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

This document presents Transformer Encoder Frankenstein as a configuration-driven toolkit for experimentation with modern encoder blocks, optimizer families, quantized deployment, and sentence-embedding workflows. The paper is organized as a technical map: it first explains how the schema constrains the system, then compares the supported model families, optimizer families, deployment path, and SBERT workflows. To improve practical readability, the document includes architecture diagrams, execution-flow diagrams, decision tables, and appendices that condense the supporting literature on transformer variants, sparse attention, gated attention, and optimizers.

Contents

1 Introduction **4**
1.1 Reading Guide . . . 4

2 Configuration-Centric Architecture **4**
2.1 Schema Scope and Validation Rules . . . 6
2.2 Complete Model Feature Inventory . . . 6
2.3 Complete Training Feature Inventory . . . 7
2.4 Optimizer Prefix Contract (Full) . . . 8
2.5 Training Safety and Runtime Semantics . . . 9

3 Normalization Variants: RMSNorm, Dynamic Tanh, and Dynamic Erf **9**
3.1 RMSNorm . . . 9
3.2 Dynamic Tanh (DyT) . . . 10
3.3 Dynamic Erf (Derf) . . . 10
3.4 Schema Implications . . . 10

4 Attention and Sequence-Mixer Families **10**
4.1 Standard Attention . . . 11
4.2 Sigmoid Attention . . . 11
4.3 Retentive Formulation . . . 11
4.4 Selective SSM (Mamba) . . . 11
4.5 ODE-style Continuous Updates . . . 12
4.6 Test-time Memory (Titans) . . . 12
4.7 Sparse and Gated Extensions in the Current Codebase . . . 12
4.8 Implemented Sparse Attention Blocks (Detailed) . . . 12
4.9 Implemented Gated Attention Blocks (Detailed) . . . 13

5 Optimizer Families and Training Dynamics **16**
5.1 Core Adaptive Form . . . 16
5.2 Examples from the Supported Set . . . 16

6 Quantization and Deployment **17**
6.1 Ternary Quantization . . . 17
6.2 Activation Quantization . . . 17
6.3 Size Estimates . . . 17

7 SBERT Downstream Tasks **18**

8 Summary Tables **18**
8.1 Attention and Sequence-Mixer Summary . . . 18
8.2 Optimizer Summary . . . 19

9 Discussion **20**

10 Conclusion **20**

A Annex A: Optimizer Families from `OPTIMIZERS.md` **20**
A.1 The Evolution of Optimization in Neural Networks . . . 20
A.2 Standard Baseline and Adaptive Optimizers . . . 20
A.3 Advanced Momentum and Variance Reduction (2024–2025) . . . 21
A.4 Large-Batch, Memory-Efficient, and Parameter-Free Optimizers . . . 21
A.5 Second-Order, Geometric, and Orthogonality Optimizers . . . 21

B Annex B: Transformer Families from `TRANSFORMER_TYPES.md` **22**
B.1 Standard Attention . . . 22
B.2 Sigmoid Attention . . . 22
B.3 RetNet . . . 23
B.4 Mamba . . . 23
B.5 ODE Transformer . . . 23
B.6 Titans . . . 23
B.7 Synthesis and Systemic Insights . . . 23

C Annex C: Sparse Attention Families from `SPARSE_TRANSFORMER_TYPES.md` **24**
C.1 Executive Summary . . . 24
C.2 Sparse Transformer . . . 24
C.3 Longformer . . . 24
C.4 BigBird . . . 24
C.5 FASA . . . 24
C.6 NSA . . . 25
C.7 SparseK . . . 25
C.8 SpargeAttn . . . 25
C.9 Comparison Pattern . . . 25

D Annex D: Gated Attention Families from `GATED_TRANSFORMER_TYPES.md` **27**
D.1 Executive Summary . . . 27
D.2 Gated Linear Attention (GLA) . . . 27
D.3 DeltaNet . . . 27
D.4 Gated DeltaNet . . . 27
D.5 RetNet . . . 27
D.6 HGRN2 . . . 27

D.7 Forgetting Transformer (FoX) 27
D.8 Gated Attention after SDPA 27
D.9 Taxonomy and Comparison Dimensions 28

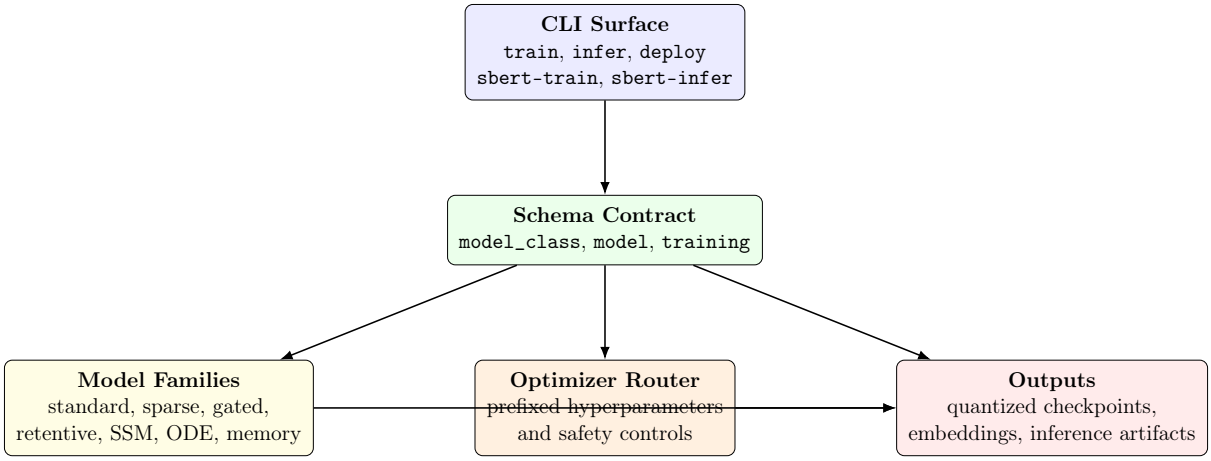

Figure 1: System view of the project: the CLI dispatches into a strict schema, which then controls architecture selection, optimizer behavior, and deploy/inference outputs.

1 Introduction

Transformer systems are now a production concern as much as a modeling concern [38, 14]. In practice, users need a single toolchain that can: (i) define training configurations with strict contracts, (ii) train with multiple optimizer and mixer families, (iii) deploy quantized checkpoints, and (iv) run sentence-embedding workflows inspired by SBERT [31].

The project command surface is:

`frankestein-transformer`

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

This paper is written to answer four operational questions:

1. What is the software contract that defines a valid experiment?
2. Which mixer or attention family should be used for a given memory/latency regime?
3. How are optimizer choices exposed through the schema and training loop?
4. How do deployment and sentence-embedding workflows connect back to the same model definition?

1.1 Reading Guide

Sections 2 and 3 describe the configuration contract and the supported model families. Section 4 explains normalization options exposed by the current schema. Section 5 covers optimizer routing and training dynamics. Sections 6 and 7 describe deployment and SBERT workflows. Appendix A–D act as annexes summarizing the supporting literature files in `docs/bibliography/`.

2 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`

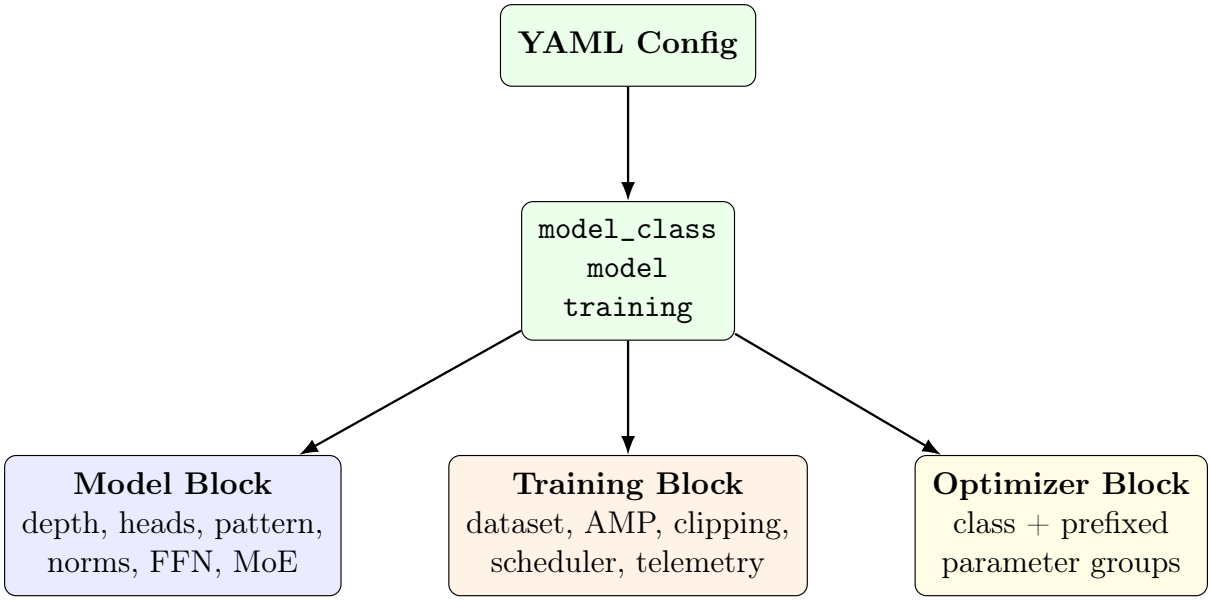

Figure 2: The schema acts as the experiment contract. The three top-level objects partition model definition, training runtime, and optimizer parameterization.

which corresponds to current attention and sequence-mixer literature [35, 16, 52, 2, 30, 38, 11, 3, 51, 45, 46, 47, 28, 21, 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.

2.1 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.

2.2 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.

2.3 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.

2.4 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

2.5 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 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:

3.1 RMSNorm

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

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

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

3.2 Dynamic Tanh (DyT)

Dynamic Tanh proposes replacing explicit normalization with a bounded elementwise map [56]:

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

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

3.3 Dynamic Erf (Derf)

Derf extends the same normalization-free direction by using an error-function based map [9]:

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

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

3.4 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	$y_i = \gamma_i x_i / \sqrt{\frac{1}{d}\sum_j x_j^2 + \epsilon}$	RMS only	Lower overhead than LayerNorm; widely used normalization baseline [**?**].
Dynamic Tanh	$\tanh(\alpha x)$	none	Normalization-free bounded transform; simple drop-in replacement direction [56].
Dynamic Erf (Derf)	$\mathrm{erf}(\alpha x + s)$	none	Normalization-free alternative designed to improve over DyT [9].

4 Attention and Sequence-Mixer Families

This is the most heterogeneous part of the project. The supported blocks fall into five practical groups:

1. dense attention baselines,
2. linear-recurrent or retentive alternatives,
3. continuous-depth blocks,
4. sparse attention families for longer contexts,
5. gated mechanisms that control forgetting or memory writes.

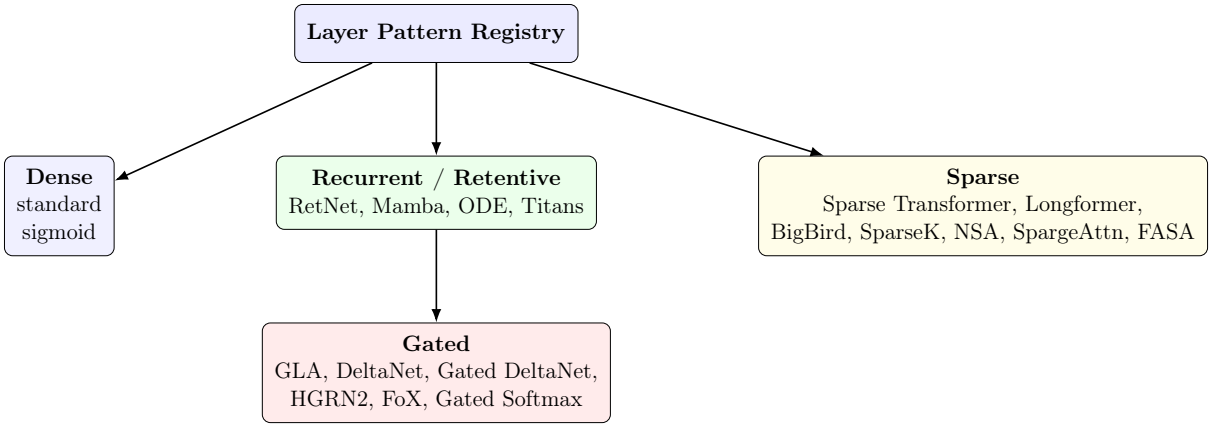

Figure 3: Taxonomy of the supported sequence mixers and attention blocks. The registry mixes classical dense attention, recurrent alternatives, sparse approximations, and gated memory mechanisms.

4.1 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].

4.2 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].

4.3 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 [35, 43].

4.4 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 [16, 19].

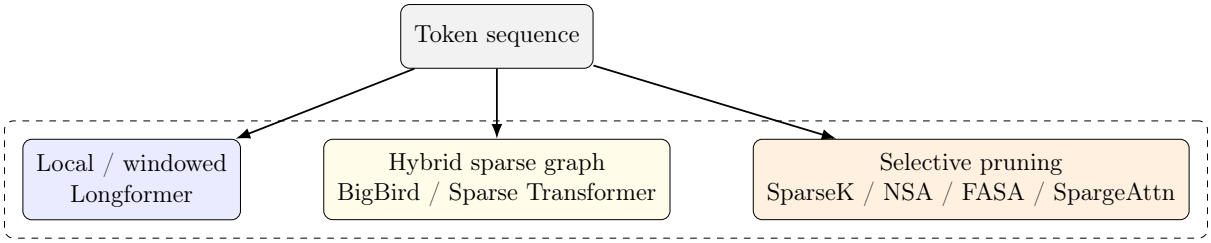

Figure 4: Conceptual map of sparse attention design choices used in the codebase. Different methods reduce cost by restricting neighborhoods, constructing sparse graphs, or selecting only high-value tokens/blocks.

4.5 ODE-style Continuous Updates

Continuous-depth framing:

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

with practical RK integrators for discrete execution [52].

4.6 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].

4.7 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.

4.8 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` [11, 3, 51, 25, 49, 54, 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. [11]

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. [51]

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. [25]

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

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

[49]

SpargeAttn (`sparge_attn`). Two-stage training-free block filtering: first predicts negligible block interactions, then applies softmax-aware pruning to remove low-contribution blocks. [54]

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.	[11]
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.	[51]
SparseK	Yes	linear-like (selected KV)	differentiable top-k KV selection	Learned score net + SparseK projection + gathered KV attention.	[25]
NSA	Yes	reduced-token multi-branch	compressed blocks + selected blocks + local window	Three sparse branches gated into one output tensor.	[49]
SpargeAttn	No (training-free)	sparse block dependent	block-level predicted sparsity	Eval-only in this repo; raises in training mode.	[54]
FASA	No (training-free)	selected-token dependent	dominant frequency chunks + selected tokens	Eval-only in this repo; raises in training mode.	[41]

4.9 Implemented Gated Attention Blocks (Detailed)

This codebase includes seven gated blocks aligned with `docs/bibliography/GATED_TRANSFORMER_TYPES.md` [45, 46, 47, 35, 28, 21, 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.

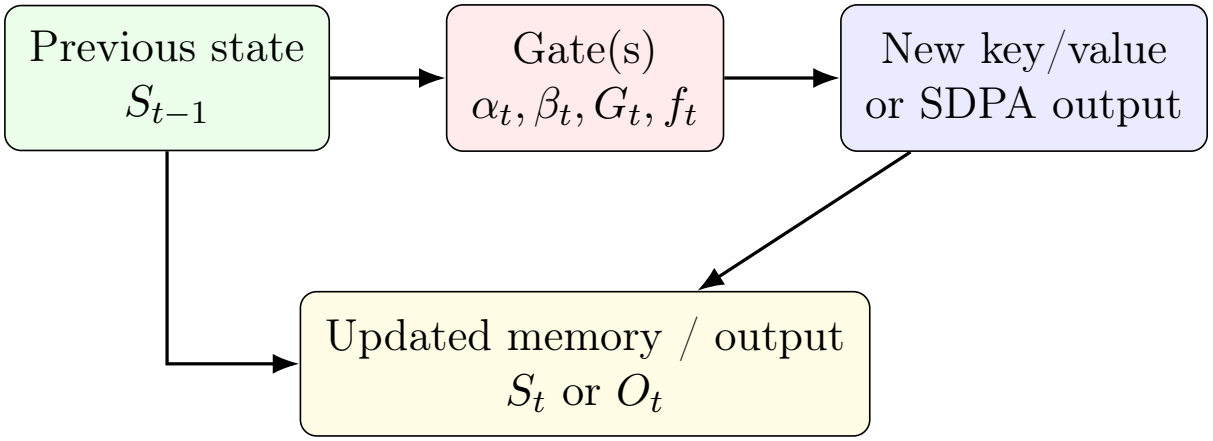

Figure 5: Generic gating template. A gate can decay existing memory, regulate write strength, or modulate dense attention outputs, depending on the block family.

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. [45]

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. [46]

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. [47]

RetNet Attn Alias (`retnet_attn`). Provides an explicit gated-package alias wrapping multi-scale retention behavior for naming consistency in layer registries. [35]

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. [21]

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

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

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

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.	[45]
DeltaNet	matrix recurrent state	write gate (β)	No (linear recurrent)	Delta-rule correction with normalized Q/K.	[46]
Gated DeltaNet	matrix recurrent state	decay + write gates (α, β)	No (linear recurrent)	Combined forgetting and targeted writing.	[47]
RetNet Attn	matrix recurrent state	fixed multi-scale decay	No (retention)	Alias wrapper over existing RetNet mixer.	[35]
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.	[21]
Gated Softmax	full attention matrix	post-attention sigmoid gate	Yes	Sigmoid gating applied after SDPA output.	[29]

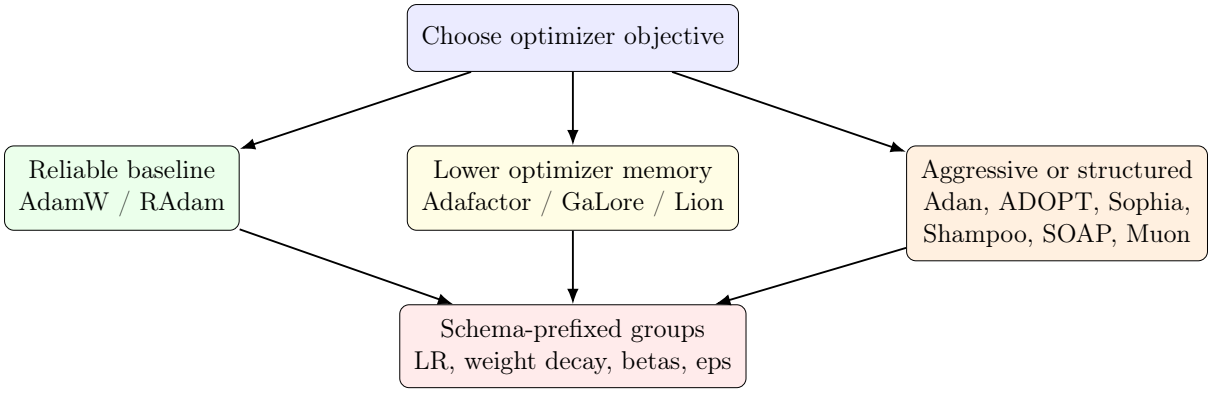

Figure 6: Optimizer selection in practice: the class determines the update rule, while the schema controls how hyperparameters are applied across embeddings, norms, recurrent blocks, attention blocks, and other parameters.

5 Optimizer Families and Training Dynamics

Optimizer support is broad because the project is intended as a research workbench, not a single-model training script. The schema therefore separates two concerns: selecting an optimizer class and routing the right prefixed hyperparameters to the correct parameter groups.

5.1 Core Adaptive Form

Many supported optimizers share moment tracking:

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

followed by preconditioned updates (e.g., AdamW) [24].

5.2 Examples from the Supported Set

- **RAdam**: variance rectification for early-step instability [23].
- **Adan**: adaptive Nesterov momentum for faster convergence [42].
- **ADOPT**: modified Adam order yielding stronger convergence guarantees [37].
- **AdEMAMix**: dual-EMA history mixing [27].
- **MARS**: variance reduction in preconditioned optimization [50].
- **Cautious optimizers**: sign-consistent masking of momentum updates [20].
- **Schedule-free**: remove explicit scheduler dependence [13].
- **Shampoo/SOAP**: matrix preconditioning families [18, 39].
- **Adafactor/GaLore**: memory reduction via factorization or low-rank projection [33, 55].
- **Prodigy/Lion/Sophia**: parameter-free adaptation, sign momentum, and clipped second-order scaling [26, 10, 22].
- **Muon/Turbo-Muon**: orthogonality-oriented updates with acceleration [34, 4].

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 [24]
else if $c =$ `radam` **then**
    apply rectified adaptive step [23]
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 [55]
else
    dispatch to selected optimizer implementation
end if
return updated θ

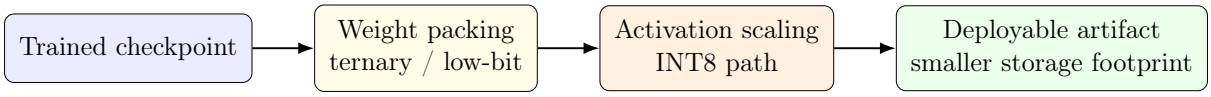

Figure 7: Deployment path from a trained checkpoint to a compact artifact. The codebase treats quantization as a deploy-stage transformation rather than a separate model family.

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.

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].

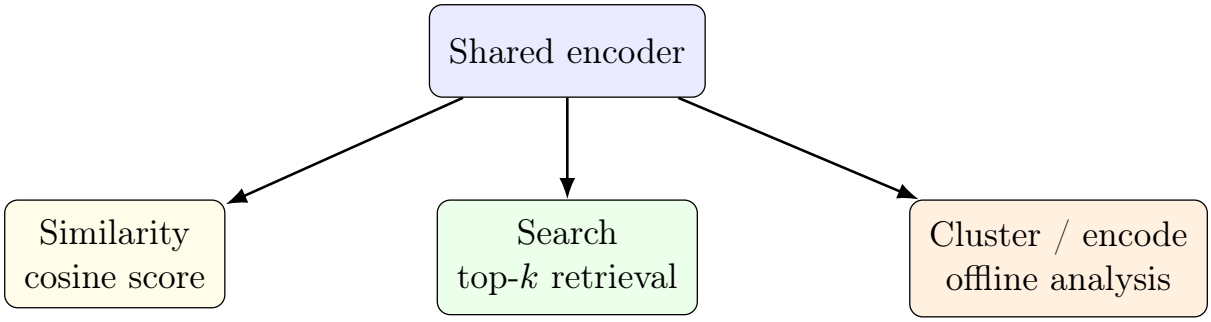

Figure 8: SBERT workflow reuse. A single encoder supports online pair scoring, corpus retrieval, clustering, and persistent embedding export.

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

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	$\text{softmax}(QK^\top/\sqrt{d_k})V$	$\mathcal{O}(n^2d)$	$\mathcal{O}(n)$/step	Baseline expressive global routing [38].
Sigmoid Attention	$\sigma(QK^\top/\sqrt{d_k}+b)V$	$\mathcal{O}(n^2d)$	$\mathcal{O}(n)$/step	Element-wise gating; often needs stabilization norm [30].
RetNet	$(QK^\top \odot D)V$	$\mathcal{O}(n^2d)$ or chunkwise	$\mathcal{O}(1)$/step	Parallel/recurrent dual form with decay retention [35].
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 [16].
ODE-style block	$\frac{dh}{dt} = f_\theta(h,t)$	solver-dependent	solver-dependent	Continuous-depth interpretation; RK integration [52].
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	[24]
RAdam	Adaptive variance-corrected	High	Rectifies early adaptive variance	[23]
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	[27]
MARS	Variance-reduced preconditioned	High	Recursive momentum correction	[50]
Cautious AdamW	Masked momentum	High	Apply updates only on sign-consistent directions	[20]
Schedule-free AdamW	Scheduler-free adaptive	High	Remove explicit LR schedule dependence	[13]
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	[55]
Prodigy	Parameter-free adaptation	Medium	Distance-adaptive step calibration	[26]
Lion	Sign momentum	Low	Momentum sign update, reduced state	[10]
Sophia	Approx. second-order	Medium	Diagonal Hessian preconditioning with clipping	[22]
Shampoo	Matrix preconditioner	High	Kronecker-structured second-order statistics	[18]
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

From an engineering perspective, the toolkit couples modern research ideas with reproducible interfaces:

- Explicit schema contracts lower configuration ambiguity.
- Multiple attention/mixer families let users tune for context length, latency, and memory.
- Broad optimizer support enables controlled studies over convergence and stability.
- Quantized deployment reduces artifact size and improves portability.
- SBERT workflows cover practical retrieval and semantic similarity tasks.

The design also aligns with multilingual and compact-model directions in the literature [8, 36, 6, 15].

10 Conclusion

Transformer Encoder Frankenstein is positioned as a practical experimentation platform: a strict configuration schema, extensible optimizer and attention families, deploy-time quantization, and sentence embedding workflows in one CLI. This makes it suitable for both rapid iteration and reproducible model operations.

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

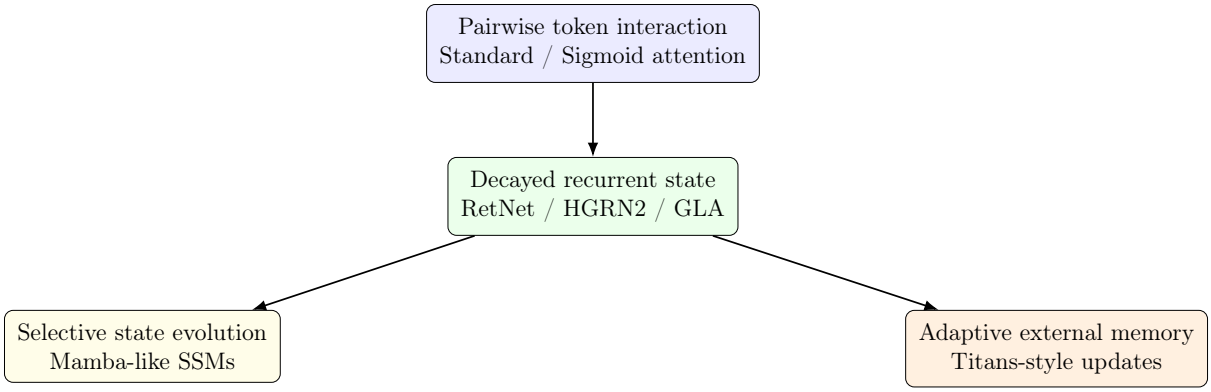

Figure 9: Annex view of transformer evolution: the field moves from explicit pairwise routing toward progressively more compressed or adaptive memory formulations.

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.

References

[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] Minh Duc Bui, Fabian David Schmidt, Goran Glavaš, and Katharina von der Wense. Knowledge distillation vs. pretraining from scratch under a fixed (computation) budget. URL http://arxiv.org/abs/2404.19319.

[7] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. pages 1–20. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2025.3554028. URL http://arxiv.org/abs/2407.06204.

[8] José Cañete, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, and Jorge Pérez. Spanish pre-trained BERT model and evaluation data. URL http://arxiv.org/abs/2308.02976.

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

[10] 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.

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

[12] Chang Dai, Hongyu Shan, Mingyang Song, and Di Liang. HoPE: Hyperbolic rotary positional encoding for stable long-range dependency modeling in large language models. URL http://arxiv.org/abs/2509.05218.

[13] 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.

[14] 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.

[15] Taj Gillin, Adam Lalani, Kenneth Zhang, and Marcel Mateos Salles. BERT-JEPA: Reorganizing CLS embeddings for language-invariant semantics. URL http://arxiv.org/abs/2601.00366.

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

[17] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, . URL http://arxiv.org/abs/2312.00752.

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

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

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

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

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

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

[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] 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 http://arxiv.org/abs/2307.08621.

[36] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: a compact task-agnostic BERT for resource-limited devices, . URL http://arxiv.org/abs/2004.02984.

[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] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms, . URL http://arxiv.org/abs/2311.12424.

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

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

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

[48] Daryl Noupa Yongueng and Hamidou Tembine. Holonorm. URL http://arxiv.org/abs/2511.10504.

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

[50] 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.

[51] 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.

[52] 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.

[53] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity, . URL http://arxiv.org/abs/1905.11881.

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

[55] 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.

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