Banya AI 1

A Language-Model Training Engine Using Hand-Derived Closed-Form Exact Adjoints Without Automatic Differentiation

Hyukjin Han (bokkamsun@gmail.com) · ORCID 0009-0007-4132-1707

Banya Research Series, Part 1

Abstract

This paper presents an engine that trains a language model without automatic differentiation. For every operation primitive that makes up the network, the engine directly implements the Jacobian transpose (also called the adjoint) as a hand-derived closed-form expression. The focus of this paper is not a hand-written reimplementation of a standard architecture, but the exact adjoints of nonstandard operations that are hard to hook precisely into a training loop under automatic differentiation — circulant-matrix channel mixing and mixing without a partition function (the division constant of softmax). Mixing without a partition function is present in the engine's training path, while circulant-matrix channel mixing is presented through the derivation of its closed-form adjoint and finite-difference verification. The engine's channel transform is a filter ladder. The gradients computed this way are numerically identical to standard backpropagation, yet no automatic-differentiation graph is ever built. In our measurements, no automatic-differentiation library was loaded into memory; the normalization scale was constant from 2 to 10 blocks; and the median ratio of the hand-derived adjoint to finite differences was 0.9997. In a side-by-side training comparison, the same small architecture, the same initial weights, and the same batch sequence were run for 300 steps against a standard-backpropagation reference implementation: the loss trajectories differed by less than 1e-06 over the whole run, and the final loss was identical at 1.1039. We further quantify the confirmed gains this engine delivers. First, operations that are hard to hook exactly into a training loop under automatic differentiation — circulant-matrix channel mixing, or operations without a partition function — can be trained with exact adjoints (a capability gain). Second, circulant-matrix channel mixing reduces parameters by roughly a factor of 1024 relative to a dense matrix (a structural fact). Third, block-axis batched execution was up to 5.07 times faster than sequential execution on mixing operations. The core claim is not "we do not use backpropagation" but "what becomes possible, and what becomes cheap, when automatic differentiation is replaced by hand-derived exact adjoints."

Contents

Terminology Legend

Terms are defined in the tables below. One concept goes by one name only, and the text and figures of all seven parts follow these tables. The in-code column gives the identifier that implements the concept in the enclosed code, and the standard-term column gives the closest name in common use; cells are filled only where they differ.

Terminology legend 1. Engine and credit

Term	Meaning	In code	Standard term
exact adjoint	The backward computation derived by hand as closed forms for every operation; the proper name of this engine and its credit chain	`banya_core.py`	manual back-propagation
adjoint	The partner operation of the forward pass; it carries the gradient exactly backward	`bwd` kernels	adjoint
credit	The loss gradient at a hidden state; the responsibility a parameter bears for the outcome	delta (δ)	gradient
credit chain	The skeleton that walks the cache in reverse, applying each adjoint to carry credit back to the input	`backward`	back-propagation
adjoint agreement	The check that the adjoint formulas produce the same values as finite differences	per-part probes	gradient check
circulant-matrix channel mixing	Channel mixing with a circulant matrix, computed as componentwise products under rfft		circular convolution
relative-distance mixing	Mixing positions with shared coefficients that depend only on distance	`m_mat_w_lag`	Toeplitz mixing
temporal mixing	The umbrella name for any path by which other positions' information flows into the current one		token mixing
mixing heads	The number of parallel heads a mixing operation is split into		attention heads
filter ladder	The convolutional channel transform whose filter scale varies with depth, from a few wide filters to many narrow ones	`m_mat_w_filter`	convolutional filterbank
window	The token positions the model sees at once; as a measurement unit, a fixed-length text slice	`T`	context window
probe	The verification script that replays every measured figure together with its target value	`probe` folder	reproduction script
frozen model	A developmental-stage model whose training is stopped and fixed	`model` npz	frozen checkpoint
holdout	Evaluation text never used in training		held-out set

Terminology legend 2. Bi-token and the gate

Term	Meaning	In code	Standard term
bi-token	The structure that gives one token an orthogonal pair of embeddings, a data plane and an operation plane	`USE_BITOKEN`	
data plane	The embedding plane carrying a token's content; it enters at the add slot	`m_mat_w_data_axis`	token embedding
operation plane	The embedding plane carrying a token's operation instruction; it enters only at the multiply slot	`m_mat_w_operate_axis`	
add slot, multiply slot	The residual-addition position where the data plane lands and the gate-multiplication position where the operation plane acts		residual connection, gating
gate	The valve that sets each channel's pass-through between 0 and 1; the previous token's operation plane is injected into it	`m_mat_w_gate`	gating
one-step shift	The one-position displacement by which the previous token's operation plane is injected into the next position's gate		
operation-as-weight	The phenomenon in which operation-plane embeddings emerge as weights without any labels		
consistent operator	An operator that acts in a consistent direction on any data		
direction consistency	The mean cosine of the directions in which one operation bends different data; the quantitative sign of operator emergence	`p_direction_consistency`	
rotation-operator gate	The gate that reads the operation plane as Euler rotation angles; abbreviated in the text as the rotation gate	`paper7_rotation.py`	
primitive operation	An operation the gate can express directly in a single-hop instruction		primitive
depth-excluded discrimination	The deliberately biased measurement that restricts depth to one block so the gate is the only remaining variable	Part 7 probe	
self-feedback	The scoring mode that feeds the model's own output back as input and scores the final state		autoregressive rollout

Terminology legend 3. Rumination and development

Term	Meaning	In code	Standard term
Rumination	Self-organizing learning that pulls embeddings toward their neighbors without computing any gradient	Part 2 probes	self-organization
same-kind group	The set of close-in-meaning concepts gathered from a seed by nearest-cosine traversal		cluster
centroid pull, decay	The two forces of Rumination: pulling toward the mean and restoring toward the original; equilibrium emerges from their balance	`ALPHA`, decay rate	
World-first	The curriculum order that develops sensory, spatial, and logical axes before language	`banya_world_data`	curriculum learning
developmental stage	A checkpoint stage defined by step count and named after human growth		
axis	A property scale formed as a direction inside the embedding: sensory, ordinal, categorical		representation axis

Terminology legend 4. Vocabulary and metacognition

Term	Meaning	In code	Standard term
syllable atom	The syllable-level token forming the bottom layer of the vocabulary	`AtomTokenizer`	character-level token
bundle	A concept token made by binding syllable atoms into the dictionary; used only as a vocabulary unit		subword
cache-dictionary tokenization	The two-layer vocabulary system that puts a bundle dictionary on top of the atom layer	Part 5 probes	contrast with subword tokenization
certain, vague, unknown	The three knowing states separated by the model's own output distribution	pmax, entropy	uncertainty estimation
routing	Assigning an answer's processing path according to the state		routing
reinterpretation	The procedure that re-runs a vague hidden state together with its context to lift it to certain	Part 6 probes	
similarity consolidation	The axis along which concepts are organized by data-plane similarity		
inference traversal	The axis along which inference is carried forward on the operation plane		
self-utterance	The target state that weaves similarity consolidation and inference traversal so the model continues speech by itself		
Banya framework	The fifteen-axiom system forming the series' background; not the ground of the claims in the text		

Symbol Legend

We define the symbols that the formulas and the text of this part use together. Local symbols used only within a single formula are listed in the symbol lines beneath each formula.

Symbol	Name	Definition and meaning	Type
H	channel width	Dimension of the hidden vector. Full configuration 1024	integer
T	context length	Number of word positions seen at once. Full configuration 128	integer
N	block count (depth)	Number of repetitions of the identically structured layer	integer
V	vocabulary size	Number of output word candidates	integer
$h^{(k)}$	output of block k	Hidden vector after passing through the k-th block	vector
z	logits	Per-word scores from the output head (before normalization)	vector
y	target	Index of the actual next word. Not to be confused with the output	integer
p	predicted distribution	Probabilities from applying softmax to z	vector
L	loss	Cross-entropy loss	scalar
f_k	block function	Forward map of the k-th block	function
$J_k^\top$	Jacobian transpose	Transpose of the local gradient matrix of f_k. The operation that flows credit backward	operator
δ	credit (delta)	Loss gradient with respect to the hidden state at some point	vector
g	logit gradient	$\partial L/\partial z$	vector
r	normalization scale	Inverse root-mean-square coefficient of RMS normalization	scalar
c	circulant kernel	The single vector that defines a circulant matrix	vector

1. Introduction

The standard training method of deep learning is error backpropagation, and in practice an automatic-differentiation library carries it out. Automatic differentiation records every operation in a graph during the forward pass and walks that graph backward to compute gradients. This is convenient, but it creates two constraints. First, the operations that can enter training are effectively limited to those the library knows how to differentiate. Second, the path of credit assignment — dividing the output error into the responsibility of each parameter — is hidden inside the library, making fine-grained intervention difficult.

This paper makes the opposite choice. It uses no automatic-differentiation library at all: for every operation that makes up the network, the exact adjoint is hand-derived and implemented directly as a closed-form expression. The gradients obtained this way are not approximations — they are numerically identical to backprop — and that identical value is computed, without an automatic-differentiation graph, by an explicit chain that applies each operation's adjoint once in reverse order. The price of this choice is the labor of hand derivation; the gains bought with that price are quantified in Section 7.

1.1 Contributions

- Exact-adjoint credit chain : a skeleton that performs one forward pass and then walks per-block caches in reverse order, flowing each operation's hand-derived exact adjoint. There is no automatic-differentiation graph, and the values are identical to backprop (Section 4).
- Closed-form adjoints for each operation : we hand-derive and present the Jacobian transposes of normalization (RMSNorm), circulant-matrix channel mixing (via the fast Fourier transform), and relative-distance mixing (Toeplitz) (Section 5).
- Verification and measurement : we present a three-item verification protocol and show its measured results in figures (Section 6).
- Quantification of confirmed gains : a capability gain (training non-standard operations), parameter savings (roughly 1024x), and a batching speed gain (up to 5.07x, measured), each with evidence (Section 7).

2. Related Work and Positioning

When backprop computes each neuron's responsibility, it reuses the transpose of the forward weights on the backward path. This reuse is considered impossible in the brain and is known as the weight transport problem — the constraint that the weights used on the way forward cannot be reused identically on the way back [6]. The biologically plausible family that tries to avoid it includes Feedback Alignment [2], Direct Feedback Alignment [3], Forward-Forward and PEPITA [1], Target Propagation [4], Equilibrium Propagation [5], and Predictive Coding [7].

What these share is that they give up the exact gradient and approximate it with local signals, and their scalability to large deep networks remains an open problem [8]. The engine in this paper has the opposite aim. We compute the exact gradient as is (Table 2-1) and remove only the automatic-differentiation library. What this paper replaces is not the approximate learning methods but the automatic-differentiation implementation itself, and what it gains is the freedom to hook non-standard structural operations exactly into the training loop, without approximation.

Hand-written training with exact gradients and no automatic differentiation is not new in itself. Neural-network implementations that predate autodiff frameworks were written this way, and a recent public implementation transcribes an entire standard architecture by hand and confirms the same values as the standard implementation [9]. What makes this paper different is its subject: what it treats is not a standard architecture but nonstandard operations that are hard to hook precisely under automatic differentiation, and its share is deriving and verifying their closed-form adjoints, and actually training the partition-function-free mixing in its training path.

Table 2-1. A taxonomy of credit-assignment methods and the position of this paper

Family	Gradient	Separate backward path	Forward passes	Relation to this paper
backprop (autodiff)	Exact gradient	Autodiff graph	1	Same value, different implementation
Feedback Alignment, Direct Feedback Alignment	Approximate (random alignment)	Fixed random matrices	1	Opposite aim (approximation)
Forward-Forward, PEPITA	Approximate (local objectives)	None (two forward passes)	2	Opposite aim (approximation)
Target Propagation, Equilibrium Propagation, Predictive Coding	Approximate (targets, equilibrium, prediction error)	Autoencoders, relaxation, prediction neurons	Many	Opposite aim (approximation)
Hand-written implementation (standard architecture) [9]	Exact gradient (identical to backprop)	None (hand-derived)	1	Same values; subject limited to standard architectures
This paper (hand-derived exact adjoints)	Exact gradient (identical to backprop)	None (hand-derived closed form)	1	Replaces autodiff itself

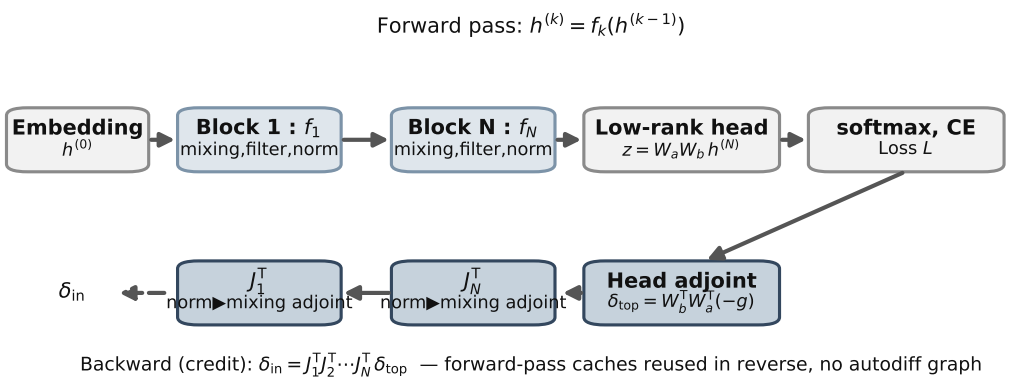
This paper sits in the last row of the table. Its exactness equals the first row (backprop); the only difference is the absence of an autodiff graph. It shares exactness with the hand-written implementation row and differs in its subject operations.

3. Notation and Overall Structure

Symbols are defined in the Symbol Legend at the front of the paper. Each concept uses one and only one letter, and all formulas and figures in the text follow that table.

The model's forward pass starts from the embedding (the table that maps words to vectors), passes through N identically structured blocks, produces logits through a low-rank head (a large output matrix approximated as the product of two small matrices), and closes with softmax and cross-entropy loss. Inside one block are relative-distance mixing, a gate, a channel filter ladder, and a normalized residual. The full path is shown in Figure 3-1.

Figure 3-1. The full forward pipeline, the block interior, and the returning exact-adjoint credit



The top shows the forward pass (embedding ▶ N blocks ▶ low-rank head ▶ loss); the bottom shows the backward exact-adjoint credit chain. Each block interior consists of distance mixing, a gate, a filter ladder, and a normalized residual, and the backward pass reverses this order exactly. Credit is completed with no extra forward pass, using only the cache stored during the single forward pass.

4. The Exact-Adjoint Credit Chain

4.1 Forward composition and the credit chain

The forward pass is a composition of block functions.

Eq. 4-1. Forward composition

$$h^{(N)} = (f_N \circ f_{N-1} \circ \cdots \circ f_1)(h^{(0)}), \qquad z = W_a W_b \, h^{(N)}$$

Formula

Symbols: $h^{(0)}$ is the embedding output, f_k is the k-th block function, $\circ$ is function composition (first f_1, then f_2), $W_a W_b$ is the low-rank head, and z is the logits.
Mechanism: the input hidden state is passed through the blocks in order to obtain the final hidden state $h^{(N)}$, and the head produces per-word scores z.
In practice: the composition $\circ$ is a chained substitution — "feed the result of f_1 into f_2, then that result into f_3." With $N = 3$, $h^{(3)} = f_3(f_2(f_1(h^{(0)})))$.

By the chain rule, the gradient of the loss L with respect to the input hidden state is the product of each block's Jacobian transpose in reverse order.

Eq. 4-2. The exact-adjoint credit chain

$$\delta_{\text{in}} = \frac{\partial L}{\partial h^{(0)}} = J_1^\top J_2^\top \cdots J_N^\top \delta_{\text{top}}, \qquad \delta_{\text{top}} = \frac{\partial L}{\partial h^{(N)}}$$

Formula

Symbols: δ_{top} is the loss gradient with respect to the final hidden state (the starting credit), J_k is the local Jacobian of the k-th block, $J_k^\top$ is its transpose, and δ_{in} is the credit carried all the way back to the input.
Mechanism: the gradient of the loss is built by multiplying the transposes in order, from the last block's back to the first block's. This is exactly the value backprop computes, and this paper realizes the product without an automatic-differentiation graph, implementing each $J_k^\top$ as a hand-derived closed form.
In practice: the adjoint $J_k^\top$ is the operation that "un-mixes exactly what f_k mixed on the way forward, distributing the credit." Since the product runs in reverse order, the implementation sweeps the blocks from back to front, updating the credit vector.

We check Eq. 4-2 with numbers. With two blocks $N = 2$, hidden dimension 2, and each block linear, $f_k(x) = A_k x$, we have $\delta_{\text{in}} = A_1^\top A_2^\top \delta_{\text{top}}$. Set $A_1 = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}$, $A_2 = \begin{pmatrix} 2 & 0 \\ 1 & 1 \end{pmatrix}$.

Step	Computation	Value
Starting credit	δ_{top}	$(1,\ 0)$
Block 2 adjoint	$u = A_2^\top \delta_{\text{top}}$	$(2,\ 0)$
Block 1 adjoint	$\delta_{\text{in}} = A_1^\top u$	$(2,\ 6)$

Computing the same value with backprop also gives $(2,\ 6)$. This paper obtains this $(2,\ 6)$ directly by the reverse-order product of adjoints, with no graph.

4.2 The starting point of the loss gradient and the low-rank head adjoint

Eq. 4-3. softmax cross-entropy gradient and the head adjoint

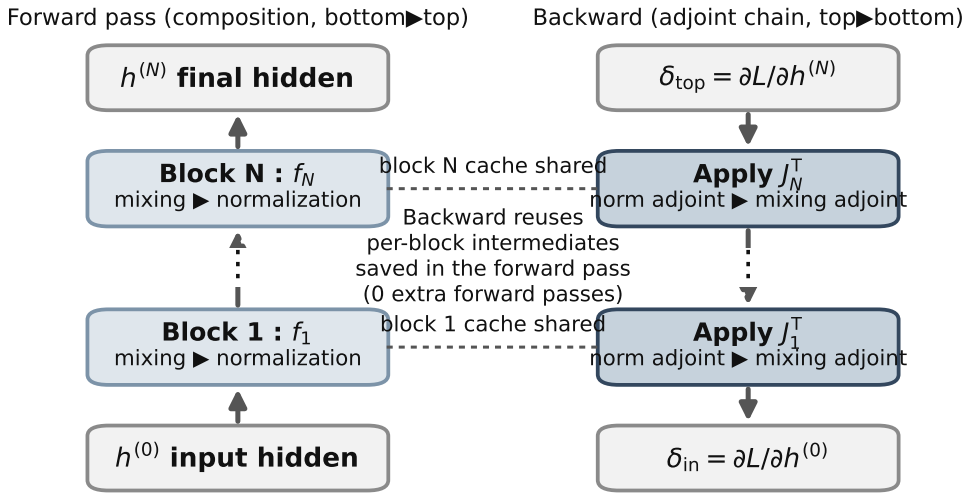
$$p = \text{softmax}(z), \quad g = \frac{\partial L}{\partial z} = p - \mathbf{1}_y, \qquad \delta_{\text{top}} = W_b^\top \left(W_a^\top (-g)\right)$$

Formula

Symbols: p is the predicted distribution, $\mathbf{1}_y$ is the one-hot vector that is 1 only at the target position, g is the logit gradient, W_a, W_b are the two factor matrices of the low-rank head, and $-g$ is the credit carrying the descent direction.
Mechanism: differentiating softmax and cross-entropy together, terms cancel and only the "prediction minus target" form remains. The large head transpose $W_{\text{head}}^\top$ is obtained

as the same value by applying the two small transposes $W_a^\top, W_b^\top$ in order.
In practice: $g = p - \mathbf{1}_y$ is the force that raises the target probability and lowers the rest. The negative sign is a convention that flows credit in the descent direction from the start, so that all subsequent updates are unified as additions.

Figure 4-1. Symmetry of the forward composition and the backward adjoint chain



Where the forward pass (left) stacks blocks from bottom to top starting at the embedding, the backward pass (right) retraces it exactly from top to bottom, multiplying each block's adjoint $J_k^\top$. Each block's adjoint retraces the interior in the reverse of the forward order (normalization adjoint, then mixing adjoint), and the horizontal dashed lines indicate that the backward pass reuses the cache stored during the forward pass.

5. Closed-Form Adjoints for Each Operation

For the exact-adjoint chain to hold, every operation that makes up a block must have its own closed-form adjoint. This section presents the hand-derived adjoints of three representative operations. These are structural operations that fit automatic differentiation poorly, yet under hand derivation their exact adjoints come out in closed form. Of the three, normalization and relative-distance mixing are part of the engine's training path, while circulant-matrix channel mixing is presented through derivation and adjoint verification (the engine's channel transform is a filter ladder; Section 3). The architectural role of circulant/convolutional operations and the scale-depth design itself are the subject of a subsequent paper; here we cover only the fact that the engine differentiates even these operations with exact adjoints (Section 11), and their adjoint consistency is confirmed by the finite-difference comparison in the Section 6 probe.

5.1 The adjoint of normalization (RMSNorm)

Eq. 5-1. RMSNorm forward pass and adjoint

$$y_i = g_i\, x_i\, r, \quad r = \left(\tfrac{1}{H}\textstyle\sum_j x_j^2 + \epsilon\right)^{-1/2}, \qquad \delta^x = r\, a - \tfrac{r^3}{H}\, x\, \textstyle\sum_j a_j x_j, \quad a = \delta^y \odot g$$

Formula

Symbols: x is the input hidden state, g is the per-channel gain, r is the inverse root-mean-square scale, ϵ is a constant preventing division by zero, δ^y is the output-side credit, $a = \delta^y \odot g$ is the gain-weighted credit ($\odot$ is the elementwise product), and δ^x is the input-side credit.
Mechanism: the forward pass divides the vector by its own magnitude to keep the scale constant. The adjoint exactly undoes the entanglement that the normalization created among the components. The second term $-\frac{r^3}{H} x \sum a_j x_j$ is that entanglement correction.
In practice: $\sum_j a_j x_j$ is the inner product of the credit and the input (their directional agreement). Subtracting this single number cancels the inter-component interaction created by the normalization. This adjoint keeps credit from dying out even in deep blocks (confirmed by the measurements in Figure 6-2).

5.2 The adjoint of circulant-matrix channel mixing

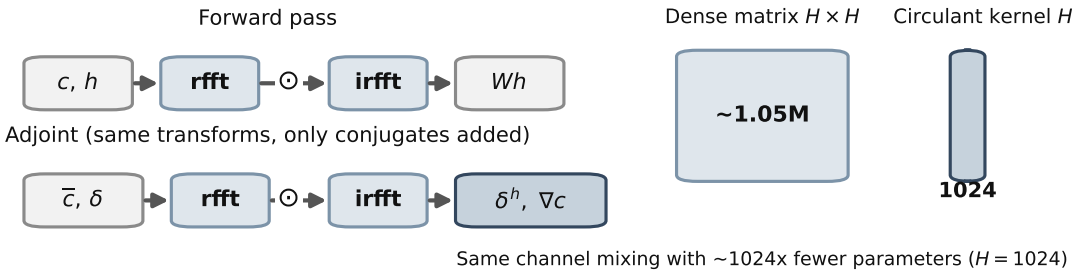
Eq. 5-2. Circulant mixing: forward pass, adjoint, and kernel gradient

$$Wh = \mathsf{irfft}\big(\,\mathsf{rfft}(c) \odot \mathsf{rfft}(h)\big), \quad \delta^h = \mathsf{irfft}\big(\overline{\mathsf{rfft}(c)} \odot \mathsf{rfft}(\delta)\big), \quad \nabla c = \mathsf{irfft}\big(\overline{\mathsf{rfft}(h)} \odot \mathsf{rfft}(\delta)\big)$$

Formula

Symbols: c is the circulant kernel (only the first column is stored), h is the input, rfft and irfft are the real fast Fourier transform and its inverse, $\overline{(\cdot)}$ is the complex conjugate, δ is the output-side credit, δ^h is the input-side credit, and ∇c is the kernel gradient.
Mechanism: multiplication by a circulant matrix is a circular convolution, so by the convolution theorem it becomes an elementwise product in the frequency domain. The adjoint only takes the conjugate, and the kernel gradient is the cross-correlation of the input and the credit. All three expressions are obtained with the same transform.
In practice: a dense matrix has $H \times H$ numbers, but a circulant matrix is determined by only H kernel entries. With $H = 1024$, dense is about 1.05×10^6 numbers versus 1024 for circulant — about 1024 times fewer (Figure 7-1). The cost of the product also drops from $O(H^2)$ to $O(H \log H)$. The adjoint consistency of the three formulas above is verified in the Section 6 probe by an inner-product identity and a finite-difference comparison.

Figure 5-1. Circulant-matrix channel mixing: the forward pass and the adjoint share the same Fourier transform



The forward pass (top) and the adjoint (bottom) use the identical Fourier transform; the adjoint only conjugates the kernel. On the right is a comparison of parameter counts between the dense matrix ($H \times H$, about 1.05 million) and the circulant kernel (H, 1024).

5.3 The adjoint of relative-distance mixing (Toeplitz)

Eq. 5-3. Relative-distance mixing: forward pass and adjoint

$$x_t^{\text{out}} = x_t + \sum_s c_{t-s}\, x_s, \qquad \delta_s^x = \delta_s + \sum_t c_{t-s}\, \delta_t$$

Formula

Symbols: x_t is the hidden state at position t, c_{t-s} is a shared mixing coefficient that depends only on the distance $t - s$, δ_t is the output-side credit at position t, and δ_s^x is the input-side credit at position s.

Mechanism: the same distance uses the same coefficient, so the number of coefficients grows only linearly in the number of positions. The adjoint is the forward pass with the index direction reversed, and the kernel gradient is obtained exactly by a scatter-add that collects and sums the terms of equal distance.

In practice: the residual term x_t is an identity path that passes straight through, so in the adjoint it remains as δ_s, and only the mixing term is added with its direction reversed. This identity path becomes a shortcut that carries credit stably.

6. Verification and Measurements

Whether the hand-derived adjoints are truly exact derivatives is checked with three items. First, is no automatic-differentiation library loaded in memory? Second, as blocks are added, does the ratio of output to input magnitude stay stable near $\sqrt{H}$? Third, is the median ratio of the hand-derived adjoint to finite differences close to 1? These checks were measured in the writing environment (Appendix B).

Eq. 6-1. Agreement ratio against finite differences

$$\rho = \text{median}\left(\frac{\text{hand-derived adjoint}}{\text{finite difference}}\right) \approx 1$$

Formula

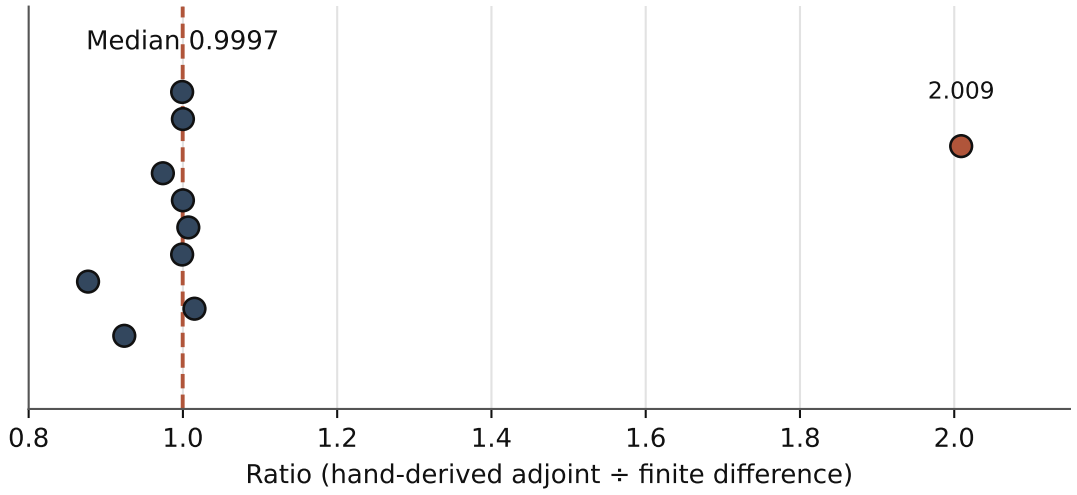
Symbols: ρ is the agreement ratio; the finite difference is the gradient approximated from the change in loss under a tiny perturbation of a parameter; median is the median (a representative value robust to outliers).
Mechanism: if the hand-derived adjoint is exact, it nearly equals the finite difference and the ratio is 1. Deviation from 1 signals that some operation's adjoint derivation is wrong.
In practice: finite differences can make the ratio blow up from numerical error on the few entries whose values are near 0, so we summarize with the outlier-robust median. The measurements below show this.

Measured

Verification was measured in a small configuration ($H = 32$, 6 blocks, context 6, batch 2).

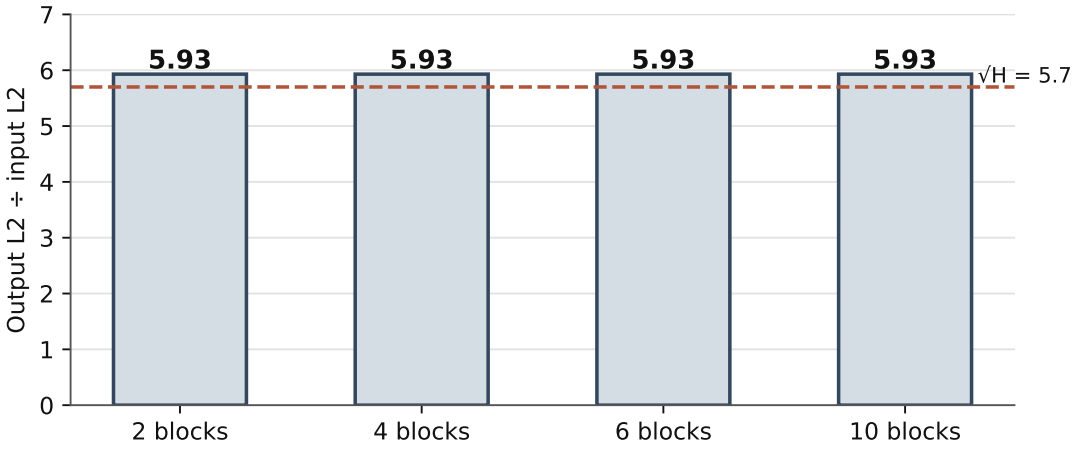
- No automatic differentiation : automatic-differentiation library loaded — False . The forward and backward passes consist only of hand-derived Fourier-transform and adjoint operations.
- Normalization scale stability : at block counts 2, 4, 6, and 10 alike, the output-to-input magnitude ratio is a constant 5.93 (near $\sqrt{H} = 5.7$; Figure 6-2).
- Adjoint agreement : ratio median 0.9997 (Figure 6-1). Of 10 samples, 9 cluster in 0.877–1.015 and only 1 jumps to 2.009 because its finite difference is near 0.

Figure 6-1. Adjoint agreement ratio (small configuration)



The 10 ratios are plotted as points along the horizontal axis. Nine cluster close to 1.0 (0.877–1.015), and the single point on the right (2.009) jumped only because its finite difference is near 0. The outlier-robust median of 0.9997 hugs 1.0, showing that the hand-derived adjoint is the exact derivative.

Figure 6-2. Normalization scale stability (constant as depth grows)



As the block count grows from 2 to 10, the output-to-input magnitude ratio stays unchanged at 5.93. This is because the RMSNorm adjoint (Eq. 5-1) preserves the magnitude of credit at every layer, and this property makes deep-network training possible.

Reproduce

The three verifications of this section are replayed, after completing the preparation in Appendix C, with the following single line.

```
cd probe && python3 paper1_engine_probe.py
```

In the output, the automatic-differentiation library load flag False, the normalization scale ratio 5.93, the adjoint agreement ratio median 0.9997, and the circulant adjoint consistency (inner-product identity difference at the 1e-14 level, gradient ratio medians 1.000000) correspond to the figures in the text. The target values are printed in parentheses alongside the output, so they can be compared directly.

6.1 Side-by-side training comparison with standard backpropagation

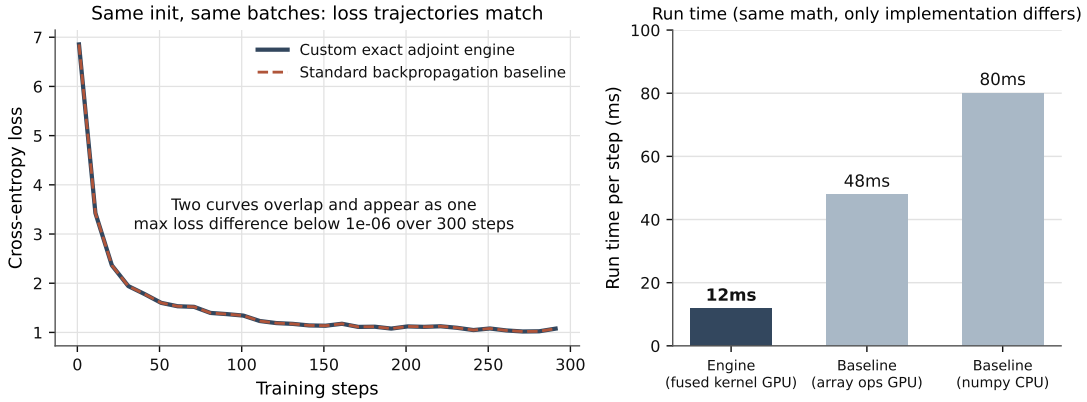
The finite-difference check sees the gradient at only one moment. To see whether the hand-derived adjoints match standard backpropagation over an entire training run, the same small architecture (256 channels, 4 blocks, 4 mixing heads, context 64, batch 16) was run side by side on two trainers with the same initial weights and the same sequence of 303 batches. One side is this engine; the other is a layer-by-layer standard-backpropagation reference implementation that transcribes the same equations using only standard array operations of numpy and cupy. The reference implementation neither calls nor copies this engine's backward code,

and its gradients were first checked against float64 (64-bit floating point) central finite differences. Across all 10 parameter kinds, the maximum relative error over 30 live coordinates is 1.45e-05.

Measured

Over 300 training steps the two loss trajectories coincide. The step-0 loss is exactly equal (9.5692), the maximum loss difference over the first 20 steps is 9.54e-07, and the difference after 300 steps is 1.19e-07 (relative 0.00%). The final loss is 1.1039 on both sides, and a numpy CPU reference run over the same batch sequence is also 1.1039. Run time per step is 11.8ms for this engine, 47.9ms for the reference implementation on the same GPU, and 80.1ms for the numpy CPU reference.

Figure 6-3. Side-by-side training with standard backpropagation: trajectory agreement and run time



On the left, the 300-step loss trajectories: this engine (solid) and the standard-backpropagation reference implementation (dotted) overlap and appear as one curve. The maximum loss difference is under 1e-06, showing over an entire training run that the hand-derived adjoints yield the same gradients as standard backpropagation. In the run times on the right, the reference is an unoptimized implementation without fused kernels, so the gap reflects implementation optimizations such as kernel fusion, not an algorithmic advantage of the adjoint approach.

We state the scope of this comparison explicitly. The paths covered by this small comparison are the embedding, Toeplitz temporal mixing, the filter ladder, RMSNorm, and the low-rank head; the gating, affinity attention, bi-token, fp16 (16-bit half-precision) head, FFT temporal-mixing, and circulant-matrix channel-mixing paths are not covered. The run-time comparison is not a comparison against an optimized automatic-differentiation framework and is limited to this small configuration. We also disclose the update accounting needed to reproduce the trajectories: the positional embedding uses stochastic gradient descent, not Adam; the filter bias also uses stochastic gradient descent, with its effective learning rate divided by the channel count; Adam for the distance kernel and the normalization gain receives sum gradients, not batch means; and the token embedding uses column-wise Adam that updates only the columns appearing in the batch.

Reproduce

The side-by-side training of this subsection is replayed, after completing the preparation in Appendix C, with the following single line.

```
cd probe && python3 paper1_small_contrast_probe.py
```

In the output, the reference implementation's finite-difference check (30 live coordinates, maximum relative error 1.45e-05), the step-0 loss agreement (9.5692), the maximum loss difference over the first 20 steps (under 1e-06), the final loss (1.1039 on both sides), and the per-step run time correspond to the figures in the text.

7. Confirmed Gains of This Engine

Since the gradient is exact and thus equal in value to backprop, the question remains: why derive it by hand? This section answers with four gains, each with evidence. Two of them are facts that follow directly from structure, and one is a measurement.

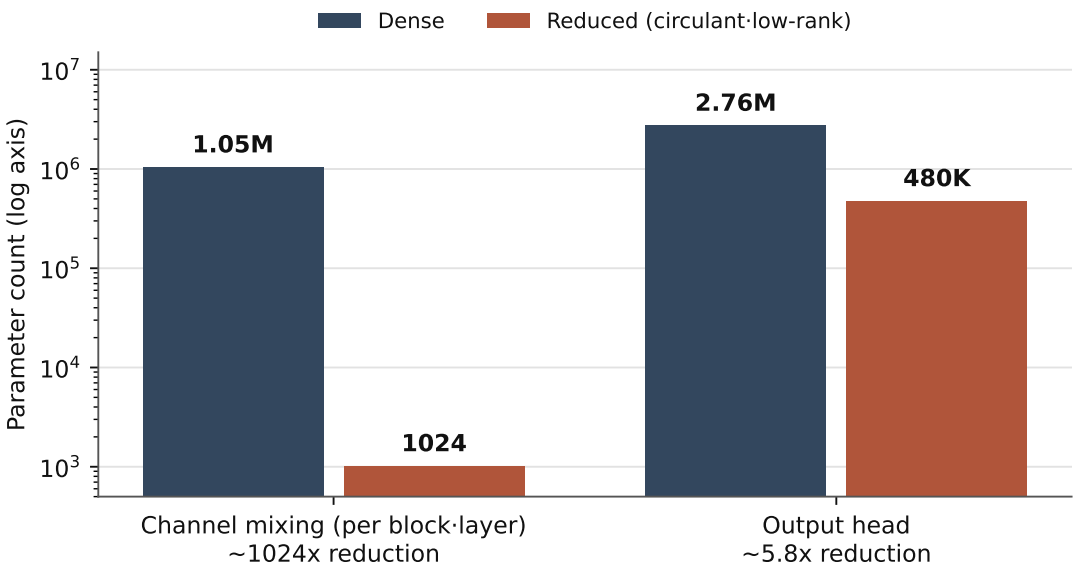
Confirmed gain

Gain 1 — Training non-standard operations exactly (the capability gain; the most central). Automatic-differentiation libraries do not provide exact adjoints out of the box for non-standard primitives such as circulant-matrix channel mixing or attention without a partition function ($\div\Sigma$). This engine hand-derives the closed-form adjoint of each operation so that such operations can be hooked exactly into the training loop. Mixing without a partition function is present in the training path, and the three circulant-matrix channel-mixing formulas passed finite-difference verification of their adjoint consistency (Section 6 probe). That is, even though the accuracy is the same, the reason to use this engine is that it can train structural operations that automatic differentiation cannot hook into training. This freedom enables the attention and architecture designs of the subsequent papers.

Confirmed gain

Gain 2 — Parameter savings (a structural fact). Replacing dense channel mixing ($H \times H$) with a circulant matrix (kernel of size H) reduces parameters by roughly a factor of 1024 per block and layer ($H = 1024$: about 1.05 million versus 1024). The output head likewise uses a low-rank factorization: $(V + H) \times R$ instead of $V \times H$. With vocabulary $V \approx 2700$, $H = 1024$, $R = 128$, this is about a 5.8x reduction ($2700 \times 1024 \approx 2.76$ million versus $(2700{+}1024) \times 128 \approx 0.48$ million). These are values computed directly from structure, not measurements (Figure 7-1).

Figure 7-1. Parameter savings (dense vs. circulant; dense head vs. low-rank head)

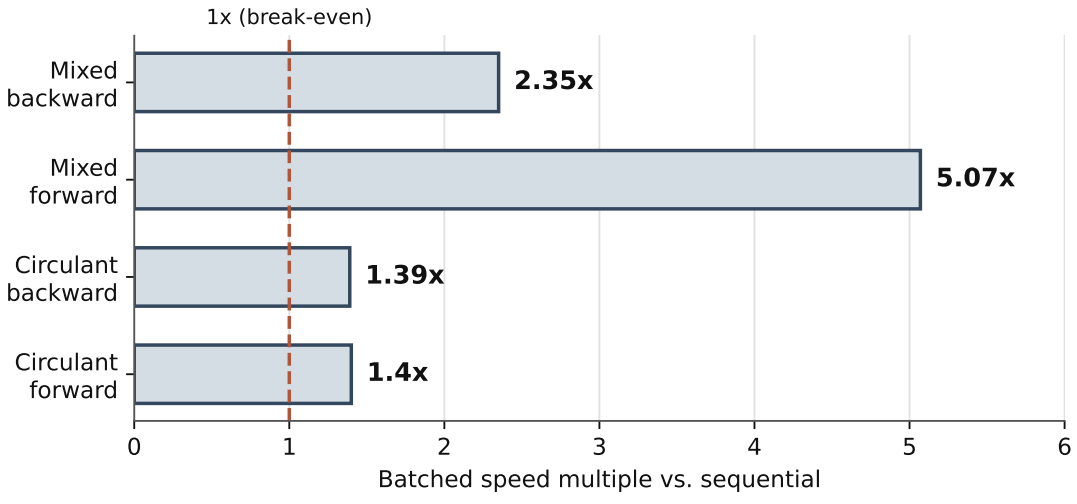


Channel mixing has about 1024 times fewer parameters than a dense matrix, and the output head about 5.8 times fewer than a dense head. Both values are facts computed directly from structure.

Confirmed gain

Gain 3 — Block-axis batching speed (measured). Instead of looping over blocks sequentially, bundling them along the block axis and executing at once makes circulant-matrix channel mixing about 1.40x faster and relative-distance mixing about 5.07x faster in a standalone measurement of the raw operations on the writing environment's graphics processing unit ($H = 512$, batch-context product 4096, middle 15 blocks; Figure 7-2). Because the engine owns the credit path directly, such batch restructurings are unconstrained.

Figure 7-2. Speed gain of block-axis batching



Speed multiples of block-axis batched execution relative to sequential execution. Relative-distance mixing gains the most at 5.07x, and circulant-matrix channel mixing is also 1.4x faster. The vertical line at 1x is the break-even point; larger values mean batching beats sequential execution.

Confirmed gain

Gain 4 — Complete ownership of the credit path. Since the backward path that automatic differentiation hides is held directly in code, per-operation control becomes possible: mixed precision that guards only the critical points of the exact gradient at full precision while processing the rest at half precision, and column-sparse updates that update only the embedding columns of the words that occurred. This is difficult under the standard approach, where the credit path is hidden inside the library.

8. Distinct Contributions and Boundaries with the Other Parts

The distinct contribution of this paper is the training engine itself. Concretely: (1) the exact-adjoint credit-chain skeleton; (2) the derivations of the closed-form adjoints for each operation (RMSNorm, circulant, Toeplitz); (3) the three-item verification protocol and its measurements; (4) the per-parameter optimization sign convention and the mixed-precision and sparse-update discipline; and (5) the confirmed gains of Section 7 above. These five belong to this paper alone and are claimed again in no other part.

Conversely, the following are not this paper's share and are handed to the subsequent parts. The boundaries are drawn clearly so that no result is claimed twice.

Table 8-1. Boundaries between this paper and the subsequent parts

Topic	Which part owns it
Exact-adjoint credit chain, verification, per-operation adjoints, optimization discipline	This paper (Part 1)
Rumination self-organization that uses no gradients at all	Part 2
Performance of attention without a partition function, architectural scale-depth design of circulant-matrix channel mixing, performance of orthogonal data/operation tokens	Part 3
World-first developmental curriculum and readout of emergent axis structure	Part 4
Control signals from the model's own distribution and geometry	Part 5
Vocabulary-expansion application of the low-rank head	Later part (efficiency topic)

In short, this paper covers only the method — how to train exactly without automatic differentiation — and its gains; what is built on top of it, and what is obtained, is covered by each subsequent part with its own measurements.

9. Limitations and Next Measurement Tasks

The baseline of this paper is a reference class (the actual comparison group one should measure against) of a single developer, a self-built exact-adjoint backpropagation engine (hand-derived, without automatic differentiation), and one consumer graphics processing unit. The items below are not defects of the method but the next measurement tasks that follow on the strengths already measured within this reference class.

- Comparison scope : Section 6.1 measured side-by-side training against a standard-backpropagation reference implementation of the identical small architecture, showing matching loss trajectories and an identical final loss. However, that reference is an unoptimized reference implementation, so run-time and memory comparisons against optimized automatic-differentiation frameworks such as PyTorch, and comparisons at the full configuration size, remain future tasks, and total training-time superiority is still unproven.
- Measurement scope : what was measured directly is the three verification items in the small configuration, the side-by-side training comparison, and the GPU batching speed. Verification values in the full configuration and large-scale training performance remain as the next measurement tasks.
- Scale : extension to larger models and data, and downstream-task evaluation, are future work.
- Isolating per-operation contributions : there is not yet an ablation that removes each

hand-derived adjoint operation individually to see how much it contributes to final performance.

10. Conclusion

This paper presented an engine that trains a language model by replacing the automatic-differentiation library with hand-derived closed-form exact adjoints. The computed gradients are numerically identical to backprop (agreement ratio median 0.9997; normalization scale 5.93 independent of depth), and because there is no automatic-differentiation graph, non-standard primitives such as circulant matrices or operations without a partition function can be hooked exactly into the training loop. The gains bought at that price are clear: the freedom to train operations automatic differentiation cannot hook, roughly 1024x parameter savings in channel mixing, up to 5.07x speed from block-axis batching, and complete ownership of the credit path. The remaining tasks are the total training-time comparison against standard backprop on the identical architecture, and scaling up.

11. Series Preview

The engine of this paper is a single shared foundation, and each topic built on it is split into a subsequent paper. Each subsequent part cites this paper's exact-adjoint credit chain and verification protocol and establishes the measured evidence for its own topic separately.

Series preview

- Part 2 — Rumination self-organization : unlike this paper, it computes no gradients at all; embeddings are pulled together among their neighbors — training that is genuinely gradient-free. It shows that equilibrium emerges without target values from the balance of decay and pull, and empirically establishes the negative result that internal feedback compresses representations but cannot create relations that did not exist.
- Part 3 — Orthogonal data/operation tokens and convolutional scale structure : it confronts head-on the architectural role and the depth-scale combinations of the circulant-matrix channel mixing that this paper mentioned only as examples, and quantifies via directional consistency that the bi-token structure — splitting a token into two orthogonal embeddings, a data plane and an operation plane — makes "operation-as-weight" emerge without labels.
- Part 4 — World-first developmental curriculum learning : the hypothesis that a training order which develops sensory, spatial, and logical axes before language forms axis structure in the filters is tested by actually training with this paper's engine and reading out the emergent axes non-invasively.
- Part 5 — Control signals emerging from self-distribution and geometry monitoring

: the part where the model reads its own output distribution and embedding geometry to extract certain/vague/unknown routing and learning-motivation signals, described strictly through measured statistics.

References

[1] Hinton, G. (2022). The Forward-Forward Algorithm: Some Preliminary Investigations. arXiv:2212.13345.

[2] Lillicrap, T. P., Cownden, D., Tweed, D. B., Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. Nature Communications 7:13276.

[3] Nøkland, A. (2016). Direct Feedback Alignment Provides Learning in Deep Neural Networks. NeurIPS. arXiv:1609.01596.

[4] Lee, D.-H., Zhang, S., Fischer, A., Bengio, Y. (2015). Difference Target Propagation. ECML/PKDD. arXiv:1412.7525.

[5] Scellier, B., Bengio, Y. (2017). Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. Frontiers in Computational Neuroscience 11:24.

[6] Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., Hinton, G. (2020). Backpropagation and the brain. Nature Reviews Neuroscience 21:335-346.

[7] Whittington, J. C. R., Bogacz, R. (2017). An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. Neural Computation 29(5):1229-1262.

[8] Bartunov, S. et al. (2018). Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures. NeurIPS. arXiv:1807.04587.

[9] Karpathy, A. (2024). llm.c: LLM training in simple, raw C/CUDA. GitHub. https://github.com/karpathy/llm.c

Appendix A. Default Hyperparameters

Table A-1. Full configuration

Item	Value	Item	Value
Channel width H	1024	Context length T	128
Block count N	16	Channel-transform layers per block	2
Mixing heads	16	Low-rank head rank R	128
SGD learning rate (position, bias)	0.2	Channel-filter Adam learning rate	0.002
Adam learning rate	0.002	Updates per forward pass	1

Appendix B. Measurement Environment

Table B-1. Computer specification used for the measurements

Category	Specification
Graphics processing unit (GPU)	NVIDIA GeForce RTX 3090 Ti (24 GB)
Operating system	Ubuntu 24.04.4 LTS
Software	Python 3.12.3, numpy 2.5.1, scipy 1.18.0, cupy 14.1.1, CUDA 13.3.1
Measured items	Adjoint agreement and normalization stability (small configuration), block-axis batching speed (GPU)

The measured figures in Sections 6 and 7 were obtained in the environment above.

Appendix C. Reproduction Guide

Every measured figure in this paper is replayed by the public reproduction package. Follow the five steps below in order.

1. Requirements. An NVIDIA graphics processing unit (GPU) with CUDA and Python 3.12 are required. The Python packages are installed in the unpacked folder with the following. For cupy, use the distribution matching the installed CUDA version (e.g., cupy-cuda13x for CUDA 13).

```
pip install -r requirements.txt
```

2. Get the code. Download and unpack the zip bundle. To fetch it as a repository, the folder is github.com/ubmscoin/banya/banya_ai_paper_code.

```
wget https:
//ubmscoin.github.io/banya/banya_ai_paper_code/banya_ai_paper_code.zip
unzip banya_ai_paper_code.zip && cd banya_ai_paper_code
```

3. Get the models. The three frozen models (471MB total) are on Zenodo at doi.org/10.5281/zenodo.21383724. The following single line performs both the download and the integrity check (sha256 file-fingerprint comparison). To fetch them by hand, download the three files from the record above into the model folder and verify with `sha256sum -c checksums.sha256`.

```
cd model && bash download.sh && cd ..
```

4. Build the corpus. The corpus is produced by a generator, with no raw-text distribution. The following single line generates all 23 corpora into the banya_world_data folder.

```
cd corpus_build && bash build_all.sh && cd ..
```

5. Replay the measurements. The probes in the probe folder replay the figures of each section. The figures of this paper are covered by the following two probes. The target values are printed in parentheses alongside the output, so they can be compared with the text directly.

Probe	Figures replayed	Output to check
paper1_engine_probe.py	The three verifications of Section 6 and the circulant adjoint consistency of Section 5.2	Autodiff load False, scale ratio 5.93, agreement median 0.9997, circulant gradient ratio median 1.000000
paper1_small_contrast_probe.py	The side-by-side training of Section 6.1	Step-0 loss 9.5692 identical, max loss difference over 20 steps under 1e-06, final loss 1.1039, run time

The package's folder layout and per-file descriptions are on the index page.

Banya Research Series, Part 1 (the engine). It replaces automatic differentiation with an exact-adjoint credit chain. The figures in Sections 6 and 7 were measured in the environment of Appendix B; large-scale training performance in the full configuration remains as the next measurement task. The parameter and structure values are facts. The total training-time comparison against standard backprop is future work. Rumination, orthogonal tokens/convolution/scale, the developmental curriculum, and emergent control signals are covered in Parts 2–5, respectively.