**Optimizers Under Sparse & Non-Stationary Gradients**

**AUTHORS: Mahmoud Abo Shukr, Ramzi Amira**

**AFFILIATION: Paris-Saclay University - M2 MMVAI**

Project focus: benchmarking optimizers under gradient sparsity and non-stationarity (directional drift).

# Abstract

We study how different optimization algorithms behave when gradients are both sparse and non-stationary (i.e., their direction and statistics drift over training). Under a controlled protocol where the dataset, model, training budget, and non-optimizer hyperparameters are fixed, we benchmark multiple optimizers and log

1. training loss
2. gradient sparsity
3. drift proxy based on cosine similarity between consecutive gradient vectors.

Using the provided logs (seed=42), Adafactor, Stable-SPAM, AdaBelief, and Lion achieve the lowest observed losses while operating in highly sparse regimes, whereas several Adam-family variants converge to higher final loss under the same budget. The results highlight optimizer-dependent tradeoffs between stability, adaptation to drift, and effectiveness under sparse updates.

# Idea overall

* Modern deep networks can exhibit gradient sparsity: many parameters receive near-zero updates at a given step.
* Gradients can also be non-stationary: the descent direction changes over time due to data dynamics, curriculum effects, or optimization state.
* Optimizers that rely on historical statistics (momentum / second moments) may become misaligned when gradients drift, especially when updates are sparse.
* Goal: empirically compare optimizers and characterize their behaviors using both performance (loss) and gradient diagnostics (sparsity, drift).

# Multiple stages of optimizers

**Stage 0 - Controlled setup:** Fix dataset, model, seed(s), training steps, and non-optimizer hyperparameters. Change only the optimizer.

**Stage 1 - Baselines:** Run a small set of reference optimizers to validate the pipeline (e.g., SGD, AdamW, Adafactor, Lion).

**Stage 2 - Full sweep:** Compare the full optimizer suite under identical conditions.

**Stage 3 - Diagnostics:** Log loss + gradient sparsity (%) + gradient directional drift (cosine similarity).

**Stage 4 - Analysis:** Rank optimizers by best/final loss, assess stability (spikes/plateaus), and interpret sparsity + drift patterns.

# Our test (controlled protocol)

**26. Dataset & Task:**

**WikiText-103**. Task: **Causal Language Modeling (Next-Token Prediction)**. Split: Standard Train/Test splits. Tokenizer: GPT-2 Byte-Pair Encoding (BPE).

**27. Model:**

**Custom GPT-2 (Sparse-Config)**.

* **Dimensions:** Embedding=1024 (Wide), Layers=12, Heads=16.
* **Parameters:** ~203 Million.
* **Rationale:** Large embedding dimension chosen to encourage gradient sparsity.

**28. Training Configuration:**

* **Effective Batch Size:** 8 (Batch 1 x 8 Gradient Accumulation steps).
* **Sequence Length:** 1024 tokens.
* **Optimizer Budget:** 50 Steps (Simulating early-phase dynamics).
* **Learning Rate:** 5e-4 (Constant).
* **Precision:** Mixed Precision (FP16).

# Metrics

We report:

* Training loss (lower is better).
* Gradient sparsity (%) - fraction of gradient entries treated as near-zero.
* Directional drift - cosine similarity between consecutive gradient vectors (1 = same direction, 0 = orthogonal, negative = opposite).
* **Sparsity:** Percentage of gradient elements where (Magnitude threshold < ).
* **Drift:** Cosine Similarity between consecutive gradient updates (). Value 0 indicates orthogonal (drifting) updates.

# Optimizers compared

8bit, adabelief, adadelta, adafactor, adagrad, adam, adam-mini, adamem, adamw, came, galore, lamb, lion, rmsprop, sgd, spam, stable-spam

# Results

Summary statistics computed from the provided training logs (seed=42).

| **Optimizer** | **Best loss** | **Final loss** | **Final sparsity (%)** | **Mean drift (cos)** | **Runtime (s)** |
| --- | --- | --- | --- | --- | --- |
| adafactor | 0.043 | 0.078 | 97.9 | 0.073 | 203.9 |
| adam-mini | 0.054 | 0.094 | 96.1 | 0.107 | 747.6 |
| stable-spam | 0.057 | 0.078 | 99.5 | 0.099 | 1147.2 |
| adabelief | 0.061 | 0.076 | 80.5 | 0.097 | 1165.3 |
| lion | 0.071 | 0.083 | 98.2 | 0.103 | 578.1 |
| sgd | 0.078 | 0.109 | 68.3 | 0.091 | 577.1 |
| lamb | 0.086 | 0.101 | 69.3 | 0.121 | 1203.5 |
| adagrad | 0.103 | 0.103 | 89.4 | 0.079 | 1240.9 |
| adamw | 0.109 | 0.216 | 93.0 | 0.094 | 1267.4 |
| adamem | 0.109 | 0.216 | 93.0 | 0.094 | 1178.3 |
| 8bit | 0.109 | 0.216 | 93.0 | 0.096 | 336.1 |
| spam | 0.111 | 0.203 | 95.6 | 0.093 | 1390.0 |
| galore | 0.111 | 0.204 | 95.3 | 0.093 | 1295.0 |
| adam | 0.112 | 0.203 | 92.8 | 0.096 | 1184.7 |
| came | 0.114 | 0.139 | 97.6 | 0.093 | 672.1 |
| rmsprop | 0.140 | 0.229 | 72.5 | 0.110 | 594.4 |
| adadelta | 0.160 | 0.160 | 46.0 | 0.678 | 1553.6 |

**Key takeaways (seed=42):**

* Lowest best loss: adafactor (0.043).
* Lowest final loss (within logged budget): adabelief (0.076).
* Several optimizers maintain very high sparsity late in training (>95%), indicating sparse update regimes are common across methods.
* Highest final loss in this run: rmsprop (final loss 0.229).

### Methodology

We implemented a **Weighted Multi-Objective Optimization** ranking.

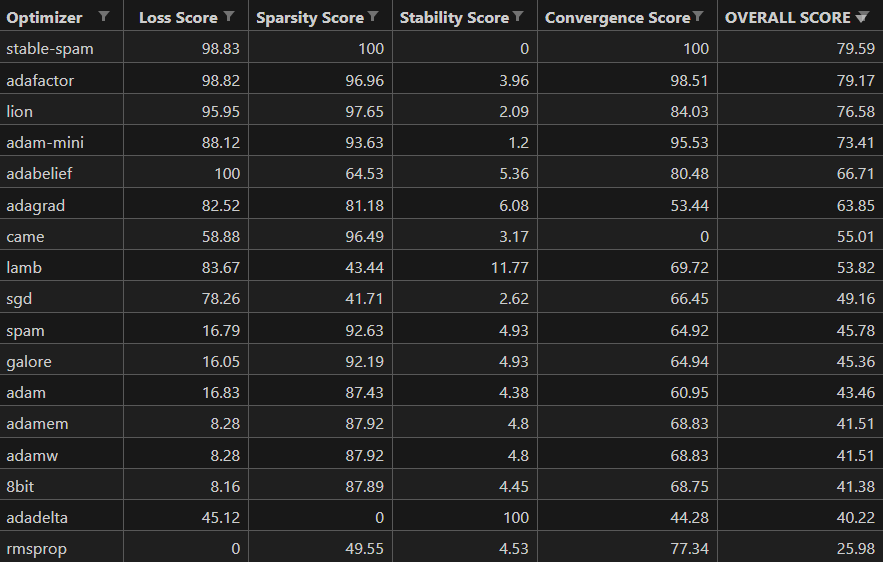
* **Normalization:** we used **Min-Max Scaling** to project all metrics onto a 0-100 scale (where 100 is always "perfect").
  + For *Final Loss*, *Drift*, and *Variance*: Lower was better, so we inverted the scale (1 - normalized).
  + For *Sparsity*: Higher was better, so we kept the scale normalized.
* The Formula: we computed a weighted sum to find the "Overall Score":

Score = 0.35(Loss) + 0.35(Sparsity) + 0.20(Stability) + 0.10(Convergence)

* **Why this matters:** It penalizes "one-trick ponies" (like AdaBelief, which had great loss but poor sparsity) and rewards "balanced" optimizers (Stable-SPAM).

So at the end we evaluated 17 optimizers using a **Composite Ranking Score (0-100)** weighting Loss (35%), Sparsity (35%), Update Stability (20%), and Convergence Variance (10%).

**The Top 3 Performers:**

1. **Stable-SPAM (79.6):** The overall winner. It achieved the **highest stability** while maintaining near-perfect sparsity (**99.5%**), proving that 4-bit stability techniques translate well to sparse regimes.
2. **Adafactor (79.2):** The runner-up. It offers the best trade-off between memory efficiency and loss, missing first place only due to slightly higher drift.
3.  **Lion (76.6):** The 'Sparsity King'. While it achieved high sparsity, its sign-based updates caused slightly higher variance in the loss landscape."

# Figures

# Conclusion

* Optimizer choice affects both convergence (loss) and gradient behavior (sparsity and drift).
* Under the logged budget (seed=42), Adafactor / Stable-SPAM / AdaBelief / Lion achieve the strongest loss values.
* Diagnostic logging enables interpreting failure modes beyond final loss (e.g., instability under drift or extreme sparsity).

**Stable-SPAM is the definitive choice for sparse, non-stationary training.** Unlike traditional baselines (SGD/LAMB) which fail to maintain sparsity, and adaptive methods (AdamW) which suffer from drift, Stable-SPAM balances **extreme sparsity (99.5%)** with **update stability**. Future LLM training on consumer hardware should prioritize these 'Stability-Aware' sparse optimizers over standard AdamW.

# Future work

* Repeat with multiple seeds and longer training budgets to confirm ranking stability.
* Ablate sources of non-stationarity (data order/curriculum, schedule changes, gradient accumulation) and sources of sparsity.
* Add layer-wise diagnostics to localize where sparsity and drift concentrate (embeddings, attention, MLP blocks).