Deep Delta Learning

Yifan Zhang[1] **Yifeng Liu**[2] **Mengdi Wang**[1] **Quanquan Gu**[2]
[1]Princeton University [2]University of California, Los Angeles
yifzhang@princeton.edu

Abstract

Transformer residual streams evolve by additive accumulation: each layer appends a feature update to a shared hidden state, but has no direct mechanism for replacing content that has become obsolete or conflicting. We introduce **Deep Delta Learning (DDL)**, a residual update rule that preserves the identity path while giving every layer the ability to selectively rewrite residual content. DDL reads the current state along a learned direction, compares it with a learned target value, and writes back a gated correction along the same direction. When the gate is closed, the update reduces to the identity; when the gate is fully open, the selected component is overwritten, yielding a depth-wise delta-rule generalization of standard residual addition. We integrate DDL in decoder-only language models with both scalar and expanded residual states, while keeping attention and MLP sublayers at the original compute width. Controlled pretraining and downstream evaluations show that residual rewrite operations improve language modeling quality relative to pure additive accumulation introduced in ResNet, suggesting that a learned delta-rule update is an effective mechanism for managing Transformer residual streams.

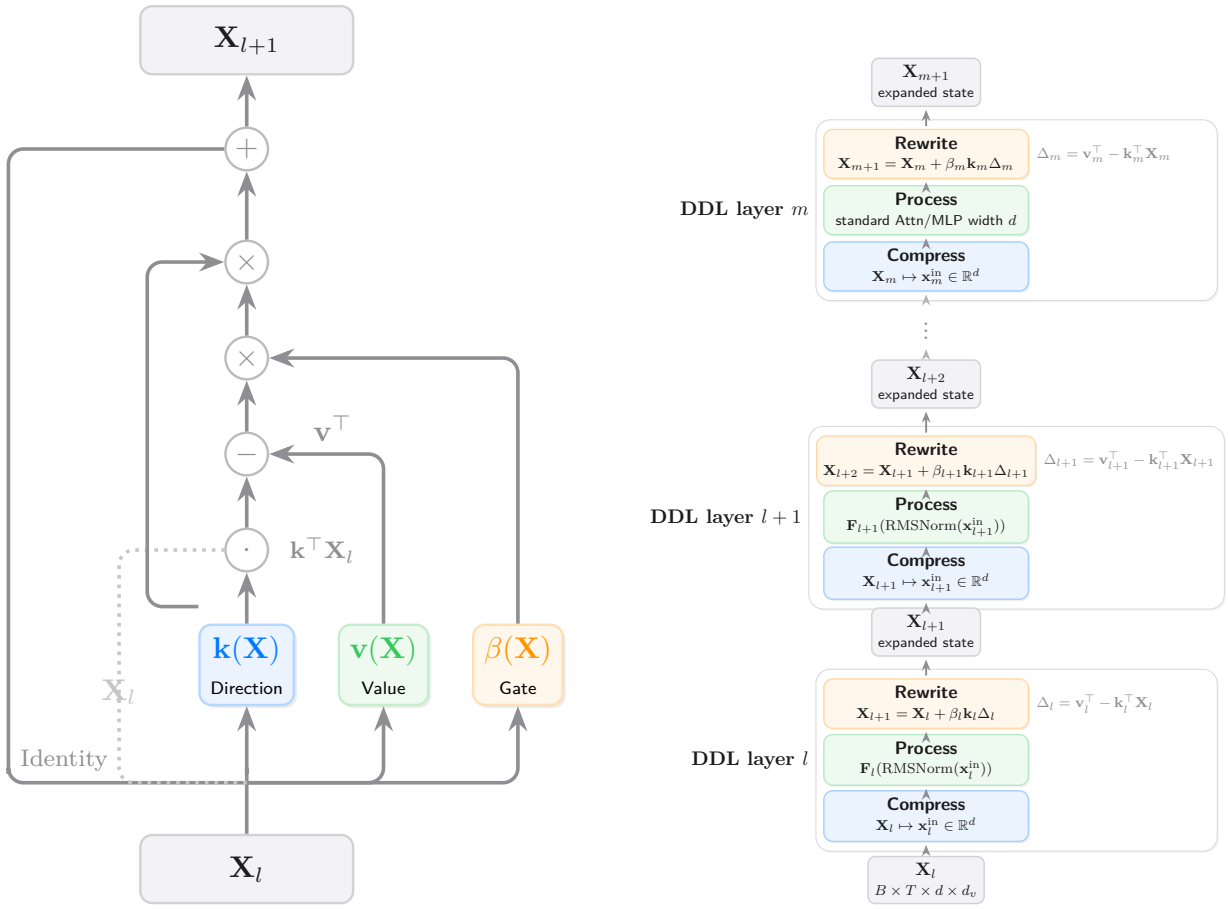
Project Page: https://github.com/yifanzhang-pro/deep-delta-learning

1 Introduction

Residual connections provide the default interface for composing very deep networks: an identity path stabilizes optimization while each block contributes an incremental transformation (He et al., 2016). In Transformer language models, the same interface also serves as the persistent token-level state shared by all attention and MLP blocks. A standard block updates this residual stream by addition:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathbf{F}_l(\mathbf{x}_l),$$

where $\mathbf{x}_l \in \mathbb{R}^{B\times T\times d}$ for batch size B, sequence length T, and compute width d. This additive rule is simple, stable, and hardware-friendly, but it has an important asymmetry: accumulating new features is direct, whereas replacing an obsolete, redundant, or conflicting component must be expressed indirectly by producing an update that cancels the old content and writes the new content at the same time. Prior residual and gated-pathway designs regulate information flow across depth (Srivastava et al., 2015; Menghani et al., 2025; Zhu et al., 2025; Xie et al., 2025), yet the local residual-stream primitive remains largely additive. As a result, residual-state replacement is treated as an implicit consequence of vector addition rather than as an explicit operation.



(a) Deep Delta residual block. The architecture generalizes the standard residual connection. A learnable scalar gate β controls a shortcut operator with a rank-1 perturbation.

(b) DDL Transformer structure. A DDL Transformer repeatedly applies a Compress-Process-Rewrite interface across depth.

Figure 1 Deep Delta Learning overview. (a) The Deep Delta residual block generalizes the standard residual connection with a gated rank-1 rewrite operation. (b) A DDL Transformer repeatedly applies a Compress-Process-Rewrite interface across depth, where the Process part generates the DDL key $\mathbf{k}$ and value $\mathbf{v}$ used by the Rewrite update, while keeping attention and MLP sublayers at width d. Each layer compresses the expanded residual state to a width-d token representation, processes it with the standard attention or MLP sublayer, uses the Process output and normalized residual context to generate the DDL key $\mathbf{k}_l$, value $\mathbf{v}_l$, and gate β_l, and then rewrites the persistent expanded state through the DDL read-compare-write update.

We introduce **Deep Delta Learning (DDL)**, a residual interface that preserves the identity shortcut while making localized replacement a first-class primitive. DDL applies a depth-wise analogue of delta-rule memory updates used in efficient sequence models (Schlag et al., 2021; Yang et al., 2024):

$$\mathbf{X}_{l+1} = \mathbf{X}_l + \beta_l \mathbf{k}_l (\mathbf{v}_l^\top - \mathbf{k}_l^\top \mathbf{X}_l).$$

The update reads the current residual content along a learned unit direction $\mathbf{k}_l$, compares that readout with a learned target $\mathbf{v}_l^\top$, and writes the gated correction back into the same subspace.

A single scalar gate β_l synchronizes erasure and writing: closing the gate recovers the identity map, while setting $\beta_l = 1$ exactly overwrites the selected component. This shared-gate coupling distinguishes DDL from generic gated residual branches and connects it to low-rank erase/write memory mechanisms (Schlag et al., 2021; Mak and Flanigan, 2025) while applying the operation over network depth rather than sequence time.

DDL can operate on the ordinary scalar residual state ($d_v = 1$) or on an expanded residual state $\mathbf{X}_l \in \mathbb{R}^{B\times T\times d\times d_v}$ with a small number of value channels. For $d_v > 1$, the expanded state is compressed to a width-d representation before the standard attention or MLP sublayer and is then updated by the DDL rewrite rule. This separates residual storage from backbone compute: the persistent state gains additional editable memory, while attention keys, queries, values, and MLP activations remain at the original width d.

We evaluate DDL in decoder-only language models trained on FineWeb-Edu for 49.15B tokens at GPT-2 small and GPT-2 medium parameter budgets. In our single-run comparisons, the best expanded-state variants improve validation loss by 0.024 at the GPT-2 small scale and 0.030 at the GPT-2 medium scale, and raise average one-shot downstream accuracy by 0.91 and 1.18 points, respectively. These gains are obtained without widening the attention or MLP backbone.

Our contributions in this paper are:

1. We formulate **Deep Delta Learning**, a residual rewrite rule that augments the Transformer residual stream with a learned rank-1 read-compare-write operation while preserving the identity limit. The shared gate synchronizes erasure and writing, recovering the skip connection when closed and exactly replacing the selected residual component when opened to $\beta_l = 1$.
2. We analyze the local operator induced by the DDL shortcut, $\mathbf{I} - \beta_l \mathbf{k}_l \mathbf{k}_l^\top$, showing how the learned gate controls preservation, overwrite, and stronger edit regimes along the chosen direction. This provides a geometric interpretation of DDL as a depth-wise delta-rule mechanism for residual-state replacement.
3. We integrate DDL into Transformer language models with both scalar and expanded residual states while keeping attention and MLP computation at width d unchanged. Controlled pretraining, downstream evaluation, and cost measurements at GPT-2 small and medium scales show that residual rewrite operations improve language modeling quality relative to additive residual baselines in our runs, while exposing the memory and throughput costs of expanded residual states.

2 Deep Delta Learning

Deep Delta Learning (DDL) changes the residual interface, not the attention or MLP sublayers. Its purpose is to give each layer a controlled edit operation on the persistent residual stream: when no edit is needed, the layer should reduce to the identity; when an edit is useful, it should replace a selected component rather than rely on an unconstrained additive vector to cancel and rewrite it implicitly. We derive this mechanism for a single token and a single layer. Throughout this section, we suppress the batch and sequence axes and write the residual state as $\mathbf{X}_l \in \mathbb{R}^{d\times d_v}$, where d is the Transformer model hidden size and d_v is the residual memory expansion ratio. The ordinary vector residual stream is recovered by $d_v = 1$.

2.1 Delta Residual Rewrite

The DDL update has three generated quantities: a direction that identifies where to edit, a value that specifies what to write, and a gate that determines how strongly to apply the edit. Coupling these quantities gives a depth-wise delta rule for residual-state replacement.

Read/write direction. A lightweight branch first proposes an unnormalized direction $\widetilde{\mathbf{k}}(\mathbf{X}_l) \in \mathbb{R}^d$. We normalize it as

$$\mathbf{k}_l = \mathbf{k}(\mathbf{X}_l) = \frac{\widetilde{\mathbf{k}}(\mathbf{X}_l)}{\|\widetilde{\mathbf{k}}(\mathbf{X}_l)\|_2}, \qquad \|\mathbf{k}_l\|_2 = 1,$$

with the usual small-norm guard in implementation. The unit-norm constraint makes $\mathbf{k}_l^\top \mathbf{X}_l$ a normalized readout of the current residual content and gives the gate a consistent scale across layers. The value branch produces $\mathbf{v}_l = \mathbf{v}(\mathbf{X}_l) \in \mathbb{R}^{d_v}$, interpreted as the target content for this readout, and the gate branch produces a scalar $\beta_l = \beta(\mathbf{X}_l)$.

Erase and write as one affine update. Conditioned on $\mathbf{X}_l$, define the direct shortcut operator

$$\mathbf{A}_l = \mathbf{A}(\mathbf{X}_l) = \mathbf{I} - \beta_l \mathbf{k}_l \mathbf{k}_l^\top.$$

The DDL block applies this shortcut to the current residual state and then writes a rank-1 component along the same direction:

$$\mathbf{X}_{l+1} = \mathbf{A}_l \mathbf{X}_l + \beta_l \mathbf{k}_l \mathbf{v}_l^\top. \tag{2.1}$$

Equivalently,

$$\mathbf{X}_{l+1} = \mathbf{X}_l + \beta_l \mathbf{k}_l \Big(\mathbf{v}_l^\top - \mathbf{k}_l^\top \mathbf{X}_l \Big). \tag{2.2}$$

The row vector $\mathbf{v}_l^\top - \mathbf{k}_l^\top \mathbf{X}_l \in \mathbb{R}^{1 \times d_v}$ is the residual error in the selected coordinate: it vanishes when the current readout already matches the target. Multiplying this error by $\mathbf{k}_l$ writes the correction back only into $\mathrm{span}\{\mathbf{k}_l\}$, leaving the orthogonal subspace on the direct shortcut path unchanged.

Shared gate. The same β_l gates both erasure and writing. This coupling is what turns the operation into a delta rule rather than a generic gated residual branch. At $\beta_l = 0$, Eq. (2.2) gives $\mathbf{X}_{l+1} = \mathbf{X}_l$, so the complete block has an identity limit. At $\beta_l = 1$, projection onto the chosen direction gives

$$\mathbf{k}_l^\top \mathbf{X}_{l+1} = \mathbf{v}_l^\top, \tag{2.3}$$

so the selected component is exactly overwritten by the target value. Using separate erase and write gates would be more general, but it would permit erase-without-write or write-without-erase behavior and would lose this clean read-compare-write interpretation.

Gate parameterization. In the experiments, we parameterize the gate as

$$\beta(\mathbf{X}) = 2 \cdot \sigma(\mathrm{Linear}(\mathcal{G}(\mathbf{X}))),$$

where $\mathcal{G}$ denotes the context readout used by the gate branch and σ denotes the logistic sigmoid. The value branch itself is linear in the reported experiments; the sigmoid is used only for the gate. Thus $\beta \in (0, 2)$ in normal operation, with the endpoint cases interpreted as saturated-logit limits. This range covers identity-like, overwrite-like, and stronger over-relaxed edit regimes while keeping the shortcut spectrum bounded.

2.2 Expanded Residual State

The matrix form above is not only notation; it separates residual memory from sublayer compute. In a standard Transformer, the same width d must both store the persistent residual stream and feed the attention and MLP blocks. DDL instead lets the residual stream carry d_v value channels per feature direction while the expensive backbone computation remains width d.

Concretely, the baseline residual state has shape $\mathbf{x}_l \in \mathbb{R}^{B\times T\times d}$. DDL may instead maintain $\mathbf{X}_l \in \mathbb{R}^{B\times T\times d\times d_v}$. For $d_v > 1$, each token stores a small matrix of residual values. Before a standard Transformer sublayer, a learned compressor reads this expanded state into a vector $\mathbf{x}_l^{\text{in}} \in \mathbb{R}^d$; the attention or MLP sublayer is then applied at the original compute width d; finally, the resulting representation and normalized residual context are used to generate the unnormalized direction $\widetilde{\mathbf{k}}_l$, the normalized direction $\mathbf{k}_l$, $\mathbf{v}_l$, and β_l for the update in Eq. (2.2). The resulting protocol is therefore compress, process, and rewrite.

This design increases residual-state storage and the cost of compression/rewrite operations, but it does not widen the attention keys, queries, values, or MLP activations from d to $d \cdot d_v$. The added capacity is used as persistent editable memory, rather than as a uniformly wider compute path.

2.3 Spectral Analysis

We next isolate the direct shortcut in Eq. (2.1) to clarify what the gate controls locally. The complete layer is state-dependent because $\beta(\mathbf{X})$, $\mathbf{k}(\mathbf{X})$, and $\mathbf{v}(\mathbf{X})$ are generated from the input; the following frozen spectrum therefore describes only the shortcut operator after conditioning on a particular layer, token, and residual state. It is not a claim about the full Jacobian of the nonlinear DDL block.

Proposition 2.1 (Frozen shortcut spectrum)**.** Let $\mathbf{A} = \mathbf{I} - \beta\mathbf{k}\mathbf{k}^\top$, where $\mathbf{k} \in \mathbb{R}^d$ is a unit vector and $\beta \in \mathbb{R}$ is fixed. If $\beta \neq 0$, the eigenvalues of $\mathbf{A}$ are 1 with multiplicity $d-1$ and $1-\beta$ with multiplicity 1. The eigenvector for $1-\beta$ is $\mathbf{k}$, and the eigenspace for eigenvalue 1 is $\mathbf{k}^\perp = \{\mathbf{u} \in \mathbb{R}^d : \mathbf{k}^\top\mathbf{u} = 0\}$. If $\beta = 0$, then $\mathbf{A} = \mathbf{I}$ and the eigenspace for eigenvalue 1 is all of $\mathbb{R}^d$.

Proof. For any $\mathbf{u} \in \mathbf{k}^\perp$, $\mathbf{A}\mathbf{u} = \mathbf{u} - \beta\mathbf{k}(\mathbf{k}^\top\mathbf{u}) = \mathbf{u}$. Also, $\mathbf{A}\mathbf{k} = \mathbf{k} - \beta\mathbf{k}(\mathbf{k}^\top\mathbf{k}) = (1-\beta)\mathbf{k}$. When $\beta \neq 0$, these d independent eigen-directions span $\mathbb{R}^d$; when $\beta = 0$, the claim reduces to $\mathbf{A} = \mathbf{I}$. □

For any vector $\mathbf{u} = \mathbf{u}_\perp + (\mathbf{k}^\top\mathbf{u})\mathbf{k}$ with $\mathbf{u}_\perp \in \mathbf{k}^\perp$,

$$\mathbf{A}\mathbf{u} = \mathbf{u}_\perp + (1-\beta)(\mathbf{k}^\top\mathbf{u})\mathbf{k}.$$

Thus, the frozen shortcut leaves $\mathbf{k}^\perp$ unchanged and scales only the selected component. Since the matrix-valued residual state is updated by left-multiplication, the same spatial operator is applied independently to each of the d_v value columns. Under vectorization, the induced shortcut is $\mathbf{I}_{d_v} \otimes \mathbf{A}$.

The corresponding selected readout evolves as

$$\mathbf{k}^\top\mathbf{X}_{l+1} = (1-\beta)\mathbf{k}^\top\mathbf{X}_l + \beta\mathbf{v}^\top = \mathbf{v}^\top + (1-\beta)(\mathbf{k}^\top\mathbf{X}_l - \mathbf{v}^\top),$$

which makes the gate regimes explicit:

- $\beta \approx 0$**: skip.** The shortcut approaches $\mathbf{I}$, the write term vanishes, and the complete block approaches the identity map.
- $\beta \approx 1$**: overwrite.** The shortcut removes the old $\mathbf{k}$-component and the synchronized write inserts $\mathbf{v}^\top$, giving exact replacement at $\beta = 1$.

- $\beta > 1$: **over-relaxed edit.** The selected coordinate moves past the target because $1 - \beta < 0$. At the saturated endpoint $\beta \to 2$, the direct shortcut approaches the Householder reflector $\mathbf{I} - 2\mathbf{k}\mathbf{k}^\top$.

This local geometry is the mechanism DDL exposes to every layer: preserve most of the residual stream by default, choose a single direction to edit, and use one gate to interpolate between skipping, overwriting, and stronger corrective updates.

3 DDL Transformer

We evaluate DDL as a drop-in replacement for additive residual connections in decoder-only Transformer language models. Across comparisons, the backbone uses pre-norm RMSNorm, RoPE multi-head attention, and SwiGLU MLPs; each DDL implementation changes the residual update while preserving the attention/MLP compute width d. Unless explicitly qualified by "w/o EC", expanded-state DDL variants use EC by default. In the experimental sections, bare DDL denotes DDL-CC.

3.1 Scalar residual state: $d_v = 1$

When $d_v = 1$, the residual state is a vector $\mathbf{x}_l \in \mathbb{R}^d$ and Eq. (2.2) becomes

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \beta_l(v_l - \mathbf{k}_l^\top \mathbf{x}_l)\mathbf{k}_l.$$

For each attention or MLP sublayer, let $\mathbf{h}_l = \mathbf{F}_l(\mathrm{RMSNorm}(\mathbf{x}_l)) \in \mathbb{R}^d$ denote the standard backbone output. In the main experiments, we use this output as the unnormalized rewrite direction, $\widetilde{\mathbf{k}}_l = \mathbf{h}_l$, followed by normalization. The scalar write value v_l is produced by a lightweight linear projection from the normalized residual context, while β_l is produced by the bounded gate branch. This keeps the high-capacity Transformer sublayer responsible for choosing the rewrite direction while adding only a small value/gate branch.

Precision-friendly normalization. For low-precision training, we implement the $\|\mathbf{k}_l\|_2 \approx 1$ constraint via RMS normalization and a fixed scaling factor $k_{\mathrm{scale}} = 1/\sqrt{d}$:

$$\widehat{\mathbf{k}} = \mathrm{RMSNorm}(\widetilde{\mathbf{k}}; \epsilon_k^2/d), \qquad \mathbf{k} = \widehat{\mathbf{k}}/\sqrt{d}.$$

With $\epsilon_k = 0$, this is exact L_2 normalization. With $\epsilon_k > 0$, it is equivalent to $\mathbf{k} = \widetilde{\mathbf{k}}/\sqrt{\|\widetilde{\mathbf{k}}\|_2^2 + \epsilon_k^2}$, so the spectral interpretation applies whenever $\|\widetilde{\mathbf{k}}\|_2 \gg \epsilon_k$.

3.2 Expanded residual state: $d_v > 1$

For expanded-state DDL, the residual stream has shape $\mathbf{X}_l \in \mathbb{R}^{B\times T\times d\times d_v}$. We evaluate $d_v = 4$. By default, the embedding layer initializes this expanded state with embedding convolution (EC), a learnable depthwise causal short convolution that maps token embeddings into the d_v value channels. In no-EC ablations, the state is initialized by repeating the token embedding across value channels, $\mathbf{X}_0 = \mathbf{x}_{\mathrm{emb}}\mathbf{1}_{d_v}^\top$.

To interface with ordinary Transformer sublayers, we use a Compress-Process-Rewrite protocol:

1. **Compress:** read $\mathbf{X}_l$ into a token representation $\mathbf{x}_l^{\mathrm{in}} \in \mathbb{R}^d$. In DDL-TC, a short causal depthwise convolution is applied over the token dimension independently per expanded channel, followed by learned pooling across d_v. In DDL-CC, compression instead mixes along the value-channel axis; under the revised naming, DDL-CC is our default expanded-state implementation.

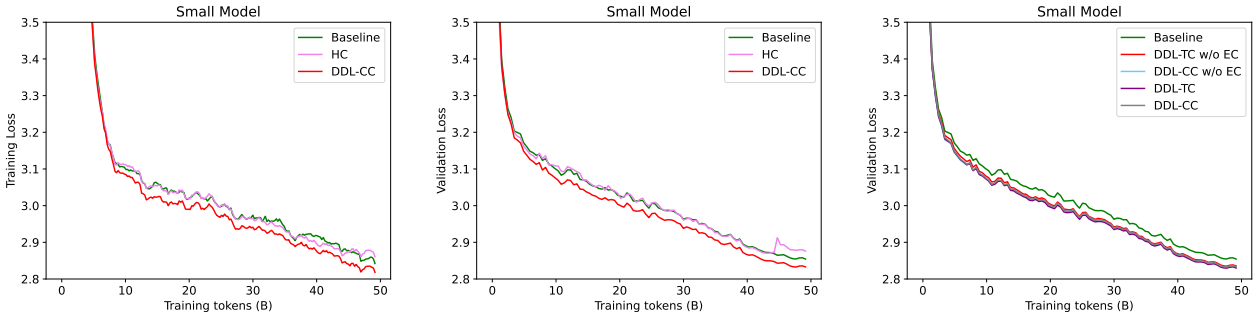
2. **Process:** apply the standard sublayer at compute width d, giving $\mathbf{h}_l = \mathbf{F}_l(\mathrm{RMSNorm}(\mathbf{x}_l^{\mathrm{in}})) \in \mathbb{R}^d$.
3. **Rewrite:** set the unnormalized direction to $\widetilde{\mathbf{k}}_l = \mathbf{h}_l$, normalize it to $\mathbf{k}_l = \widetilde{\mathbf{k}}_l / \|\widetilde{\mathbf{k}}_l\|_2$ using the precision-friendly normalization above, produce $\mathbf{v}_l \in \mathbb{R}^{d_v}$ by a lightweight linear projection and β_l by the bounded gate branch, and update the expanded residual state using Eq. (2.2).

The expanded state increases residual memory and the cost of compression/rewrite operations. It does not increase the dimensionality of the attention keys, queries, values, or MLP hidden activations, which remain tied to d. Per token and per layer, the DDL read-compare-write operation adds $O(dd_v)$ work and activation traffic. The compressor adds $O(kdd_v)$ work for the token-axis DDL-TC variant with kernel size k, and $O(dd_v)$ work for DDL-CC when the channel-axis kernel consumes the d_v value channels. The lightweight value and gate projections are lower-order relative to the standard attention and MLP blocks, but the additional residual state can be bandwidth- and memory-relevant; this is why we report empirical throughput and peak memory in Tables 4 and 5. For autoregressive generation, the token-axis compressor in DDL-TC also requires a short per-layer cache of the most recent expanded residual states; the channel-axis compressor in DDL-CC avoids this token-history cache because it mixes only across d_v at the current token.

We evaluate two main $d_v = 4$ implementations: token-axis convolution (DDL-TC) and channel-axis convolution (DDL-CC), both with EC enabled by default. To isolate the contribution of EC, we also report no-EC ablations, denoted DDL-TC w/o EC and DDL-CC w/o EC. DDL-CC is the default implementation denoted by the bare name DDL because it provides the strongest overall quality-cost tradeoff in our runs. These variants keep the comparison focused on the compression axis while treating EC as a default implementation enhancement.

4 Experiments

We compare a baseline based on the nanoGPT codebase (Karpathy, 2022) against a scalar-state DDL implementation ($d_v = 1$) and expanded-state DDL implementations ($d_v = 4$). The expanded-state implementations differ primarily in the compression axis. DDL-TC uses depthwise convolution (Chollet, 2017) along the sequence-length dimension with $d \cdot d_v$ channels, while DDL-CC instead compresses along the value-channel dimension. EC denotes the default embedding-convolution input expansion used by both DDL-TC and DDL-CC; disabling it gives the explicit ablations DDL-TC w/o EC and DDL-CC w/o EC. Unless otherwise specified, bare DDL refers to DDL-CC.



(a) Small (124M) Train Loss **(b)** Small (124M) Val Loss **(c)** Small DDL Variants Val Loss

Figure 2 Small-scale loss curves for the baseline, DDL, and DDL variants trained on FineWeb-Edu for 49.15B tokens.

Table 1 1-shot evaluation results for small models trained on FineWeb-Edu for 49.15B tokens and evaluated with lm-evaluation-harness. The best and second-best scores in each column are bolded and underlined, respectively. Abbreviations: WG = WinoGrande. For expanded-state DDL variants, the default value of d_v is 4, and EC is enabled unless the row is marked w/o EC. Bare DDL refers to DDL-CC.

Model	ARC-C	ARC-E	Hellaswag	OpenBookQA	PIQA	SciQ	Social IQA	WG	Avg.
Baseline	<u>29.01</u>	55.85	37.59	30.20	<u>65.94</u>	80.60	37.87	51.38	48.56
DDL ($d_v = 1$)	**29.35**	57.49	38.08	31.80	64.85	78.50	37.77	<u>52.01</u>	48.73
DDL-TC w/o EC	27.90	<u>58.16</u>	38.26	30.80	**66.49**	80.30	**38.54**	50.83	48.91
DDL-CC w/o EC	27.82	**58.92**	**38.44**	<u>33.20</u>	65.83	79.50	38.28	51.07	49.13
DDL-TC	28.75	57.37	<u>38.41</u>	**34.40**	64.47	<u>82.00</u>	38.38	<u>52.01</u>	**49.47**
DDL-CC	28.33	57.87	38.24	32.20	64.09	**82.60**	<u>38.43</u>	**52.57**	<u>49.29</u>

4.1 Experimental settings

We train on FineWeb-Edu (Lozhkov et al., 2024). Although the source corpus is commonly referred to as FineWeb-Edu 100B, each run in this paper is trained for 100,000 optimization steps with global batch size 480 sequences and sequence length 1,024. This gives 491,520 tokens per update and 49.15B total training tokens. We run experiments at two scales: small (124M parameters) and medium (353M parameters), using Llama-style models with RoPE (Su et al., 2024) embeddings and SwiGLU activations (Shazeer, 2020); we also add query and key normalization to stabilize training.

We use a shared μP-style parameterization and training recipe (Yang et al., 2022); we do not perform exhaustive per-method hyperparameter tuning. The learning rate is set to 1e$-$3 with a cosine learning-rate schedule and 2,000 warmup steps. We use AdamW (Loshchilov and Hutter, 2019) with weight decay 0.1, $(\beta_1, \beta_2) = (0.9, 0.95)$, and gradient clipping at 1.0. We disable bias terms and set dropout to 0.0. Each experiment uses 4 NVIDIA H200 GPUs. Other hyperparameters are listed in Appendix C.

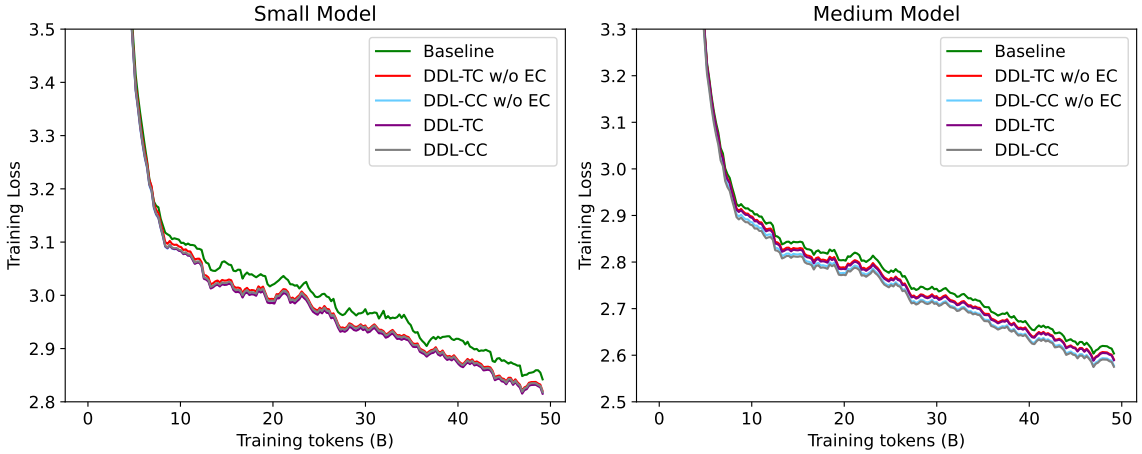
4.2 Experimental results

Figures 2, 4, and 3 summarize the available training and validation dynamics for the baseline, DDL, and DDL variants. Final validation losses and perplexities, including the no-EC ablations, are reported in Table 3, which should be read as the complete final-metric comparison. At both 124M and 353M scales, DDL improves validation loss in these single runs. The gains are modest for $d_v = 1$ and larger for $d_v = 4$, consistent with the hypothesis that expanded residual states provide additional useful capacity; because expanded variants also introduce compression and input-expansion operations, we avoid attributing all $d_v = 4$ gains solely to the rank-1 rewrite rule.

We also evaluate 1-shot and 0-shot performance on ARC (Yadav et al., 2019), HellaSwag (Clark et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), Social IQA (Sap et al., 2019), and WinoGrande (Sakaguchi et al., 2021) using `lm-evaluation-harness` (Gao et al., 2021). Tables 1 and 2 report 1-shot results; Appendix D reports 0-shot results. DDL improves the average 1-shot score over the baseline at both scales, whereas 0-shot averages depend on the implementation. We therefore treat these benchmarks as supporting evidence for the residual rewrite mechanism, rather than as a claim of uniform downstream dominance.

Tables 4 and 5 report the corresponding accuracy-cost tradeoff. The main expected pattern is that $d_v = 4$ improves the loss more than $d_v = 1$, but introduces explicit memory and throughput

overhead.

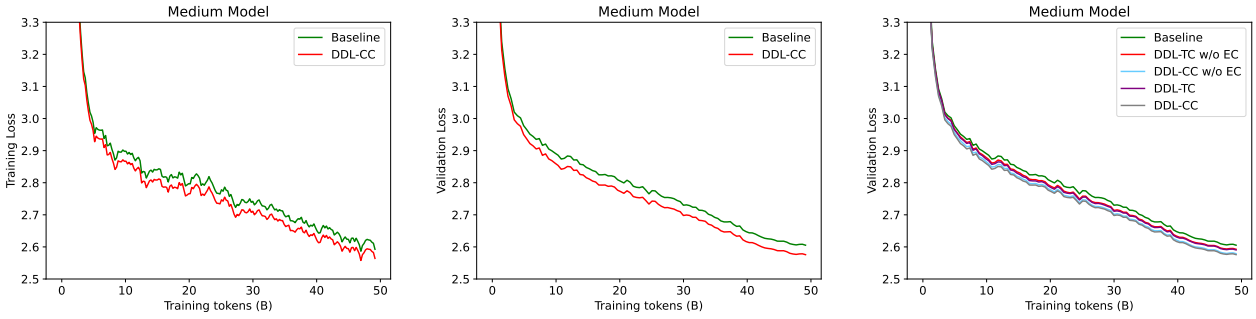


(a) Small DDL Variants Train Loss **(b)** Medium DDL Variants Train Loss

Figure 3 Training loss curves for DDL implementation variants at small (∼124M) and medium (∼353M) scales, trained on FineWeb-Edu for 49.15B tokens.

4.3 DDL variant results

We further report $d_v = 4$ DDL variants.[1] The two main implementations are token-axis convolution (DDL-TC) and channel-axis convolution (DDL-CC), both with EC enabled by default. We include DDL-TC w/o EC and DDL-CC w/o EC as ablations of the default input expansion. This naming keeps the comparison focused on TC versus CC rather than introducing separate EC-suffixed methods. Across the cost profiles, DDL-CC is the default because it provides the best overall quality–cost tradeoff. Figures 2, 4, and 3 show the corresponding loss curves; Tables 1 and 2 report 1-shot results, and Appendix Tables 7 and 8 report 0-shot results.



(a) Medium (353M) Train Loss **(b)** Medium (353M) Val Loss **(c)** Medium DDL Variants Val

Figure 4 Medium-scale loss curves for the baseline, DDL, and DDL variants trained on FineWeb-Edu for 49.15B tokens.

[1] Detailed descriptions are provided in Appendix A.3.

Table 2 1-shot evaluation results for medium models trained on FineWeb-Edu for 49.15B tokens and evaluated with lm-evaluation-harness. The best and second-best scores in each column are bolded and underlined, respectively. Abbreviations: WG = WinoGrande. For expanded-state DDL variants, the default value of d_v is 4, and EC is enabled unless the row is marked w/o EC. Bare DDL refers to DDL-CC.

Model	ARC-C	ARC-E	Hellaswag	OpenBookQA	PIQA	SciQ	Social IQA	WG	Avg.
Baseline	33.62	**67.05**	47.42	33.20	<u>70.24</u>	87.30	40.28	52.57	53.96
DDL ($d_v = 1$)	**35.49**	65.70	46.94	34.20	**70.35**	88.90	**40.99**	54.93	54.69
DDL-TC w/o EC	33.02	<u>66.16</u>	47.83	35.60	69.86	<u>89.50</u>	**40.99**	55.64	54.83
DDL-CC w/o EC	34.30	66.08	<u>48.65</u>	**36.60**	68.72	88.80	<u>40.84</u>	55.33	<u>54.92</u>
DDL-TC	<u>35.15</u>	65.70	47.74	35.40	69.04	88.60	40.63	**56.59**	54.86
DDL-CC	34.39	65.57	**48.92**	<u>36.00</u>	69.48	**90.50**	40.53	<u>55.72</u>	**55.14**

Table 3 Validation loss and perplexity for small and medium models at the final step after 49.15B training tokens. The best loss and perplexity in each column are bolded.

Model	Small		Medium	
	Valid Loss	Valid Perplexity	Valid Loss	Valid Perplexity
Baseline	2.8543	17.3616	2.6053	13.5356
DDL ($d_v = 1$)	2.8482	17.2562	2.6039	13.5161
DDL-TC w/o EC	2.8355	17.0381	2.5927	13.3654
DDL-CC w/o EC	2.8321	16.9811	2.5790	13.1834
DDL-TC	**2.8299**	**16.9438**	2.5905	13.3370
DDL-CC	2.8329	16.9947	**2.5758**	**13.1420**

Table 4 Small-model cost and quality summary. Throughput and memory are measured on the same hardware and software stack used for training. The peak-memory factor is normalized to the baseline. We use optimized Triton kernels for CC and EC-enabled implementations. For expanded-state DDL variants, the default value of d_v is 4, and EC is enabled unless the row is marked w/o EC. Bare DDL refers to DDL-CC.

Model	Params	Val loss	Avg eval	Train tok/s (K)	Inference tok/s (K)	Peak memory	Peak-memory factor
Baseline	123M	2.8543	48.56	1509.6	1826.1	2.94GB	1.00×
DDL ($d_v = 1$)	123M	2.8482	48.73	1330.8	1605.8	2.94GB	1.00×
DDL-TC	123M	2.8299	49.47	783.5	865.1	3.38GB	1.15×
DDL-CC	123M	2.8329	49.29	1158.0	1220.7	3.08GB	1.05×

Table 5 Medium-model cost and quality summary. Throughput and memory are measured on the same hardware and software stack used for training. The peak-memory factor is normalized to the baseline. We use optimized Triton kernels for CC and EC-enabled implementations. For expanded-state DDL variants, the default value of d_v is 4, and EC is enabled unless the row is marked w/o EC. Bare DDL refers to DDL-CC.

Model	Params	Val loss	Avg eval	Train tok/s (K)	Inference tok/s (K)	Peak memory	Peak-memory factor
Baseline	353M	2.6053	53.96	537.1	531.5	7.06GB	1.00×
DDL-TC	354M	2.5905	54.86	282.9	291.1	7.40GB	1.05×
DDL-CC	353M	2.5758	55.14	422.3	400.5	7.20GB	1.02×

5 Related Work

DDL is most closely related to three lines of work: residual and gated pathway design, delta-rule memory updates, and low-rank or orthogonal residual transformations.

Residual and gated pathways. Highway Networks (Srivastava et al., 2015) introduced data-dependent gates around residual pathways, and later work explored richer cross-layer or dense residual connections (Chai et al., 2020; Menghani et al., 2025; Pagliardini et al., 2024; Fang et al., 2023; Xiao et al., 2025). Hyper-Connections (Zhu et al., 2025) and manifold Hyper-Connections (Xie et al., 2025) expand or project residual streams to improve information flow. These methods primarily control how much information passes through or between layers. DDL instead changes the local residual operation itself: it gives each layer a rank-1 erase/write direction while retaining the identity limit.

Delta rules and memory updates. The DDL update is algebraically related to delta-rule memory updates used in efficient sequence models (Schlag et al., 2021; Yang et al., 2024). Those methods update memory over sequence time; DDL applies the same erase/write pattern over network depth, treating the residual stream as a state to be rewritten rather than only accumulated. Outer-product memory mechanisms in Transformers (Mak and Flanigan, 2025) are also related in their use of low-rank writes, but DDL couples the write to an explicit erase term along the same direction.

Orthogonal and low-rank transformations. Householder reflections are a classical way to parameterize orthogonal transformations, and have been used in neural architectures and adaptation methods (Yang et al., 2025; Dong et al., 2024; Arcas et al., 2025). Other work constrains weights or residual maps to be orthogonal or unitary for stability (Arjovsky et al., 2016; Jing et al., 2017; Zhang et al., 2021; Fei et al., 2022; Wang et al., 2025; He et al., 2025). DDL does not impose global orthogonality. Its frozen shortcut is identity-like, projection-like, or reflection-like, depending on the learned gate, and the complete layer remains state-dependent.

6 Conclusion

We have introduced Deep Delta Learning, a residual update rule that reframes the Transformer residual stream as an editable state rather than an append-only accumulator. Each DDL layer reads the current content along a learned direction, compares it with a learned target, and writes the gated correction back, preserving the identity shortcut that makes deep networks trainable while providing every layer with an explicit mechanism for localized content replacement. In controlled pretraining at GPT-2 small and GPT-2 medium sizes, DDL improves validation loss and 1-shot average downstream accuracy over additive baselines in our single-run comparisons. The largest gains arise from expanded residual states that increase memory capacity without widening attention or MLP computation, at the cost of additional residual-state memory, bandwidth, and compressor overhead.

References

Alejandro Moreno Arcas, Albert Sanchis, Jorge Civera, and Alfons Juan. Hoft: Householder orthogonal fine-tuning. *arXiv preprint arXiv:2505.16531*, 2025.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

Yekun Chai, Shuo Jin, and Xinwen Hou. Highway transformer: Self-gating enhanced self-attentive networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6887–6900, 2020.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Wei Dong, Yuan Sun, Yiting Yang, Xing Zhang, Zhijun Lin, Qingsen Yan, Haokui Zhang, Peng Wang, Yang Yang, and Hengtao Shen. Efficient adaptation of pre-trained vision transformer via householder transformation. *Advances in Neural Information Processing Systems*, 37:102056–102077, 2024.

Yanwen Fang, Yuxi Cai, Jintai Chen, Jingyu Zhao, Guangjian Tian, and Guodong Li. Cross-layer retrospective retrieving via layer attention. *arXiv preprint arXiv:2302.03985*, 2023.

Yanhong Fei, Yingjie Liu, Xian Wei, and Mingsong Chen. O-vit: Orthogonal vision transformer. *arXiv preprint arXiv:2201.12133*, 2022.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yi He, Yiming Yang, Xiaoyuan Cheng, Hai Wang, Xiao Xue, Boli Chen, and Yukun Hu. Chaos meets attention: Transformers for large-scale dynamical prediction. *arXiv preprint arXiv:2504.20858*, 2025.

Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning*, pages 1733–1741. PMLR, 2017.

Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024.

Brian Mak and Jeffrey Flanigan. Residual matrix transformers: Scaling the size of the residual stream. In *Forty-second International Conference on Machine Learning*, 2025.

Gaurav Menghani, Ravi Kumar, and Sanjiv Kumar. Laurel: Learned augmented residual layer. In *Forty-second International Conference on Machine Learning*, 2025.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. Denseformer: Enhancing information flow in transformers via depth weighted averaging. *Advances in neural information processing systems*, 37:136479–136508, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pages 9355–9366. PMLR, 2021.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Xiangming Wang, Haijin Zeng, Jiaoyang Chen, Sheng Liu, Yongyong Chen, and Guoqing Chao. Otlrm: Orthogonal learning-based low-rank metric for multi-dimensional inverse problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 21278–21286, 2025.

Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

Da Xiao, Qingye Meng, Shengping Li, and Xingyuan Yuan. Muddformer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections. In *Forty-second International Conference on Machine Learning*, 2025.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Liang Zhao, et al. mhc: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. *arXiv preprint arXiv:1911.07176*, 2019.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Songlin Yang, Yikang Shen, Kaiyue Wen, Shawn Tan, Mayank Mishra, Liliang Ren, Rameswar Panda, and Yoon Kim. Path attention: Position encoding via accumulating householder transformations. *arXiv preprint arXiv:2505.16381*, 2025.

Aston Zhang, Alvin Chan, Yi Tay, Jie Fu, Shuohang Wang, Shuai Zhang, Huajie Shao, Shuochao Yao, and Roy Ka-Wei Lee. On orthogonality constraints for transformers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 375–382, 2021.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *The Thirteenth International Conference on Learning Representations*, 2025.

Appendix

A Implementation and Parameterization Details

The Deep Delta Learning (DDL) framework relies on the efficient estimation of the reflection direction $\mathbf{k}(\mathbf{X})$, the scalar gate $\beta(\mathbf{X})$, and the residual value $\mathbf{v}(\mathbf{X})$. While the local frozen-operator interpretation holds regardless of the specific topology used to approximate these functions, we outline the lightweight generator parameterizations used in the experiments. The suffixes TC and CC denote user-facing compression choices; EC is enabled by default and disabled only in rows marked w/o EC. The mathematical update is shared across all variants; in the main text, bare DDL denotes DDL-CC.

Let the hidden state be $\mathbf{X} \in \mathbb{R}^{d \times d_v}$. We denote the generator branch for the reflection vector as a function $\phi_k : \mathbb{R}^{d \times d_v} \to \mathbb{R}^d$.

A.1 Parameterization of the Reflection Direction $\mathbf{k}(\mathbf{X})$

The geometric orientation of the Delta Operator is determined by $\mathbf{k}$. The experiments use a lightweight generator for ϕ_k; more expressive generators are possible but are not evaluated here.

MLP Parameterization.

For architectures prioritizing global feature mixing with low computational overhead, we parameterize $\mathbf{k}$ using a Multi-Layer Perceptron (MLP) acting on aggregated statistics of the state matrix.

$$\widetilde{\mathbf{k}}_{\text{MLP}} = \text{MLP}(\text{Pool}(\mathbf{X})), \quad \mathbf{k}_{\text{MLP}} = \frac{\widetilde{\mathbf{k}}_{\text{MLP}}}{\|\widetilde{\mathbf{k}}_{\text{MLP}}\|_2}$$

Here, $\text{Pool}(\cdot)$ is any aggregation that produces a fixed-size vector representation of $\mathbf{X}$, e.g., column-wise averaging ($\mathbb{R}^{d \times d_v} \to \mathbb{R}^d$) or flattening ($\mathbb{R}^{d \times d_v} \to \mathbb{R}^{d \cdot d_v}$), followed by an MLP that outputs $\mathbb{R}^d$. We enforce L_2 normalization to satisfy the spectral assumptions in Proposition 2.1. In implementation, the division is guarded by a small ϵ_k only when $\|\widetilde{\mathbf{k}}_{\text{MLP}}\|_2$ is extremely small; the analysis assumes $\|\mathbf{k}\|_2 = 1$.

A.2 Parameterization of the Gate $\beta(\mathbf{X})$ and Value $\mathbf{v}(\mathbf{X})$

The Gating Branch. The scalar gate β requires a bounded output in $[0, 2]$.

In `model/DDL-gpt-mha-rope*.py`, β is computed per token from the same pre-norm context used by the backbone, $\mathbf{c} = \text{RMSNorm}(\mathbf{x}^{\text{in}}) \in \mathbb{R}^d$ (where $\mathbf{x}^{\text{in}}$ is the compressed readout of the expanded state described above). We keep this estimator lightweight and configurable (`ddl_beta_single_linear`, `ddl_beta_hidden_size`):

$$\beta(\mathbf{c}) = 2 \cdot \sigma(\text{Linear}(\mathbf{c})) \quad \text{or} \quad \beta(\mathbf{c}) = 2 \cdot \sigma\big(\text{Linear}(\tanh(\text{Linear}(\mathbf{c})))\big).$$

We compute the logits in fp32 for stability and initialize the output bias to match the configured starting value $\beta_0 =$ `ddl_beta_init` $\in [0, 2]$ via $\text{logit}(\beta_0/2)$ (clamped in code), ensuring smooth interpolation between identity, projection, and reflection. The same configuration value is used for all reported variants unless explicitly stated otherwise.

The Value Branch. The residual value vector $\mathbf{v} \in \mathbb{R}^{d_v}$ represents the write target. In the Transformer experiments, $\mathbf{v}$ is produced by a lightweight linear projection from the compressed residual context, $\text{Linear} : \mathbb{R}^d \to \mathbb{R}^{d_v}$, while the main attention or MLP sublayer supplies the rewrite direction $\mathbf{k}$ in the primary configuration. We do not instantiate an additional attention or MLP block for the value branch, which keeps the parameter accounting tied to the listed lightweight projections.

A.3 Expanded-state Transformer Implementation Details

Our PyTorch implementations for the expanded-state Transformer variant ($d_v > 1$) live in `model/DDL-gpt-mha-rope*.py`. For clarity, we summarize the common tensor layout and then highlight the key differences between the repository variants.

Repository variants.

- `model/DDL-gpt-mha-rope-TC-EC.py`: user-facing DDL-TC. This is the default token-compression implementation of expanded-state DDL on top of GPT (MHA + RoPE). It expands token embeddings with a learnable depthwise causal short convolution (`input_embed_shortconv_kernel_size`, identity-initialized) and compresses the expanded residual with a token-axis short causal convolution plus a learned read vector.
- `model/DDL-gpt-mha-rope-TC.py`: user-facing DDL-TC w/o EC. This ablation keeps the token-axis residual compressor but initializes the state by repeating token embeddings across the value-channel axis.
- `model/DDL-gpt-mha-rope-CC-EC.py`: user-facing DDL-CC and the default implementation denoted by bare DDL in the main text. It expands token embeddings with a learnable depthwise causal short convolution and compresses the expanded residual along the value-channel axis d_v; the Conv consumes the value axis and returns length 1 (`ddl_state_shortconv_kernel_size` $= d_v$).
- `model/DDL-gpt-mha-rope-CC.py`: user-facing DDL-CC w/o EC. This ablation keeps the channel-axis residual compressor but initializes the state by repeating token embeddings across the value-channel axis.

State layout and initialization. For a batch of sequences, we represent the expanded residual as a rank-4 tensor of shape (B, T, d, d_v), where d is the model width and d_v is the number of value channels.

Input expansion. In the default DDL-TC and DDL-CC modules, EC uses a depthwise causal short convolution over the token dimension to map $(B, T, d) \to (B, T, d \cdot d_v)$ and then reshapes to (B, T, d, d_v); the convolution is identity-initialized, so the starting behavior matches simple repetition. In the w/o EC ablations, we initialize $\mathbf{X}_0 = \mathbf{x}_{\text{emb}} \mathbf{1}_{d_v}^\top$ by repeating the embedding across the value axis (implemented as `x_emb.unsqueeze(-1).repeat(..., d_v)`).

ShortConv compression along T (DDL-TC and DDL-TC w/o EC).
In `model/DDL-gpt-mha-rope-TC-EC.py` and `model/DDL-gpt-mha-rope-TC.py`, the `ResidualShortConvCompressor` first flattens the last two dimensions, treating the expanded residual as $(B, T, d \cdot d_v)$ channels, applies a short causal depthwise Conv1d over the token dimension (kernel size `ddl_state_shortconv_kernel_size`), reshapes back to (B, T, d, d_v), and then pools across value channels with a learned read vector $\mathbf{w}_p \in \mathbb{R}^{d_v}$:

$$\widetilde{\mathbf{X}}_{l,t,i,j} = \sum_{s=0}^{k-1} c_{i,j,s}\, \mathbf{X}_{l,t-s,i,j}, \qquad \mathbf{x}^{\text{in}}_{l,t,i} = \sum_{j=1}^{d_v} w_{p,j}\, \widetilde{\mathbf{X}}_{l,t,i,j}.$$

We initialize the read vector to a uniform average by default (`ddl_state_read_init` $= 1/d_v$ when unset). During autoregressive generation, this token-axis convolution requires retaining the last $k - 1$ expanded residual states per layer, in addition to the usual attention KV cache.

ShortConv compression along d_v (DDL-CC and DDL-CC w/o EC).
In `model/DDL-gpt-mha-rope-CC-EC.py` and `model/DDL-gpt-mha-rope-CC.py`, we instead apply a depthwise Conv1d along the value-channel axis and consume d_v in one shot (the Conv returns length 1). Concretely, we reshape the state to $(B \cdot T, d, d_v)$ and treat d_v as the convolution "sequence" length, using per-feature kernels $c_{i,j}$ with kernel size $k = d_v$:

$$\mathbf{x}^{\text{in}}_{l,t,i} = \sum_{j=1}^{d_v} c_{i,j}\, \mathbf{X}_{l,t,i,j}.$$

This changes the locality prior from time-local mixing to value-channel-local mixing. The EC-enabled CC implementation is the default DDL configuration in our runs because it provides the best overall quality–cost tradeoff. Note that in CC, `ddl_state_shortconv_kernel_size` refers to the kernel size along d_v (not along T), and must equal d_v for the Conv to return length 1.

B Relation to DeltaNets and Householder Products

Our work shares a theoretical link with the DeltaNet architecture (Schlag et al., 2021), which replaces the additive accumulation of Linear Transformers with a Delta Rule for memory updates.

We spell out the algebraic analogy between Deep Delta Learning and the DeltaNet recurrence. In DeltaNet, the hidden state (memory) $\mathbf{S}_t$ evolves over time t. To unify notation with our depth-wise formulation, we present the DeltaNet update using left-multiplication semantics, where the memory state is $\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$:

$$\mathbf{S}_t = (\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)\mathbf{S}_{t-1} + \beta_t \mathbf{k}_t \mathbf{v}_t^\top. \tag{B.1}$$

Here, the operator acts on the key dimension d_k, which is analogous to the feature dimension d in DDL. Comparing this to our Deep Delta Layer update Eq. (2.1) acting over depth l: Eq. (B.1) is algebraically equivalent to the more common right-multiplication delta-rule update $\mathbf{M}_t = \mathbf{M}_{t-1} + \beta_t(\mathbf{v}_t - \mathbf{M}_{t-1}\mathbf{k}_t)\mathbf{k}_t^\top$ by setting $\mathbf{S}_t = \mathbf{M}_t^\top$ (a transpose convention for the memory matrix).

$$\mathbf{X}_{l+1} = (\mathbf{I} - \beta_l \mathbf{k}_l \mathbf{k}_l^\top)\mathbf{X}_l + \beta_l \mathbf{k}_l \mathbf{v}_l^\top,$$

where $\mathbf{v}_l$ is the vector output of the value branch.

This reveals a direct algebraic correspondence:

- The memory state $\mathbf{S}_t$ (dimension $d_k \times d_v$) in DeltaNet corresponds to the feature activation $\mathbf{X}_l$ (dimension $d \times d_v$) in DDL.
- Both architectures employ the same shortcut operator with a rank-1 perturbation, $\mathbf{I} - \beta\,\mathbf{k}\mathbf{k}^\top$: under $\|\mathbf{k}\|_2 = 1$ it is an orthogonal projector at $\beta = 1$ and a Householder reflection at $\beta = 2$. DeltaNet applies it over time steps t, whereas DDL applies it over network depth l.
- Our modified residual update $\beta_l \mathbf{k}_l \mathbf{v}_l^\top$ aligns perfectly with the DeltaNet write operation. By incorporating β_l into the constructive term, we interpret β_l as a depth-wise delta-rule step size. This ensures that erasure and injection are modulated synchronously, preserving the identity limit of the complete update.

Thus, DDL can be interpreted as applying a delta-rule-style erase/write operation to layer-wise feature evolution.

C Hyper-parameter Settings

Table 6 lists the architecture hyperparameters for the small and medium model sizes.

Table 6 Architecture hyperparameters for the small and medium model sizes.

Model	#Param	#Layer	#Head	Head Dimension	Hidden Size
Small-size Model	124M	12	6	128	768
Medium-size Model	353M	24	8	128	1024

D Additional Results

Tables 7 and 8 report the 0-shot evaluation results for small and medium models, respectively. These results provide additional single-run evidence for the DDL residual rewrite mechanism.

Table 7 0-shot evaluation results for small models trained on FineWeb-Edu for 49.15B tokens and evaluated with lm-evaluation-harness. The best and second-best scores in each column are bolded and underlined, respectively. Abbreviations: WG = WinoGrande. For expanded-state DDL variants, the default value of d_v is 4, and EC is enabled unless the row is marked w/o EC. Bare DDL refers to DDL-CC.

Model	ARC-C	ARC-E	Hellaswag	OpenBookQA	PIQA	SciQ	Social IQA	WG	Avg.
Baseline	<u>28.33</u>	<u>52.44</u>	37.60	33.00	<u>65.94</u>	71.20	37.46	52.41	47.30
DDL ($d_v = 1$)	26.96	52.40	37.91	32.20	64.91	72.50	38.08	**53.59**	47.32
DDL-TC w/o EC	27.30	**52.95**	38.40	**33.80**	65.40	**73.70**	<u>38.64</u>	50.12	<u>47.54</u>
DDL-CC w/o EC	27.05	51.47	<u>38.72</u>	32.20	**66.16**	71.80	38.08	51.07	47.07
DDL-TC	**28.50**	51.89	**38.82**	<u>33.20</u>	65.29	73.00	**39.05**	<u>52.88</u>	**47.83**
DDL-CC	27.90	51.26	38.42	31.40	64.85	<u>73.60</u>	37.77	52.25	47.18

Table 8 0-shot evaluation results for medium models trained on FineWeb-Edu for 49.15B tokens and evaluated with lm-evaluation-harness. The best and second-best scores in each column are bolded and underlined, respectively. Abbreviations: WG = WinoGrande. For expanded-state DDL variants, the default value of d_v is 4, and EC is enabled unless the row is marked w/o EC. Bare DDL refers to DDL-CC.

Model	ARC-C	ARC-E	Hellaswag	OpenBookQA	PIQA	SciQ	Social IQA	WG	Avg.
Baseline	31.74	<u>59.85</u>	47.91	34.00	69.21	78.10	**40.69**	53.83	51.92
DDL ($d_v = 1$)	32.07	59.39	47.62	34.40	**70.08**	77.30	39.61	55.01	51.94
DDL-TC w/o EC	32.08	58.38	48.08	35.80	69.42	79.90	39.92	54.14	52.22
DDL-CC w/o EC	32.08	**61.74**	**49.12**	34.80	69.53	80.20	<u>40.43</u>	<u>55.17</u>	**52.88**
DDL-TC	**33.02**	59.68	48.00	**36.80**	68.77	<u>81.00</u>	39.82	**55.88**	<u>52.87</u>
DDL-CC	<u>32.59</u>	59.55	<u>49.01</u>	<u>36.00</u>	<u>69.70</u>	**82.30**	39.82	53.83	52.85