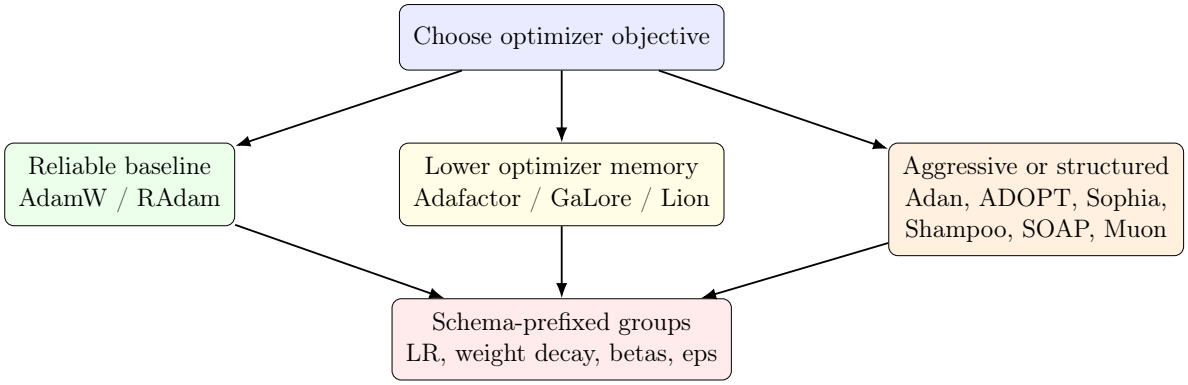

Figure 5: Optimizer selection in practice: the class determines the update rule, while the schema controls how hyperparameters are applied across embeddings, norms, recurrent blocks, attention blocks, and other parameters.

Algorithm 3 Schema-Routed Optimizer Step (Conceptual)

Require: Parameters θ, gradients g, optimizer class c, parameter map Π

Read optimizer-specific hyperparameters from prefixed keys in Π
if $c =$ `adamw` **then**
    apply AdamW step [22]
else if $c =$ `radam` **then**
    apply rectified adaptive step [21]
else if $c =$ `adan` **then**
    apply Adan three-moment step [42]
else if $c =$ `adopt` **then**
    apply ADOPT update ordering [37]
else if $c =$ `galore_adamw` **then**
    project gradients to low-rank subspace then step [53]
else
    dispatch to selected optimizer implementation
end if
return updated θ

6.2 Activation Quantization

For activations x:

$$q = \text{round}\left(x \cdot \frac{127}{\max(|x|) + \epsilon}\right), \qquad q \in [-128, 127]$$

with dequantization $x \approx q/\alpha$.

6.3 Size Estimates

For N parameters:

$$\text{FP32 size} \approx 4N, \quad \text{FP16 size} \approx 2N, \quad \text{1.58-bit size} \approx \frac{1.58}{8}N$$

before metadata and packing overhead. This aligns with lightweight deployment goals [32, 5].

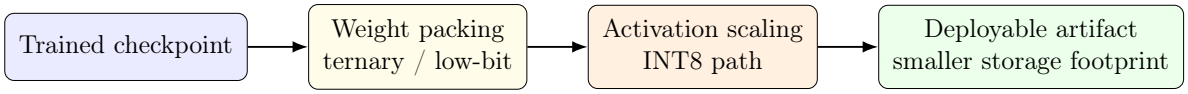

Figure 6: Deployment path from a trained checkpoint to a compact artifact. The codebase treats quantization as a deploy-stage transformation rather than a separate model family.

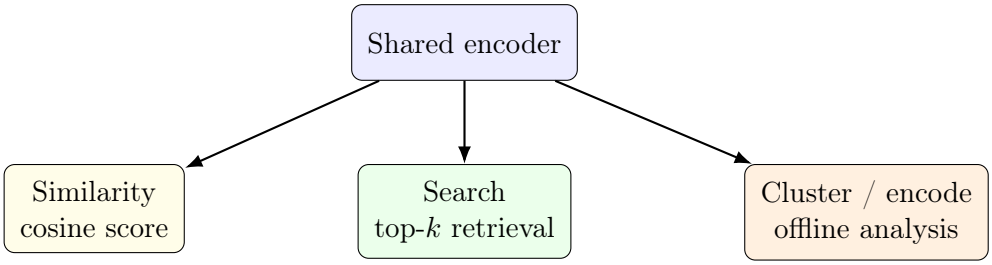

Figure 7: SBERT workflow reuse. A single encoder supports online pair scoring, corpus retrieval, clustering, and persistent embedding export.

7 SBERT Downstream Tasks

Sentence embedding is built on Siamese-style training [31]. For sentence pair (s_1, s_2) with embeddings (e_1, e_2):

$$\cos(e_1, e_2) = \frac{e_1^\top e_2}{\|e_1\|\|e_2\|}$$

and regression-style cosine loss:

$$\mathcal{L}_{\cos} = (\cos(e_1, e_2) - y)^2$$

with $y \in [-1, 1]$ in this pipeline.

Supported downstream modes:

- **Similarity**: pairwise score between two sentences.
- **Search**: top-k nearest neighbors over a corpus.
- **Cluster**: grouping embeddings (e.g., k-means).
- **Encode**: persistent embedding export for later retrieval.

8 Summary Tables

8.1 Attention and Sequence-Mixer Summary

Type	Core Equation	Train	Infer	Notes
Standard Attention	$\text{softmax}(QK^\top/\sqrt{d_k})V$	$\mathcal{O}(n^2 d)$	$\mathcal{O}(n)$/step	Baseline expressive global routing [38].
Sigmoid Attention	$\sigma(QK^\top/\sqrt{d_k} + b)V$	$\mathcal{O}(n^2 d)$	$\mathcal{O}(n)$/step	Element-wise gating; often needs stabilization norm [30].
RetNet	$(QK^\top \odot D)V$	$\mathcal{O}(n^2 d)$ or chunkwise	$\mathcal{O}(1)$/step	Parallel/recurrent dual form with decay retention [36].
Mamba	$h_t = \bar{A}_t h_{t-1} + \bar{B}_t x_t$	$\mathcal{O}(nd)$	$\mathcal{O}(1)$/step	Selective state-space with hardware-aware scan [11].

Type	Core Equation	Train	Infer	Notes
ODE-style block	$\frac{dh}{dt} = f_\theta(h, t)$	solver-dependent	solver-dependent	Continuous-depth interpretation; RK integration [51].
Titans memory	$M_t = (1-\alpha_t)M_{t-1} + S_t$	approx. $\mathcal{O}(nd)$	retrieval-centric	Test-time memory updates with surprise-driven dynamics [2].

8.2 Optimizer Summary

Optimizer	Family	State Cost	Key Idea	Ref
AdamW	Adaptive first/second moment	High	Decoupled weight decay baseline	[22]
RAdam	Adaptive variance-corrected	High	Rectifies early adaptive variance	[21]
Adan	Momentum + variance reduction	High	Nesterov-style adaptive update	[42]
ADOPT	Adam variant	High	Reordered updates with improved convergence guarantees	[37]
AdEMAMix	Multi-EMA adaptive	High	Mixes short and long horizon EMAs	[26]
MARS	Variance-reduced preconditioned	High	Recursive momentum correction	[48]
Cautious AdamW	Masked momentum	High	Apply updates only on sign-consistent directions	[18]
Schedule-free AdamW	Scheduler-free adaptive	High	Remove explicit LR schedule dependence	[9]
Adafactor	Memory-efficient adaptive	Medium	Factorized second moments for matrix tensors	[33]
GaLore AdamW	Low-rank gradient projection	Medium	Optimize in projected low-rank gradient space	[53]
Prodigy	Parameter-free adaptation	Medium	Distance-adaptive step calibration	[24]
Lion	Sign momentum	Low	Momentum sign update, reduced state	[7]
Sophia	Approx. second-order	Medium	Diagonal Hessian preconditioning with clipping	[20]
Shampoo	Matrix preconditioner	High	Kronecker-structured second-order statistics	[13]
SOAP	Shampoo + Adam basis	High	Adam-like tracking in preconditioner eigenbasis	[39]
Muon	Orthogonality-based	Medium	Orthogonalized matrix updates	[34]
Turbo-Muon	Accelerated orthogonalization	Medium	Preconditioned Newton-Schulz speedup	[4]

9 Discussion

The design choices in Transformer Encoder Frankenstein reflect several engineering and research tensions in modern deep learning tooling.

Algorithm 4 SBERT Inference Mode Router

Require: mode m, model E, inputs X

if $m =$ `similarity` **then**
    return $\cos(E(x_1), E(x_2))$
else if $m =$ `search` **then**
    return top-k by dot-product/cosine against corpus embeddings
else if $m =$ `cluster` **then**
    return clustering labels over $E(X)$
else
    return serialized embeddings $E(X)$
end if

9.1 Schema-Driven Design Trade-offs

The schema-first approach provides significant reproducibility benefits by enforcing explicit contracts and failing fast on invalid configurations. However, this approach also introduces rigidity: adding new architectures or optimizers requires schema extensions rather than loose command-line arguments. The prefixed hyperparameter system enables fine-grained control but increases configuration complexity for users accustomed to simpler interfaces.

The decision to enforce `additionalProperties: false` at all schema levels eliminates silent parameter swallowing that has plagued earlier configuration systems, but this strictness requires careful schema maintenance when extending system capabilities. Each new attention mechanism or optimizer variant must be properly integrated into the validation framework, including schema field definitions with appropriate types and constraints, prefixed hyperparameter mapping for optimizer-specific groups, default values aligned with research best practices, and documentation strings for web interface rendering.

9.2 Architectural Coverage and Gaps

The seventeen implemented mixer architectures span major research directions in sequence modeling, but certain gaps remain. The system lacks recent hybrid architectures such as Griffin [35] and Jamba [12], which combine gating with state-space models. MoE (Mixture of Experts) routing is implemented for FFN layers but not for attention computation, where recent work has shown benefits [17].

The sparse attention coverage is comprehensive, but implementation of training-free methods (FASA, SpargeAttn) raises runtime errors during training, reflecting architectural constraints: these methods require pretrained checkpoints from full-attention models or specific fine-tuning procedures that are not currently automated.

The gated mechanism coverage is strong across major categories (GLA, DeltaNet, Gated DeltaNet, HGRN2, FoX, Gated Softmax).

9.3 Optimizer Landscape Fragmentation

The support for twenty optimizer families across six algorithmic categories demonstrates comprehensiveness but also highlights the fragmented state of optimization research. Users face significant decision complexity when choosing among variance-reduction methods (Adan, MARS), memory-efficient variants (GaLore, Adafactor), and curvature-aware approaches (Sophia, Shampoo). The prefixed hyperparameter system, while powerful, requires understanding of which parameters are relevant for each optimizer class.

The implementation quality varies across optimizers: classical methods (AdamW, SGD with momentum) are highly optimized in PyTorch, while newer methods (Muon, Turbo-Muon, SOAP)

may require custom implementations that affect numerical stability and performance characteristics.

9.4 Deployment and Production Considerations

The quantization pipeline demonstrates practical deployment concerns but makes specific engineering trade-offs. Ternary weight packing reduces storage to approximately 1.58 bits per parameter, but this aggressive compression may degrade performance, especially for smaller models where quantization error is more significant. The current implementation applies quantization uniformly across parameter types.

The SBERT workflows provide practical utility for semantic similarity and retrieval tasks, but implementation assumes standard pooling strategies (CLS token, mean pooling). Recent advances such as Matryoshka embeddings [25] and contrastive learning refinements [15] are not yet incorporated.

9.5 Integration and Extensibility Challenges

The current codebase structure, while functional, presents maintenance challenges as architecture and optimizer families expand. The dispatcher pattern for mixer selection and optimizer routing handles extensibility but risks becoming a "kitchen sink" of conditional logic. Future versions would benefit from plugin-based architectures where new mixers and optimizers could be registered declaratively rather than modifying core dispatch logic.

The web configuration interface provides significant usability improvements but introduces deployment complexity: running Streamlit alongside training jobs requires additional resources and infrastructure considerations that may not be appropriate for all environments, particularly HPC clusters without web access.

10 Conclusion

Transformer Encoder Frankenstein presents a unified, configuration-driven experimentation platform addressing critical challenges in modern deep learning research: architectural fragmentation across dense attention, recurrent models, sparse patterns, and gated mechanisms; optimizer landscape complexity spanning classical baselines, variance-reduction methods, memory-efficient variants, schedule-free approaches, curvature-aware algorithms, and geometry-oriented methods; and end-to-end deployment workflows spanning quantization and sentence embedding applications.

The system's primary contributions are:

1. **Schema-First Design**: A strict YAML-based configuration contract with validation and prefixed hyperparameter routing enabling reproducible experiments across seventeen mixer architectures and twenty optimizer families.

2. **Comprehensive Architecture Support**: Implementation spanning major research categories including dense baselines (standard, sigmoid attention), recurrent alternatives (RetNet, Mamba, ODE-style, Titans), sparse attention (Sparse Transformer, Longformer, BigBird, SparseK, NSA, SpargeAttn, FASA), and gated mechanisms (GLA, DeltaNet, Gated DeltaNet, HGRN2, FoX, Gated Softmax).

3. **Unified Optimizer Framework**: Prefixed hyperparameter groups enabling fine-grained control over embeddings, normalization layers, recurrent blocks, attention weights, and FFN parameters across variance-reduction (Adan, ADOPT, AdEMAMix, MARS, Cautious), memory-efficient (Adafactor, GaLore, Lion), schedule-free (Schedule-Free AdamW, Prodigy), curvature-

aware (Sophia), second-order (Shampoo, SOAP), and geometry-oriented (Muon, Turbo-Muon) optimizers.

4. **End-to-End Workflows**: Integrated deployment pipeline supporting ternary weight packing and INT8 activation quantization; SBERT-inspired training and inference for semantic similarity, retrieval, and clustering tasks.

5. **Interactive Configuration**: Streamlit-based web interface providing schema-driven form generation, real-time validation, inline documentation, and CLI command synthesis.

This system enables rapid experimental iteration while maintaining reproducibility through strict configuration contracts. By consolidating diverse research contributions into a unified toolkit, it lowers barriers to exploring novel architectures and optimization strategies, particularly for researchers who may lack resources to implement and validate each variant independently.

10.1 Limitations and Future Directions

Several limitations and promising directions for future work emerge from this system's design and implementation:

1. **Architecture Integration**: Recent hybrid architectures (Griffin, Jamba, Mamba-X) demonstrate benefits of combining multiple mechanisms into unified blocks. Future versions should integrate these architectures and explore systematic composition patterns.

2. **Advanced Quantization**: Current implementation uses uniform ternary packing across all parameters. Research on layer-wise, channel-wise, and importance-aware quantization suggests more sophisticated strategies could improve quality-efficiency trade-offs.

3. **Plugin-Based Extensibility**: The current dispatch pattern becomes increasingly complex with each new addition. A plugin architecture allowing declarative registration of new mixers, optimizers, and normalization methods would improve maintainability and reduce risk of bugs in core dispatch logic.

4. **Automated Hyperparameter Optimization**: The schema supports extensive hyperparameter spaces, but users must manually explore these spaces. Integration with Bayesian optimization, multi-armed bandit strategies, or gradient-based hyperparameter tuning could automate effective configuration discovery.

5. **Production Deployment**: The web interface improves usability but may not be appropriate for all deployment environments. Headless configuration modes, API-based configuration management, or improved CLI ergonomics could serve HPC and production workflows.

6. **Evaluation Benchmarking**: While the system enables training with diverse architectures, comprehensive benchmarking comparing performance across mixers and optimizers on standardized tasks would provide valuable guidance for configuration selection.

7. **Training Stability Guarantees**: Current implementation includes NaN/Inf guards and gradient clipping, but formal analysis of stability conditions for different mixer-optimizer combinations, particularly with looped blocks and aggressive quantization, remains open.

8. **Multimodal and Task-Specific Extensions**: The current design focuses on sequence modeling. Extensions for vision-language models, multimodal architectures, and task-specific fine-tuning workflows (e.g., instruction tuning, RLHF) would broaden applicability.

The research trajectory of sequence modeling continues toward hybrid approaches that combine strengths of multiple paradigms—compression from recurrence, selectivity from attention, gating for memory management, and sparsity for efficiency. A unified experimentation platform like Transformer Encoder Frankenstein is increasingly valuable as this convergence accelerates, enabling researchers to systematically explore this expanding design space with reproducible, well-engineered infrastructure.

Bibliography

[1] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, . URL http://arxiv.org/abs/2501.00663.

[2] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, . URL https://arxiv.org/abs/2501.00663. Version Number: 1.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

[4] Thibaut Boissin, Thomas Massena, Franck Mamalet, and Mathieu Serrurier. Turbo-muon: Accelerating orthogonality-based optimization with pre-conditioning. URL https://arxiv.org/abs/2512.04632. Version Number: 1.

[5] Riccardo Bravin, Massimo Pavan, Hazem Hesham Yousef Shalby, Fabrizio Pittorino, and Manuel Roveri. EmbBERT: Attention under 2 MB memory. URL http://arxiv.org/abs/2502.10001.

[6] Mingzhi Chen, Taiming Lu, Jiachen Zhu, Mingjie Sun, and Zhuang Liu. Stronger normalization-free transformers, . URL http://arxiv.org/abs/2512.10938.

[7] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms, . URL https://arxiv.org/abs/2302.06675. Version Number: 4.

[8] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019. URL https://arxiv.org/abs/1904.10509.

[9] Aaron Defazio, Xingyu Alice Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled. URL https://arxiv.org/abs/2405.15682. Version Number: 4.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. URL http://arxiv.org/abs/1810.04805.

[11] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. URL https://arxiv.org/abs/2312.00752. Version Number: 2.

[12] Albert Gu, AI21 Labs, et al. Jamba: A hybrid transformer-mamba language model, 2024. URL https://arxiv.org/abs/2403.19887.

[13] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. URL https://arxiv.org/abs/1802.09568. Version Number: 2.

[14] Sukjun Hwang, Aakash Lahoti, Tri Dao, and Albert Gu. Hydra: Bidirectional state space models through generalized matrix mixers. URL http://arxiv.org/abs/2407.09941.

[15] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2020. URL https://arxiv.org/abs/2004.11362.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

[17] Mike Lewis, Shruti Bhosale, Tim Dettmers, Douwe Kiela, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models, 2021. URL https://arxiv.org/abs/2103.16716.

[18] Kaizhao Liang, Lizhang Chen, Bo Liu, and Qiang Liu. Cautious optimizers: Improving training with one line of code. URL https://arxiv.org/abs/2411.16085. Version Number: 4.

[19] Zhixuan Lin, Ke Wang, et al. Forgetting transformer: Softmax attention with a forget gate, 2025. URL https://arxiv.org/abs/2503.02130.

[20] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training, . URL https://arxiv.org/abs/2305.14342. Version Number: 4.

[21] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond, . URL https://arxiv.org/abs/1908.03265. Version Number: 4.

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. URL https://arxiv.org/abs/1711.05101. Version Number: 3.

[23] Tianyu Lou, Zheyu Chen, Tao Yu, et al. Efficient sparse attention for long-range transformers, 2024. URL https://arxiv.org/abs/2406.16747.

[24] Konstantin Mishchenko and Aaron Defazio. Prodigy: An expeditiously adaptive parameter-free learner. URL https://arxiv.org/abs/2306.06101. Version Number: 4.

[25] Niklas Muennighoff et al. Matryoshka representation learning, 2022. URL https://arxiv.org/abs/2205.13147.

[26] Matteo Pagliardini, Pierre Ablin, and David Grangier. The AdEMAMix optimizer: Better, faster, older. URL https://arxiv.org/abs/2409.03137. Version Number: 2.

[27] Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. doi: 10.1016/0041-5553(64)90137-5.

[28] Zhen Qin, Xu Han, et al. Hgrn2: Gated linear rnns with state expansion, 2024. URL https://arxiv.org/abs/2404.07904.

[29] Yuxiang Qiu, Qwen Team, et al. Gated attention for large language models, 2025. URL https://arxiv.org/abs/2505.06708.

[30] Jason Ramapuram, Federico Danieli, Eeshan Dhekane, Floris Weers, Dan Busbridge, Pierre Ablin, Tatiana Likhomanenko, Jagrit Digani, Zijin Gu, Amitis Shidani, and Russ Webb. Theory, analysis, and best practices for sigmoid self-attention. URL https://arxiv.org/abs/2409.04431. Version Number: 2.

[31] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. URL http://arxiv.org/abs/1908.10084.

[32] Hema Hariharan Samson. Lightweight transformer architectures for edge devices in real-time applications. URL http://arxiv.org/abs/2601.03290.

[33] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. URL https://arxiv.org/abs/1804.04235. Version Number: 1.

[34] Wei Shen, Ruichuan Huang, Minhui Huang, Cong Shen, and Jiawei Zhang. On the convergence analysis of muon. URL https://arxiv.org/abs/2505.23737. Version Number: 1.

[35] Rafael Soares et al. Griffin: Mixing gated linear recurrences with local attention for efficient sequence modeling, 2024. URL https://arxiv.org/abs/2402.19427.

[36] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. URL https://arxiv.org/abs/2307.08621. Version Number: 4.

[37] Shohei Taniguchi, Keno Harada, Gouki Minegishi, Yuta Oshima, Seong Cheol Jeong, Go Nagahara, Tomoshi Iiyama, Masahiro Suzuki, Yusuke Iwasawa, and Yutaka Matsuo. ADOPT: Modified adam can converge with any $_2$ with the optimal rate. URL https://arxiv.org/abs/2411.02853. Version Number: 3.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. URL https://arxiv.org/abs/1706.03762. Version Number: 7.

[39] Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing shampoo using adam. URL https://arxiv.org/abs/2409.11321. Version Number: 2.

[40] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. BitNet: Scaling 1-bit transformers for large language models. URL http://arxiv.org/abs/2310.11453.

[41] Zhe Wang, Ming Liu, et al. Fasa: Frequency-aware sparse attention, 2026. URL https://arxiv.org/abs/2602.03152.

[42] Xingyu Xie, Pan Zhou, Huan Li, Zhouchen Lin, and Shuicheng Yan. Adan: Adaptive nesterov momentum algorithm for faster optimizing deep models. URL https://arxiv.org/abs/2208.06677. Version Number: 5.

[43] Haiqi Yang, Zhiyuan Li, Yi Chang, and Yuan Wu. A survey of retentive network. URL http://arxiv.org/abs/2506.06708.

[44] Songlin Yang, Bailin Wang, et al. Gated linear attention transformers with hardware-efficient training, 2023. URL https://arxiv.org/abs/2312.06635.

[45] Songlin Yang, Bailin Wang, et al. Parallelizing linear transformers with the delta rule over sequence length, 2024. URL https://arxiv.org/abs/2406.06484.

[46] Songlin Yang, Bailin Wang, et al. Gated delta networks: Improving mamba2 with delta rule, 2024. URL https://arxiv.org/abs/2412.06464.

[47] Han Yuan, DeepSeek-AI, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

[48] Huizhuo Yuan, Yifeng Liu, Shuang Wu, Xun Zhou, and Quanquan Gu. MARS: Unleashing the power of variance reduction for training large models. URL https://arxiv.org/abs/2411.10438. Version Number: 4.

[49] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Pike Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, 2020. doi: 10.48550/ARXIV.2007.14062. URL https://arxiv.org/abs/2007.14062.

[50] Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. URL http://arxiv.org/abs/1910.07467.

[51] Jing Zhang, Peng Zhang, Baiwen Kong, Junqiu Wei, and Xin Jiang. Continuous self-attention models with neural ODE networks. 35(16):14393–14401. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v35i16.17692. URL https://ojs.aaai.org/index.php/AAAI/article/view/17692.

[52] Yichi Zhang, Yizhong Wang, et al. Accurate and training-free sparse attention accelerating any model inference, 2025. URL https://arxiv.org/abs/2502.18137.

[53] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. URL https://arxiv.org/abs/2403.03507. Version Number: 2.

[54] Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. URL http://arxiv.org/abs/2503.10622.

A Annex A: Optimizer Families from `OPTIMIZERS.md`

A.1 The Evolution of Optimization in Neural Networks

The optimizer survey frames transformer optimization as a response to three structural pressures: non-convex loss landscapes, severe curvature heterogeneity across parameter blocks, and the memory cost of storing optimizer state for very large models. The report argues that the field has diverged into several trajectories: adaptive first-order baselines, variance-reduction methods, memory-efficient methods, structured second-order preconditioners, schedule-free methods, and orthogonality-oriented updates.

A.2 Standard Baseline and Adaptive Optimizers

SGD with Momentum. The classical update accumulates a momentum buffer and then applies a fixed learning rate. Its strengths are low memory overhead and strong generalization when tuned carefully. Its main weakness in transformer workloads is poor robustness to heterogeneous curvature and a high dependence on learning-rate schedules.

Adam and AdamW. Adam tracks first and second moments of the gradient; AdamW decouples weight decay from the adaptive step. The report treats AdamW as the practical baseline for transformer fine-tuning because it converges quickly and is relatively forgiving. The tradeoff is state cost, since both moment tensors must be stored for every parameter.

RAdam. RAdam introduces variance rectification in early training, motivated by the observation that Adam's adaptive denominator is unreliable during initial steps. It is positioned as a way to reduce warmup sensitivity without abandoning the Adam family.

A.3 Advanced Momentum and Variance Reduction (2024–2025)

Adan. Adan reformulates Nesterov-style momentum so that it does not need the extra forward/backward pass of classical Nesterov acceleration. The survey emphasizes its fast convergence across CNNs, GANs, and transformers, but also notes the cost of keeping three momentum-like states.

ADOPT. ADOPT modifies Adam's update ordering to address theoretical non-convergence issues. In the report it is presented as a drop-in adaptive optimizer with stronger convergence guarantees and broad empirical performance, especially when one wants Adam-like behavior with fewer theoretical caveats.

AdEMAMix. AdEMAMix mixes short-horizon and long-horizon exponential moving averages. The key idea is to combine fast adaptation with slower historical smoothing so the optimizer can respond to sharp local changes without discarding longer-term signal.

MARS. MARS belongs to the variance-reduction line of work. The survey frames it as a way to make preconditioned adaptive optimization more stable by correcting momentum recursion and reducing gradient noise.

Cautious Optimizers. Cautious AdamW and related variants mask updates when the momentum direction and the current gradient disagree. The intended effect is to suppress harmful steps and keep only sign-consistent motion.

A.4 Large-Batch, Memory-Efficient, and Parameter-Free Optimizers

LAMB. LAMB is included in the survey as a large-batch optimizer that scales updates layerwise. Its practical role is to keep optimization stable when batch sizes become very large.

Schedule-Free AdamW. Schedule-free methods remove explicit scheduler design from the optimization recipe. The markdown emphasizes operational simplicity: instead of investing effort in warmup and decay design, one can use an optimizer whose update dynamics absorb part of that responsibility.

Adafactor. Adafactor factorizes second-moment statistics for matrix-shaped parameters, dramatically reducing optimizer-state memory. It is most attractive when memory is the bottleneck and some loss in optimizer simplicity is acceptable.

GaLore. GaLore projects gradients into a low-rank subspace before optimization. The report treats it as a complementary memory-saving path that is especially relevant when the model is too large for full-rank optimizer state.

Prodigy. Prodigy is grouped under parameter-free or distance-adaptive methods. The core claim is that it estimates effective step sizes from optimization geometry, reducing the need for explicit learning-rate tuning.

A.5 Second-Order, Geometric, and Orthogonality Optimizers

Shampoo. Shampoo computes matrix preconditioners from Kronecker-structured statistics. It is one of the most explicit second-order methods in the report and is motivated by conditioning improvements rather than minimal implementation complexity.

SOAP. SOAP keeps Adam-style tracking in the eigenbasis of a Shampoo-like preconditioner. The survey presents it as a hybrid between full matrix preconditioning and adaptive first-order behavior.

Lion. Lion uses sign momentum updates and therefore carries much smaller state than Adam-like methods. The markdown positions it as a low-memory, high-throughput alternative rather than a universally superior optimizer.

Sophia. Sophia uses approximate second-order information through diagonal Hessian estimates and clipped updates. It belongs to the family of curvature-aware methods that seek better conditioning without paying the full cost of dense second-order optimization.

Muon and Turbo-Muon. Muon and Turbo-Muon are described as orthogonality-oriented optimizers that explicitly reshape update geometry. Turbo-Muon adds faster Newton–Schulz-style orthogonalization, making the same basic idea more practical at scale.

Group	Methods	Primary Goal	Interpretation from the Survey
Classical baseline	SGD, AdamW, RAdam	stability and reference baselines	These define the comparison floor for newer optimizer claims.
Momentum redesign	Adan, AdEMAMix, MARS, Cautious AdamW	faster or safer first-order adaptation	Best when convergence speed or noisy-gradient stability is the main concern.
Large-batch and schedule simplification	LAMB, Schedule-Free AdamW	operational robustness at scale	Reduce brittleness from batch-size growth or schedule engineering.
Memory-efficient	Adafactor, GaLore, Lion	optimizer-state reduction	Most useful when VRAM is dominated by optimizer state rather than activations.
Curvature-aware	Shampoo, SOAP, Sophia	better conditioning	Prefer when richer geometry is worth implementation and compute overhead.
Geometry-oriented	Muon, Turbo-Muon	orthogonalized update structure	Specialized options for matrix geometry and representation shaping.

B Annex B: Transformer Families from `TRANSFORMER_TYPES.md`

B.1 Standard Attention

The markdown presents standard attention as the reference architecture for global contextualization. Every token attends to every other token through a softmax-normalized similarity matrix. Its major advantage is expressiveness: it can preserve perfect historical recall inside the active context window and it parallelizes well during training. Its main limitations are the quadratic training footprint and the KV-cache burden during decoding.

B.2 Sigmoid Attention

Sigmoid attention keeps the same query–key similarity matrix but replaces row-wise softmax normalization with an elementwise sigmoid. The survey highlights three implications: token competition is reduced, the computation avoids some row-wise synchronization, and the method

can be more hardware-friendly. The cost is training instability at scale, which motivated the "hybrid-norm" stabilization techniques discussed in the source document.

B.3 RetNet

RetNet is presented as a bridge between attention and recurrence. In parallel form it resembles an attention-like interaction masked by exponential decay; in recurrent form it compresses history into a fixed-size state updated with decay. The report emphasizes its triple computation modes: parallel, recurrent, and chunkwise recurrent. The main tradeoff is that the fixed decay law imposes a stronger inductive bias than unconstrained softmax attention.

B.4 Mamba

Mamba replaces explicit attention with a selective state-space recurrence whose parameters depend on the input. The annex source stresses that its importance lies not only in linear asymptotic complexity but also in the hardware-aware scan algorithm that makes the linear recurrence practical on accelerators. The downside is that state compression can weaken exact recall and copying behavior relative to full attention.

B.5 ODE Transformer

The ODE transformer treats depth as numerical integration over a continuous dynamical system. The source discusses Runge–Kutta refinement as a way to reduce truncation error and improve sequence generation quality. The benefit is a more expressive intra-block trajectory with weight sharing; the drawback is higher cost because each block now behaves like several solver stages.

B.6 Titans

Titans introduces test-time neural memorization: instead of storing only activations, the model updates an internal memory using surprise-driven learning signals during inference. The markdown frames this as a way to handle extremely long contexts and associative recall beyond what fixed-state recurrent models typically support. The systems cost is higher implementation complexity because inference now includes memory adaptation logic.

B.7 Synthesis and Systemic Insights

The transformer survey ends with four synthesis claims:

- sequence modeling is shaped by an expressivity-versus-compression tradeoff,
- hardware substrate constraints now strongly influence which architectures win,
- continuous-time views provide a useful language for understanding depth and refinement,
- the field is converging toward hybrid models that mix compression, recurrence, sparse routing, and test-time adaptation.

Architecture	**Training Trend**	**Inference Trend**	**Main Characterization in the Markdown**
Standard Attention	quadratic	KV-cache based, linear per step	Highest expressiveness and direct token routing, but bottlenecked by dense pairwise interactions.

Architecture	Training Trend	Inference Trend	Main Characterization in the Markdown
Sigmoid Attention	quadratic	linear per step	Removes zero-sum competition and improves kernel behavior, but needs stabilization for large-scale training.
RetNet	chunkwise or quadratic	constant-state recurrent inference	Unifies attention-style training with recurrent deployment through retention and decay.
Mamba	linear	constant-state recurrent inference	Selective state-space model with hardware-aware scan; strong long-context efficiency.
ODE Transformer	solver-dependent	solver-dependent	Continuous-depth interpretation with accuracy gains from multi-stage numerical integration.
Titans	roughly linear in sequence length	memory-retrieval centered	Adds test-time adaptive memorization for extreme context and recall.

C Annex C: Sparse Attention Families from `SPARSE_TRANSFORMER_TYPES.md`

C.1 Executive Summary

The sparse-attention report treats sparsity as a full design space rather than a single approximation trick. The methods vary along three axes: whether the sparse pattern is fixed or data-dependent, whether the mechanism is trainable or training-free, and whether the sparsity unit is a token, a window, or a block.

C.2 Sparse Transformer

Sparse Transformer uses factorized sparse masks built from strided and fixed connectivity patterns. The markdown emphasizes that two sparse heads can approximate full reachability with much lower cost than dense attention. The advantages are strong early empirical results and sub-quadratic complexity; the drawback is that the pattern is data-agnostic.

C.3 Longformer

Longformer combines sliding-window locality, optional dilation, and task-specific global tokens. Its main value is practical linear scaling for long documents. The report notes that it is a drop-in replacement for standard attention in many settings, but window size and global-token selection remain task-dependent design choices.

C.4 BigBird

BigBird mixes local windows, random connections, and global tokens. The sparse-attention markdown highlights its theoretical guarantees: universal approximation and Turing completeness can be preserved with sparse graphs as long as enough global structure is retained.

C.5 FASA

FASA is a training-free decode-time method that predicts token importance from dominant frequency chunks in RoPE-based models. The key claims in the source are strong KV-cache compression and decoding speedup, plus compatibility with other compression methods. The

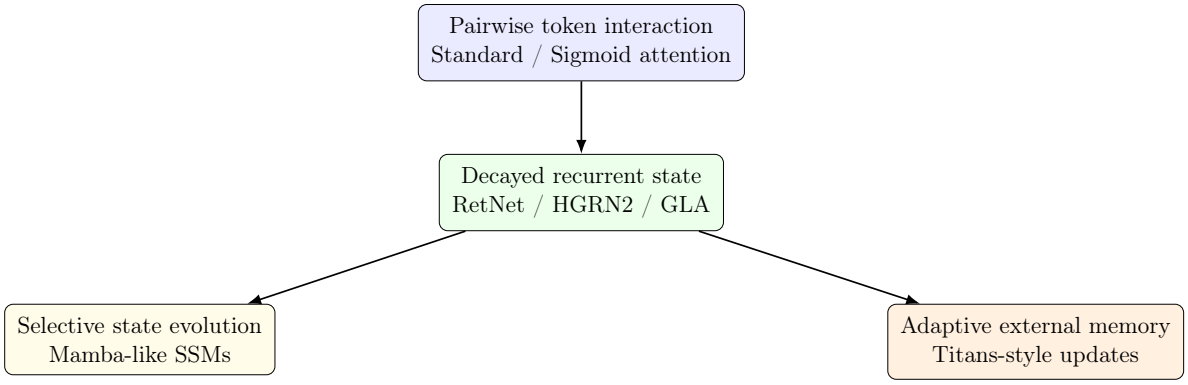

Figure 8: Annex view of transformer evolution: the field moves from explicit pairwise routing toward progressively more compressed or adaptive memory formulations.

limitations are its dependence on a frequency-analysis step and its focus on decoding rather than full training-time attention.

C.6 NSA

Native Sparse Attention is a trainable three-branch architecture composed of compressed, selected, and window branches. The report frames it as hardware-aligned sparse attention designed to use tensor cores efficiently. Its complexity is not only algorithmic but also architectural because the method introduces multiple branches and gating logic.

C.7 SparseK

SparseK uses a differentiable top-k selection mechanism to determine which key/value pairs should participate in attention. The report presents it as end-to-end trainable and especially attractive for generation because the active memory can remain small. The weakness is that top-k selection may lead to irregular memory access patterns.

C.8 SpargeAttn

SpargeAttn is a training-free universal sparse method that first predicts negligible interactions and then applies softmax-aware filtering. The survey highlights that it applies beyond language models to image and video diffusion workloads as well. Its performance depends on whether the model already contains enough inherent sparsity to exploit.

C.9 Comparison Pattern

Across the markdown source, the methods naturally group into:

1. patterned sparsity: Sparse Transformer and Longformer,
2. graph sparsity: BigBird,
3. learned or predicted selection: SparseK and NSA,
4. training-free acceleration: SpargeAttn and FASA.

Method	Mechanism	Trainable	Sparsity Unit	Main Takeaway from the Markdown
Sparse Transformer	strided + fixed factorized masks	Yes	token neighborhood	Structured reachability lowers cost but remains data-agnostic.
Longformer	sliding window + dilation + global tokens	Yes	local window + global anchors	Practical linear scaling for long documents with selective global access.
BigBird	random + local + global sparse graph	Yes	sparse edge set / blocks	Preserves strong theoretical properties while remaining sparse.
FASA	frequency-aware decode-time selection	No	selected tokens / cache entries	Training-free KV compression guided by RoPE frequency structure.
NSA	compressed + selected + window branches	Yes	branchwise reduced views	Hardware-aligned trainable sparsity with learned branch fusion.
SparseK	differentiable top-k selection	Yes	selected key/value pairs	End-to-end selective attention with small active memory.
SpargeAttn	two-stage online block filtering	No	attention blocks	Training-free acceleration path for already trained dense models.

Method	Complexity Trend	Training-Free?	Pros / Cons Emphasized in the Markdown
Sparse Transformer	sub-quadratic	No	Strong early benchmarks and long-sequence reach, but fixed patterns may miss important interactions.
Longformer	linear in sequence length for fixed window	No	Scales well and is flexible, but still depends heavily on window and global-token design.
BigBird	near-linear	No	Strong theory and long-context performance, but randomness and block structure complicate tuning.
FASA	decode-time selective	Yes	Plug-and-play compression and speedup, but depends on RoPE/frequency analysis quality.
NSA	reduced-token multi-branch	No	High speedups and strong benchmark results, but requires custom kernels and extra architecture complexity.
SparseK	linear train / small active generation state	No	Differentiable and trainable, but adds scoring overhead and scattered access.
SpargeAttn	sparsity-dependent acceleration	Yes	Universal and training-free, but gains depend on how sparse the underlying model already is.

D Annex D: Gated Attention Families from `GATED_TRANSFORMER_TYPES.md`

D.1 Executive Summary

The gated-attention survey argues that gating is the missing control mechanism in many linear-attention and recurrent alternatives. Without it, memory only accumulates. With it, the model can forget, rewrite, or selectively scale information. The source splits the field into recurrent-state gating and softmax-path gating.

D.2 Gated Linear Attention (GLA)

GLA augments linear attention with a data-dependent multiplicative decay gate. The survey emphasizes that this directly addresses memory overload in additive recurrent states and that chunkwise training makes the method hardware friendly. The tradeoff is that the gate is still less expressive than full attention on retrieval-heavy tasks.

D.3 DeltaNet

DeltaNet replaces pure accumulation with a delta-rule update: the memory is corrected by comparing the retrieved value against the desired value. The source frames this as online error correction and connects it to test-time training ideas. Its strength is strong associative recall; its weakness is memory crowding because there is no explicit global forgetting.

D.4 Gated DeltaNet

Gated DeltaNet combines the two ideas above: a decay gate controls forgetting while the delta rule controls targeted writing. In the markdown this is presented as the most balanced pure linear recurrent design because it supports rapid erasure and precise updates at the same time.

D.5 RetNet

RetNet appears again in the gated survey because fixed exponential decay can be interpreted as a gate. The report contrasts it with fully data-dependent methods: RetNet is simpler and efficient, but its forgetting pattern is predetermined rather than learned token by token.

D.6 HGRN2

HGRN2 uses lower-bounded hierarchical forget gates with outer-product state expansion. The markdown stresses that the hierarchical bounds encourage different layers to specialize to different timescales, though the method still lags behind stronger retrieval-oriented gated models.

D.7 Forgetting Transformer (FoX)

FoX keeps full softmax attention but adds a forget gate in logit space. The report treats it as a conservative modification for users who want the expressive power of dense attention while still imposing recency-aware memory control. Its main limitation remains quadratic complexity.

D.8 Gated Attention after SDPA

The final architecture in the source applies a sigmoid gate after scaled dot-product attention. The survey highlights two claimed benefits: the extra nonlinearity breaks part of the low-rank bottleneck in the output path, and the gate helps suppress the attention-sink phenomenon. This is the least disruptive gated variant because it leaves the core attention operator intact.

D.9 Taxonomy and Comparison Dimensions

The source also adds two useful comparison lenses:

- **state type**: fixed recurrent matrix state versus full attention matrix,
- **gate location**: recurrent decay, write strength, logit bias, or post-attention modulation.

Method	Gate Location	Dense Softmax?	Main Interpretation in the Markdown
GLA	recurrent state decay	No	Adds forgetting to linear attention to avoid uncontrolled memory accumulation.
DeltaNet	write strength in recurrent update	No	Uses error-correcting writes for targeted memory replacement.
Gated DeltaNet	decay + write gates	No	Combines global forgetting with local corrective memory editing.
RetNet	fixed decay in retention state	No	Uses deterministic multi-scale decay rather than a fully data-dependent gate.
HGRN2	lower-bounded recurrent forget gates	No	Hierarchical gating distributes time scales across depth.
FoX	attention-logit bias	Yes	Injects forget dynamics into the standard softmax pipeline.
Gated Softmax	post-attention output gate	Yes	Keeps SDPA intact and adds multiplicative modulation afterward.

Architecture	State Type	Recall Strength	Pros / Cons Emphasized in the Markdown
GLA	matrix recurrent state	moderate	Good length generalization and chunkwise efficiency, but still weaker than softmax on hard retrieval.
DeltaNet	matrix recurrent state	strong	Excellent associative recall and principled updates, but lacks global forgetting.
Gated DeltaNet	matrix recurrent state	very strong	Best-balanced pure linear recurrent design, but richer transitions reduce throughput.
RetNet	matrix recurrent state	weaker retrieval bias	Efficient and simple with no KV cache, but fixed decay is less adaptive.
HGRN2	matrix recurrent state	moderate	Multi-scale temporal modeling via hierarchical bounds, but lower recall than DeltaNet variants.
FoX	full attention path	very strong	Preserves softmax expressiveness and improves length extrapolation, but remains quadratic.
Gated Attention	full attention path	strong	Very simple modification with low overhead, but only applicable when dense SDPA is already present.