DEPTH: RECURSION ALONG A PHYSICAL COORDINATE FOR THERMO-MECHANICAL SURROGATE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Weight-shared recursive transformers such as the Universal Transformer (UT) apply a single block repeatedly, so the number of block applications and hence compute can grow without adding parameters. Their recursion, however, is often *unguided*: every step performs the same undifferentiated refinement toward the same target, and nothing distinguishes step 1 from step T. We study recursion in a setting where steps do carry meaning: *physically indexed sequence prediction*, in which a static condition determines an ordered sequence of outputs indexed by a physical coordinate, such as successive slices through a 3D field. We introduce **DEPTH**, which ties each recursion step to one position in the sequence. The condition is encoded once as the initial state, a single shared block advances the state by one position per step, and each step is supervised against its own target. DEPTH thus sits between a UT and an RNN, pairing a transformer transition with a carried state and per-step outputs. We find that per-slice supervision is what gives each intermediate state its physical meaning, and physical coordinate must be supplied at every step. A UT given $1\times$ to $9\times$ internal recursion per slice needs $5\times$ DEPTH's per-slice compute to reach parity and collapses beyond that point. We evaluate on finite-element stress and warpage fields and a Laplace-equation benchmark.

1 INTRODUCTION

Weight-shared recursive transformers decouple multiple transformer blocks or layers from parameter count by applying a single block repeatedly. The UT Dehghani et al. (2019), ALBERT Lan et al. (2020), looped transformers Yang et al. (2024), and the Tiny Recursive Model (TRM) Research (2025) all follow this principle, spending *compute* rather than *parameters*. This is attractive wherever data is scarce and memory is tight Saha et al. (2025); Cioflan et al. (2024), a regime that most efficiency methods, built around long sequences and large models Katharopoulos et al. (2020); Dao et al. (2022); Bolya et al. (2023); Fedus et al. (2022), do not target. Existing recursive transformers, however, share one property: their recursion is typically *unguided*. Every step performs the same undifferentiated refinement toward the same target, and nothing distinguishes step 1 from step T.

Many prediction problems offer recursion a natural meaning. In *physically indexed sequence prediction*, a static condition determines an ordered sequence of outputs indexed by a physical coordinate. In this paper, a set of design parameters determines the stress field of a 3D structure, observed as successive cross-sections (or slices) along its physical coordinate. Such problems arise throughout simulation-based engineering, where each sample is costly and inputs are low-dimensional Fan et al. (2024); Li et al. (2023; 2021). Current architectures handle this structure poorly. A UT treats each slice as an independent sample, re-encodes the condition, and discards what it computed for the neighbouring slice. Coordinate-conditioned models Park et al. (2019); Mildenhall et al. (2020) do the same. An RNN carries state along the sequence, but fundamentally it does not use modern self-attention mechanism (Section 3.4).

We introduce **DEPTH**, a recursive transformer that ties each recursion step to one position in the physical sequence. The condition is encoded once as the initial state $H^{(0)}$. A single shared block then advances the state by one position per step, $H^{(z)} = f_\theta\big(H^{(z-1)}, S^{(z)}\big)$, where $S^{(z)}$ tracks the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

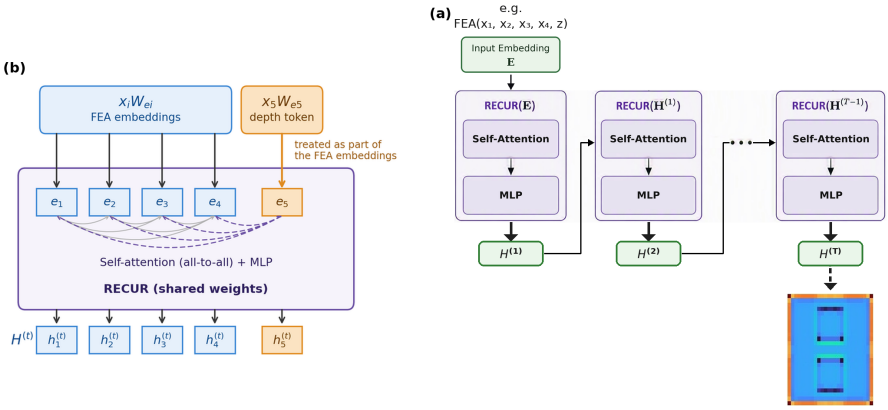


Figure 1: Universal Transformer (UT) for tabular-to-image prediction. (a) The tabular FEA inputs (x_1, x_2, x_3, x_4, z) are embedded as $\mathbf{E}$ and refined by T applications of a single weight-shared block, RECUR (self-attention followed by an MLP). The final state $H^{(T)}$ is projected by the final layer to a latent code, which a pretrained decoder (not shown) maps to the contour image. (b) Inside RECUR, the depth signal z enters as a fifth token x_5, treated as part of the FEA embeddings and processed jointly with them through self-attention.

physical coordinate. Each step is decoded and supervised against its own target. By contrast, a UT refines each slice separately, $H_z^{(0)} = \mathbf{E}(\mathbf{c}, z)$ and $H_z^{(t)} = f_\theta\big(H_z^{(t-1)}\big)$ for $t = 1, \ldots, T$, restarting for every slice, so a full sequence costs $Z \times T$ block applications against DEPTH's Z. Unlike deep supervision Lee et al. (2015), each step's target is distinct rather than shared. The full sequence is thus generated with one forward pass per position, and the recursion index becomes a physical coordinate rather than an iteration count.

This design raises four **questions** in the paper: (a) Where should the physical coordinate enter (Section 3.2)? (b) Is per-slice supervision necessary, and how does it shape the gradient through the recursion (Section 3.5)? (c) Can a UT close the gap simply by recursing more per slice (Section 3.3)? (d) How does DEPTH relate to an RNN (Section 3.4)?

Our **contributions** are as follows. (a) We formalize physically indexed sequence prediction (Section 2) and identify unguided recursion as the reason existing recursive transformers overspend on compute. (b) We propose DEPTH, which gives each recursion step a physical meaning, while within one shared block. (c) We compare injecting $S^{(z)}$ into the carried state versus as a separate token. Whether it enters the carried state or a separate token matters little, and we inject it into the state because this keeps the attention matrix smaller. (d) We sweep UT's internal recursion from $1\times$ to $9\times$ per slice. UT needs $5\times$ DEPTH's per-slice compute to reach parity and collapses beyond that point, while DEPTH matches this peak with a single pass per slice. (e) We compare two gates on DEPTH recursion: a momentum gate, which smooths the injected depth signal across slices, and a highway gate, which controls how much of the carried state is retained versus replaced by the block output at each slice. (f) We evaluate DEPTH against UT and TRM on finite-element stress and warpage fields and a Laplace-equation benchmark, reporting retrieval, parameters, and FLOPs. We will release the dataset upon publication/acceptance.

1.1 Related Work

DEPTH combines four properties: the condition is encoded once, a state is carried from one output to the next, each step is a physical coordinate with its own target, and the transition is a shared attention block. Several lines of work share some of these. *Recursive transformers* such as the UT (Dehghani et al., 2019) and the Tiny Recursive Model (TRM) (Research, 2025), along with ALBERT (Lan et al., 2020), looped transformers (Yang et al., 2024) and relaxed recursive transformers (Bae et al., 2024), use the same weight-shared block, but every step refines the *same* output toward the *same* target, and each position of a sequence is solved separately; in DEPTH each step is a *new* position

with its own target, and the state is carried between positions. *Recurrent transformers* such as Transformer-XL (Dai et al., 2019), the Block-Recurrent Transformer (Hutchins et al., 2022) and the Recurrent Memory Transformer (Bulatov et al., 2022) pair attention with a carried state, but use it to *read* a long input, consuming new tokens at every step, whereas DEPTH reads a static condition once and uses the state to *generate* the sequence, with the coordinate as its only per-step input. *Conditioned recurrent decoders* such as Show and Tell (Vinyals et al., 2015) also encode a condition once as the initial state, but they feed each output back as the next input, index steps by token position, and use a recurrent cell; Section 3.4 shows that replacing this cell with a tokenised attention block is what makes the carry work on raw design inputs. *Latent ODE* models (Chen et al., 2018; Rubanova et al., 2019) are the closest in structure, encoding a condition into an initial state, evolving it along a continuous coordinate and decoding it against a target at each observed point. DEPTH can be viewed as their discrete counterpart, with a shared attention block over one token per design parameter in place of a vector field over a single latent, and with the coordinate injected explicitly at every step at a fixed cost of one block application per slice and no ODE solver.

2 Problem statement: recursive prediction of physically indexed sequences

Recursive transformer refers to a standard transformer that uses an architecture configuration where it loops the same transformer block, RECUR, multiple times to process an input. It gains better accuracy by using more compute from running the loops, while keeping the parameter count fixed regardless of the number of loops, by reusing the same transformer block and its weights (i.e. weight sharing). The simplest RECUR is a one-block transformer consisting of self-attention and an MLP (Fig. 1b) and is independent of the configuration it is deployed in. The simplest **configuration** is a UT (Fig. 1a) that applies RECUR for T recursive loops: the input embedding E is passed through RECUR to give $H^{(1)}$, which is fed back into the same block to give $H^{(2)}$, and so on until $H^{(T)}$. The hidden states $H^{(1)}, \ldots, H^{(T-1)}$ are internal representations of UT with no physical meaning of their own. After T loops, $H^{(T)}$ is used to predict the output. We apply UT to tabular-to-image stress prediction as follows. The input tokens are the embeddings of the Finite Element Analysis (FEA) design parameters, together with a depth token specifying which slice is to be predicted (Fig. 1a). Each run of UT therefore predicts one slice of one design case, for example Case 4 at $z = 2$.

In advanced packaging thermo-mechanical simulation, we observe that for a fixed FEA design case, the contour image changes gradually from one layer to the next as we march through the depth z of the structure. The bottom of Fig. 2 shows the structure of the data. Each row is a design case, one of the combinations of the FEA design parameters, which together form the design condition $\mathbf{c}$. Each column is a slice at depth $z = 1, \ldots, Z$, and each slice is the contour image $\mathbf{y}^{(z)}$ of that case at that depth. The task is to predict an entire row from its design condition, $\mathbf{c} \mapsto \{\mathbf{y}^{(z)}\}_{z=1}^{Z}$: a sequence whose positions are physical locations, not time steps or token order. Throughout the paper, *depth* refers to this physical coordinate, never to the number of recursion steps.

UT, which applies the shared RECUR block for T loops is a valid recursive predictor, but it predicts a single slice rather than a row: its loop index t counts refinements of one answer, and being at loop 3 rather than loop 7 carries no physical meaning. The row structure is therefore imposed from outside the recursion, which causes four problems:

(a) **One depth token must separate all Z slices.** For a given design case, the four FEA tokens are identical across all Z slices, so a single depth token (Fig. 1b) is the only input that distinguishes them, and it must compete with the FEA tokens for attention at every loop. (b) **Slices in same row treated independently.** Each slice is predicted from scratch: after Case 4 at $z = 2$, UT restarts for $z = 3$, so the progression of stress along the row is never modelled directly. (c) **High recursion cost.** The loops refine a single slice. With T loops per slice, a full row costs $Z \times T$ applications of RECUR. (d) **Only the final loop is supervised.** Nothing constrains $H^{(1)}, \ldots, H^{(T-1)}$ to be physically meaningful.

We therefore seek a recursive predictor whose recursion steps *are* the physical positions, $t \equiv z$, in which (1) depth enters at every step t, (2) hidden state is carried from one slice to the next, and (3) every t step produces a supervised prediction.

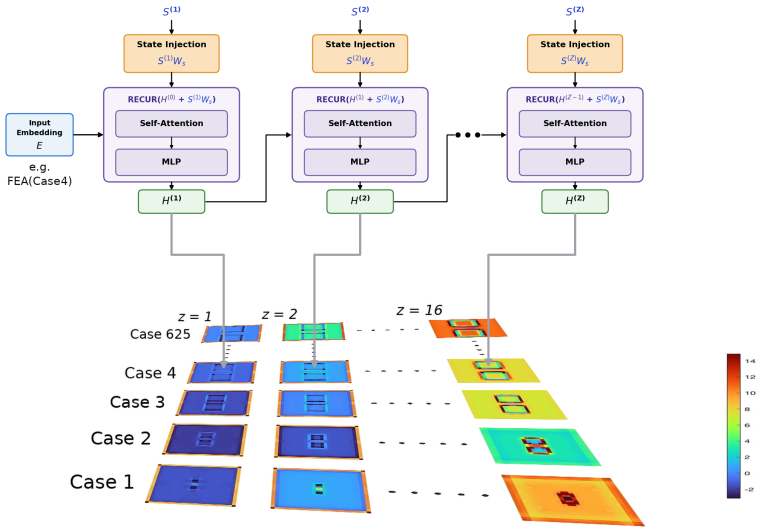


Figure 2: DEPTH architecture for Stress prediction **Bottom:** Entire sequence of stress contours for the same FEA Case ID. **Top:** The embedding E of a design case, e.g. FEA(Case 4), forms the initial state $H^{(0)}$. At recursion step z, the depth signal $S^{(z)}$ is injected into the carried state $H^{(z-1)}$, and the RECUR block maps the result to $H^{(z)}$, from which the contour slice at depth z is predicted. $H^{(z)}$ is then carried to step $z+1$, and the recursion proceeds for $z = 1, \ldots, Z$, producing one slice per step.

3 Proposed Recursive Model

3.1 DEPTH architecture

Motivated by the problems in Section 2, we re-purpose RECUR to have physical meaning, by associating each output with a contour image. We call this approach the Depth Recursive Model (DEPTH), shown in Fig. 2. Specifically, DEPTH replaces UT's recursion index t with the physical depth index z, so that $\mathbf{H}^{(z)}$ corresponds to the stress contour at depth slice $z = 1, \ldots, Z$, where Z is the total number of slices, all for the same design case, i.e. the same FEA parameters. At each step, the depth signal $\mathbf{S}^{(z)}$ enters through state injection, $\mathbf{S}^{(z)}\mathbf{W}_s$ in Eq. (1).

$$\mathbf{H}^{(0)} = \mathbf{E}, \qquad \mathbf{H}^{(z)} = \mathrm{RECUR}\big(\mathbf{H}^{(z-1)} + \mathbf{S}^{(z)}\mathbf{W}_s\big), \quad z = 1, \ldots, Z, \tag{1}$$

3.2 Where should depth go? State injection vs. token injection

DEPTH must supply the depth of each slice to the recursion at every step. Fig. 3 compares two ways. In *state injection*, used by DEPTH, the depth signal $S^{(z)}W_s$ is added to every row of the carried state, so each token enters RECUR already carrying depth: $\tilde{h}_i = h_i^{(z-1)} + S^{(z)}W_s$. In *token injection*, the carried state is left unchanged, $\tilde{h}_i = h_i^{(z-1)}$, and depth enters as a fresh token $\tilde{h}_5 = S^{(z)}W_{e5}$ at every step; the carried tokens then access depth only through self-attention. The output depth token $h_5^{(z)}$ contributes to the prediction at step z but is not carried forward, since it is replaced by the next depth. The two injection schemes share the carried state and one block application per slice, and differ only in *where* depth enters: into every token's state, or as one token that must be reached through attention. The latter is the mechanism behind the first problem of UT (Section 2), now with the carry in place.

All ablations in this section use Stress10K (Z=16; 500 training and 124 held-out test design cases; Appendix A.2). We report cross-modal retrieval (Recall@1, Recall@5, MAP; Section 4 and Appendix A.3) together with RMSE, SSIM and peak error of the decoded contours, as mean $\pm$ std over n runs.

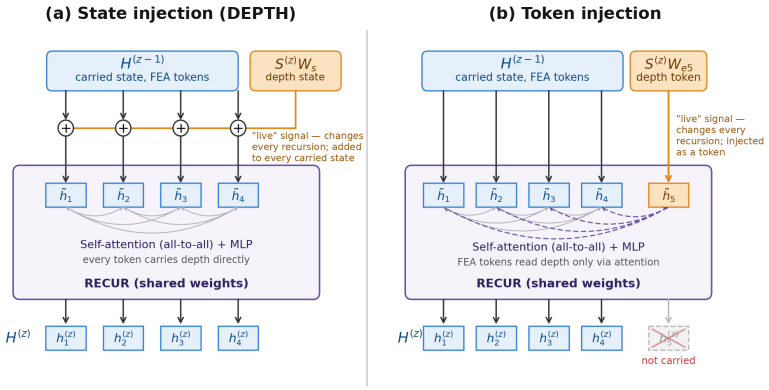


Figure 3: Two ways to supply depth in the forward pass. **(a) State injection (DEPTH):** $S^{(z)}W_s$ is added to every row of the carried state, giving $\tilde{h}_i = h_i^{(z-1)} + S^{(z)}W_s$, so each state carries depth directly. **(b) Token injection:** the carried states are unchanged, $\tilde{h}_i = h_i^{(z-1)}$, and depth enters RECUR as a "live" token $\tilde{h}_5 = S^{(z)}W_{e5}$; the carried states access depth only through self-attention. The output token $h_5^{(z)}$ contributes to the prediction at step z but is not carried to step $z{+}1$, where it is replaced by the next depth.

Table 1 compares the two injection sites with everything else held fixed: both variants carry the hidden state across slices, apply the block once per slice, and are trained with per-slice supervision. The two mechanisms perform comparably. State injection has slightly higher mean retrieval scores (Recall@1 of 0.48 vs. 0.43, MAP of 0.62 vs. 0.58), while token injection is marginally better on SSIM and peak error, and RMSE is effectively identical. Every difference lies within one standard deviation, so neither mechanism can be said to outperform the other. Token injection does show a wider spread on the retrieval metrics. The result indicates that what matters is that the depth signal is supplied at every step of the recursion, not the site at which it enters: even when depth must reach the other tokens through attention, the model keeps its predictions for different slices apart. We nevertheless adopt state injection in DEPTH on practical grounds. It adds depth without enlarging the token set, keeping four tokens instead of five and thus a smaller attention score matrix (4×4 vs. 5×5) at every slice.

Table 1: Token vs. state injection, with the carry and per-slice supervision fixed. Token injection had 4 runs and state injection had 6.

	R@1↑	R@5↑	MAP↑	RMSE↓	SSIM↑	Peak↓
Token	0.43±0.14	0.77±0.083	0.58±0.12	0.0133±0.0004	**0.97**±0.003	**0.054**±0.0035
State	**0.48**±0.077	**0.80**±0.063	**0.62**±0.068	**0.0132**±0.0018	0.96±0.0074	0.057±0.0086

3.3 Is a UT Enough? Recursion per Slice from UT 1x to 9x vs. DEPTH

DEPTH applies the block exactly once per slice: each of its z steps is a single forward pass, because moving along depth itself provides the sequential structure. UT has no such axis and instead refines each slice by applying the block T times. We sweep UT's per-slice recursion from $T = 1$ to $T = 9$ to measure how much repeated computation UT needs to approach DEPTH, and where that computation begins to destabilize training. All models use the same single weight-shared block, so parameter counts are comparable and only the number of block applications varies: $Z \times T$ for UT against Z for DEPTH.

Table 10 and Fig. 4 show three regimes. At matched compute (16 block applications), DEPTH clearly outperforms UT-1x, with a MAP of 0.62 against 0.25. Adding loops helps UT at first: UT-3x narrows the gap, and UT-5x matches DEPTH, reaching the same Recall@1 and MAP while spending $5\times$ the computation (80 block applications against 16). Beyond this point, more recursion hurts. UT-7x falls to a Recall@1 of 0.037, and UT-9x collapses to chance-level retrieval, with

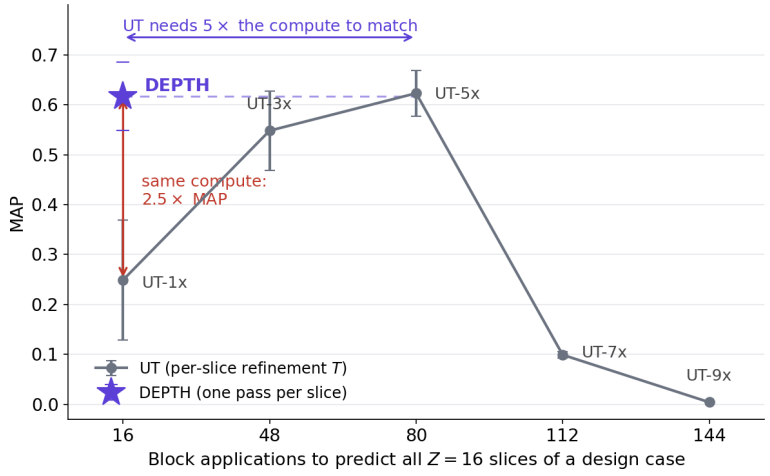


Figure 4: Retrieval accuracy (MAP) against compute, measured as block applications per design case. UT refines each slice with T passes, costing $Z \times T$ applications for all $Z = 16$ slices, whereas DEPTH applies the block once per slice. UT needs $5\times$ DEPTH's compute to match it and collapses at greater recursion. Mean $\pm$ std over 4-6 runs.

SSIM dropping to 0.54, so the extra loops degrade the predicted contours themselves, not only their ranking. UT's recursion therefore cannot be extended to buy more accuracy. DEPTH, in contrast, runs a 16-step recursion stably, because each step is a different slice with its own target rather than another refinement of the same one.

3.4 How DEPTH is related to RNN

An RNN updates a state vector as $h^{(z)} = \tanh\big(W_{hh}h^{(z-1)} + W_{hx}S^{(z)} + b_h\big)$ and predicts $\hat{y}^{(z)} = W_{hy}h^{(z)} + b_y$. DEPTH differs in its update: each design parameter is embedded as a token rather than stored as a scalar state, the tokens are mixed by a residual self-attention block that can be stacked, and the depth signal is injected into the state before each application of the block, $H^{(z)} = \mathrm{RECUR}\big(H^{(z-1)} + S^{(z)}W_s\big)$. We train the RNN with the same depth signal, per-slice loss and optimiser as DEPTH, so the two differ only in the form of the state and the update applied to it. We consider two initial states: the four raw design parameters ($h^{(0)} \in \mathbb{R}^4$), which matches DEPTH's input, and a degree-4 polynomial expansion of them ($h^{(0)} \in \mathbb{R}^{70}$), which supplies hand-engineered interaction terms.

Table 3 shows that a carried state and per-slice supervision are not enough on their own. Given only the raw design parameters, the RNN retrieves at chance: its state has only four units, one per parameter, and a single tanh layer driven by a scalar depth input lacks the capacity to form the nonlinear interactions between parameters that distinguish design cases. Embedding the parameters as tokens and mixing them with self-attention removes this limit. DEPTH-1b, which also receives only the four raw parameters, exceeds even an RNN given 70 hand-engineered polynomial features (0.498 vs. 0.430 Recall@1), learning the interactions that the expansion supplies by hand. Such features also scale poorly: a degree-4 expansion of n parameters has $\binom{n+4}{4}$ terms (1,001 for $n = 10$), whereas DEPTH adds one token per parameter. Stacking a second block raises Recall@1 to 0.614 for DEPTH-2b, 0.18 above the strongest RNN.

3.5 Per-slice supervision and the carry gradient

In DEPTH, a prediction and a target are attached to every depth slice z, not only the final one, and the loss sums over all slices, $\mathcal{L} = \sum_{z=1}^{Z} \mathcal{L}_z$ with $\mathcal{L}_z = \frac{1}{2}\|\mathbf{y}^{(z)} - \hat{\mathbf{y}}^{(z)}\|^2$, where $\hat{\mathbf{y}}^{(z)}$ is read from $H^{(z)}$ by the shared output layer. Each state $H^{(z)}$ affects the loss in two ways: directly, through its own slice loss $\mathcal{L}_z$, and indirectly, because it is the input to every deeper slice. Its gradient is therefore

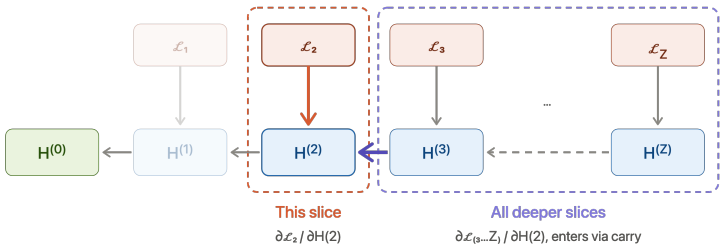


Figure 5: Gradient at slice $z = 2$ under per-slice supervision. The local term comes from the slice's own loss $\mathcal{L}_2$ (orange). The carry term collects all deeper losses $\mathcal{L}_3, \ldots, \mathcal{L}_Z$ (purple) and reaches $H^{(2)}$ through a single application of the block, $H^{(3)} \to H^{(2)}$. $\mathcal{L}_1$ does not depend on $H^{(2)}$ (faded); it enters one step later, at $H^{(1)}$.

the sum of two terms,

$$\frac{\partial \mathcal{L}}{\partial H^{(z)}} = \underbrace{\frac{\partial \mathcal{L}_z}{\partial H^{(z)}}}_{\substack{\text{local gradient} \\ \text{(this slice)}}} + \underbrace{\frac{\partial \mathcal{L}_{>z}}{\partial H^{(z)}}}_{\substack{\text{carry gradient} \\ \text{(all deeper slices)}}}, \qquad \mathcal{L}_{>z} = \sum_{k=z+1}^{Z} \mathcal{L}_k. \tag{2}$$

Fig. 5 illustrates this for slice $z = 2$. The local gradient comes from $\mathcal{L}_2$, the loss computed directly from $H^{(2)}$. The carry gradient collects the losses of all deeper slices, $\mathcal{L}_3, \ldots, \mathcal{L}_Z$. The shallower loss $\mathcal{L}_1$ does not appear, since it is computed before $H^{(2)}$ and cannot be affected by it; it enters one step later, at $H^{(1)}$. Repeating this from $z = Z$ down to $z = 1$, each state receives its own slice loss plus the carried losses of every slice beneath it. Under final-only supervision, as in UT, only $\mathcal{L}_Z$ remains in Fig. 5. The only local gradient is $\partial \mathcal{L}_Z / \partial H^{(Z)}$ at the final slice; every earlier state $H^{(z)}$ is trained solely by this gradient carried back through $Z - z$ applications of the block like a a single L-shaped path.

3.6 Gating mechanisms

DEPTH's recursion has two signal pathways at each step z: a *momentum* pathway, where the depth signal $S^{(z)}W_s$ is added to the carried state $H^{(z-1)}$ to indicate which slice the block is processing, and a *highway* pathway, where the block's output becomes the state carried to step $z+1$. Gating is a common way to stabilize deep and recursive networks, as in highway networks and depth-recurrent transformers (Srivastava et al., 2015; Jaber & Jaber, 2026). We test it at each pathway independently.

The *momentum-gate* smooths the depth signal with an exponential moving average,

$$v^{(z)} = \beta_v \, v^{(z-1)} + (1 - \beta_v) \, S^{(z)}W_s, \qquad v^{(0)} = 0, \tag{3}$$

and injects $v^{(z)}$ in place of $S^{(z)}W_s$, so each step receives an accumulation of past depth signals rather than the current one alone. The *highway-gate* interpolates the state transition,

$$H^{(z)} = \beta_h \, H^{(z-1)} + (1 - \beta_h) \, \mathrm{RECUR}\left(H^{(z-1)} + S^{(z)}W_s\right), \tag{4}$$

controlling how much of the carried state is retained versus replaced by the block's fresh output. We empirically tested the best values at $\beta_v = 0.3$ and $\beta_h = 0.88$. The two gates differ in both location and form: the momentum-gate smooths the conditioning signal over depth, while the Highway-gate interpolates the state itself.

Table 2 shows that the two locations behave differently. The Highway-gate gives a modest improvement on five of six metrics, raising Recall@1 from 0.48 to 0.52, and it also reduces the spread across runs, with the Recall@1 std falling from 0.077 to 0.028. The momentum-gate slightly degrades every metric. A plausible explanation is that averaging depth signals across slices blurs what distinguishes one slice from the next, the same information UT's single depth token struggles to carry

Table 2: Effect of gating location, with $n = 6$ runs per configuration. Best value per column in bold. Mean $\pm$ std over runs.

	R@1↑	R@5↑	MAP↑	RMSE↓	SSIM↑	Peak↓
Plain DEPTH	$0.48_{\pm 0.077}$	$0.80_{\pm 0.063}$	$0.62_{\pm 0.068}$	$\mathbf{0.013}_{\pm 0.0019}$	$0.96_{\pm 0.0070}$	$0.057_{\pm 0.0086}$
Momentum-gate	$0.45_{\pm 0.047}$	$0.77_{\pm 0.053}$	$0.59_{\pm 0.048}$	$0.014_{\pm 0.0012}$	$0.96_{\pm 0.0050}$	$0.061_{\pm 0.0036}$
Highway-gate	$\mathbf{0.52}_{\pm 0.028}$	$\mathbf{0.83}_{\pm 0.022}$	$\mathbf{0.66}_{\pm 0.024}$	$0.014_{\pm 0.00070}$	$\mathbf{0.97}_{\pm 0.0030}$	$\mathbf{0.056}_{\pm 0.0026}$

(Section 2), whereas retaining part of the carried state smooths the trajectory across depth without weakening the depth signal. The Highway-gate's gain is within about one standard deviation of plain DEPTH, so we treat it as an optional refinement rather than a core component.

4 EXPERIMENTS

We compare DEPTH 1 with recursive-transformer baselines 6 on three datasets A.2: Stress10K and Warpage10K, each with $Z{=}16$ depth slices per design case, and the PINN dataset, with $Z{=}15$. All models (their Acronym in Table 6) are evaluated on held-out design cases by cross-modal retrieval: the predicted stress latent of each slice is used to query a gallery of ground-truth slices, and we report Recall@1, Recall@5 and MAP, as mean $\pm$ std over independent runs. We use retrieval rather than pairwise errors such as MAE or RMSE because field maps within a dataset share the same geometry, so pairwise errors remain uniformly low across methods and fail to separate good predictions from poor ones (Lim & Puvvada, 2026). Retrieval instead requires each prediction to be distinguished from every other slice in the test gallery. Since each query has exactly one true match, MAP equals the mean reciprocal rank. All baselines predict one slice per run, so a design case costs Z independent runs. DEPTH instead predicts all Z slices of a case in a single call that recurses across depth. We therefore report two cost measures alongside the parameter count. **Blocks/case** counts the independent block (i.e. forward pass) runs needed per design case. **FLOPs/case** is the training cost (forward and backward pass) of predicting one design case, and it captures the recursion inside DEPTH's single call. Models are grouped into 1-block and 2-block tiers by backbone depth. vanilla-1b and trm-zy-1b (Section A.6) use LayerNorm; all other models tested for best run on LayerNorm, RMSNorm and no normalization.

All baselines condition on depth in the same way, by embedding z as a learned input token, and all are trained against the target of the slice they predict. They differ only in how they spend compute on each slice: vanilla applies its block once, UTkx applies it k times, and TRM recurses a latent and an answer state. None of them carries information from one slice to the next; each slice is predicted from scratch. TRM does carry a state, but across refinements of a single slice, and it restarts for every new slice. DEPTH is the only model whose state is carried across depth. The comparisons below therefore test whether marching a state through z is a better use of compute than refining each slice independently.

4.1 RESULTS

In the main comparisons we run UT at $3\times$ recursion per slice. UT peaks at $5\times$ on Stress10K, where it matches DEPTH at $5\times$ its compute, and collapses beyond that (Section 3.3); we use $3\times$ as a setting that is stable across datasets and tiers and refer to Section 3.3 for UT at its best.

Across the two datasets (third dataset in Section B.3), DEPTH achieves the best MAP, while having the fewest parameters and the lowest FLOPs per case in every tier. The baseline that comes closest in accuracy, trm-zy-1b, does so only at 27–34$\times$ DEPTH's training cost. The UT results also reproduce the pattern of Section 3.3: extra recursion per slice buys limited accuracy, and on PINN at 2 blocks makes results worse. This is consistent with applying the same transformation repeatedly without a target for the intermediate states, which yields diminishing returns and can accumulate error. DEPTH avoids this by tying each recursion step to a physical slice with its own target.

Stress10K. Table 3 (All models are evaluated on held-out designs: 500 training and 124 test cases for Stress10k and Warpage10k) shows that DEPTH gives the best Recall@1 and MAP in both tiers while being the cheapest model in each. In the 1-block tier, UT3x-1b reaches a MAP of 0.550 at

Table 3: Retrieval performance on Stress10K (Z=16). *Block/case* counts every block application needed to predict all Z slices of a design case (runs $\times$ blocks $\times$ recursion steps per run). The recurrent baselines use the same depth signal, per-slice supervision and optimiser as DEPTH, starting from the raw inputs $\mathbf{x}$ (1×4) or their degree-4 polynomial features $\phi_4(\mathbf{x})$ (1×70); DEPTH embeds each input as its own token of width d=24 (4×24).

Method	Params	Blocks/case	FLOPs/case	Recall@1	Recall@5	MAP
Recurrent						
RNN, $\mathbf{x}$	264	16	20,352	0.002 ± 0.002	0.009 ± 0.011	0.011 ± 0.011
RNN, $\phi_4(\mathbf{x})$	8,448	16	799,680	0.430 ± 0.011	0.760 ± 0.014	0.575 ± 0.012
1-block						
vanilla-1b	10,584	16	2,973,984	0.175 ± 0.019	0.466 ± 0.031	0.311 ± 0.021
UT3x-1b	10,488	48	7,527,744	0.390 ± 0.092	0.740 ± 0.064	0.550 ± 0.079
trm-z-1b	11,640	256	17,601,024	0.247 ± 0.040	0.557 ± 0.084	0.389 ± 0.056
trm-zy-1b	17,496	496	62,133,024	0.474 ± 0.005	$\mathbf{0.829 \pm 0.011}$	0.628 ± 0.007
DEPTH-1b	**9,336**	16	**2,293,056**	$\mathbf{0.498 \pm 0.042}$	0.806 ± 0.023	$\mathbf{0.633 \pm 0.033}$
2-block						
vanilla-2b	15,096	32	5,207,424	0.434 ± 0.042	0.792 ± 0.028	0.587 ± 0.036
UT3x-2b	15,096	96	14,488,704	0.583 ± 0.035	0.879 ± 0.022	0.713 ± 0.029
DEPTH-2b	**13,944**	32	**4,136,256**	$\mathbf{0.614 \pm 0.052}$	$\mathbf{0.901 \pm 0.018}$	$\mathbf{0.739 \pm 0.036}$

Table 4: Retrieval performance on the PINN dataset (Z=15). *Block/case* is defined as in Table 3. Bold marks the best value per column within each tier.

Method	Params	Blocks/case	FLOPs/case	Recall@1	Recall@5	MAP
1-block						
UT1x-1b	8,136	15	1,606,095	0.557 ± 0.029	0.983 ± 0.010	0.735 ± 0.026
UT3x-1b	8,136	45	4,178,925	0.717 ± 0.063	0.995 ± 0.004	0.841 ± 0.042
trm-z-1b	9,288	240	10,851,840	0.445 ± 0.054	0.911 ± 0.039	0.640 ± 0.045
trm-zy-1b	12,840	465	36,098,190	0.746 ± 0.080	$\mathbf{0.996 \pm 0.003}$	0.857 ± 0.048
DEPTH-1b	**6,984**	15	**1,074,708**	$\mathbf{0.753 \pm 0.068}$	$\mathbf{0.996 \pm 0.003}$	$\mathbf{0.862 \pm 0.040}$
2-block						
UT1x-2b	12,744	30	2,892,510	0.606 ± 0.062	0.987 ± 0.007	0.766 ± 0.044
UT3x-2b	12,744	90	8,038,170	0.627 ± 0.026	0.988 ± 0.005	0.782 ± 0.017
DEPTH-2b	**11,592**	30	**1,899,288**	$\mathbf{0.786 \pm 0.010}$	$\mathbf{0.998 \pm 0.002}$	$\mathbf{0.883 \pm 0.005}$

3.3$\times$ DEPTH-1b's FLOPs. trm-z-1b, which recurses only a latent state, stays well below UT3x-1b despite costing 2.3$\times$ as much. The strongest baseline, trm-zy-1b, comes within 0.005 MAP of DEPTH-1b and leads slightly on Recall@5, but it uses 1.9$\times$ the parameters and 27$\times$ the FLOPs. In the 2-block tier, DEPTH-2b leads on all three metrics, ahead of UT3x-2b, which costs 3.5$\times$ as much.

PINN. On the PINN dataset (Table 4; 384 training and 100 test cases, 1,500-slice gallery), Recall@5 is near saturation for most models, so Recall@1 and MAP are the informative metrics. In the 1-block tier, DEPTH-1b again matches or exceeds every baseline, with the fewest parameters and the lowest cost; trm-zy-1b is close in accuracy but needs 34$\times$ the FLOPs. The 2-block tier shows the clearest separation on any dataset: DEPTH-2b reaches a Recall@1 of 0.786, against 0.627 for UT3x-2b, and is also the most consistent across runs (MAP std 0.005). Adding recursion barely helps UT here: moving from UT1x-2b to UT3x-2b raises MAP by only 0.016, at 2.8$\times$ the cost.

5 Conclusion

We introduced DEPTH, a recursive transformer for physically indexed sequence prediction that encodes a static condition once and advances a carried state by one physical slice per step with a single shared attention block. Across stress, warpage and Laplace-equation benchmarks, DEPTH matches or exceeds recursive baselines with the fewest parameters and the lowest compute per design case. This advantage is consistent with the carried state: baselines that predict each slice from scratch perform worse however they spend compute per slice, a UT needs 5$\times$ DEPTH's per-slice compute to reach parity before collapsing, and TRM matches DEPTH only at 27–34$\times$ its cost. Our datasets are small and from a single domain, and comparisons with continuous-coordinate models such as latent ODEs remain future work.

REFERENCES

Mistral AI. Mixtral of experts. *arXiv*, 2024.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. *arXiv preprint arXiv:2410.20672*, 2024.

Daniel Bolya et al. Token merging: Your vit but faster. In *ICLR*, 2023.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer. In *Advances in Neural Information Processing Systems*, 2022.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Cristian Cioflan et al. Distributed inference with minimal off-chip traffic for transformers on low-power mcus. *arXiv preprint arXiv:2412.04372*, 2024.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Tri Dao. Flashattention-2: Faster attention with better parallelism. In *ICLR*, 2024.

Tri Dao et al. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*, 2022.

Mostafa Dehghani et al. Universal transformers. In *ICLR*, 2019.

Nan Du et al. Glam: Efficient scaling of language models with mixture-of-experts. In *ICML*, 2022.

Shaoze Fan, Haoshu Lu, Shun Zhang, Ningyuan Cao, Xin Zhang, and Jing Li. Graph-transformer-based surrogate model for accelerated converter circuit topology design. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024.

William Fedus et al. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. In *JMLR*, 2022.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *International Conference on Learning Representations (ICLR)*, 2024.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022.

Song Han et al. Learning both weights and connections for efficient neural networks. *NeurIPS*, 2015.

Geoffrey Hinton et al. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2015.

Edward Hu et al. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. In *Advances in Neural Information Processing Systems*, 2022.

Jaber Jaber and Osama Jaber. Ouroboros: Dynamic weight generation for recursive transformers via input-conditioned LoRA modulation. *arXiv preprint arXiv:2604.02051*, 2026.

Angelos Katharopoulos et al. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, 2020.

Arbaaz Khan and David A. Lowther. Physics informed neural networks for electromagnetic analysis. *IEEE Transactions on Magnetics*, pp. 1–1, 2022. doi: 10.1109/TMAG.2022.3161814.

Zhenzhong Lan et al. Albert: A lite bert for self-supervised learning of language representations. In *ICLR*, 2020.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 38 of *Proceedings of Machine Learning Research*, pp. 562–570, 2015.

Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*, 2023.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

Kart Leong Lim and Ji Lin. Deep clustering based boundary-decoder net for inter and intra layer stress prediction of heterogeneous integrated ic chip. In *2025 IEEE 75th Electronic Components and Technology Conference (ECTC)*, pp. 239–245. IEEE, 2025.

Kart Leong Lim and Thanmaya Bharadwaj Puvvada. Stressscore: Evaluating ai predicted stress contours in advanced packaging with contrastive learning. In *2026 IEEE 76th Electronic Components and Technology Conference (ECTC)*, pp. 1027–1035. IEEE, 2026.

Kart Leong Lim, Rahul Dutta, and Mihai Rotaru. Physics informed neural network using finite difference method. In *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1828–1833. IEEE, 2022.

Kart-Leong Lim, Rahul Dutta, and Mihai Rotaru. Inverse prediction of capacitor multiphysics dynamic parameters using deep generative model. In *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*, pp. 1825–1831. IEEE, 2023.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

James R Nagel et al. Solving the generalized poisson equation using the finite-difference method (fdm). *Lecture Notes, Dept. of Electrical and Computer Engineering, University of Utah*, 2011.

NVIDIA. Tensorrt-llm, 2023. GitHub Repository.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Yongming Rao et al. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, 2021.

Samsung SAIL Research. Tiny recursive models. *arXiv*, 2025.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Matthew NO Sadiku and Sudarshan R Nelatury. *Analytical Techniques in Electromagnetics*. CRC press, 2015.

Shaibal Saha et al. Vision transformers on the edge: A comprehensive survey of model compression and acceleration strategies. *arXiv preprint arXiv:2503.02891*, 2025.

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

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Liu Yang et al. Looped transformers are better at learning learning algorithms. In *ICLR*, 2024.

A EXPERIMENTAL SETUP

A.1 FEA PROBLEM

We represent the FEA design conditions as a structured token sequence $\mathbf{x} = [\text{token}_1, \text{token}_2, \cdots, \text{token}_L] \in \mathbb{R}^{L \times d_{\text{token}}}$, and the corresponding multi-slice FEA response as a three-dimensional map $\mathbf{I} = \{\mathbf{I}^{(1)}, \mathbf{I}^{(2)}, \ldots, \mathbf{I}^{(Z)}\} \in \mathbb{R}^{X \times Y \times Z}$ in Fig. 7. Our predictive framework in Fig. 6 decouples the task between latent prediction and image reconstruction, the former using a recursive transformer $B_{\text{pred}}(\cdot)$, and the latter handled by a pretrained decoder $A_{\text{dec}}(\cdot)$. This surrogate modeling framework operates through a four-stage sequential pipeline. First, a pretrained image encoder A_{enc} maps each cross-sectional field slice into a low-dimensional latent representation, formulated as $\mathbf{y}^{(z)} = A_{\text{enc}}\left(\mathbf{I}^{(z)}\right) \in \mathbb{R}^{1 \times d_{\text{latent}}}$. Second, the predictive transformer B_{pred} is trained to project the FEA design condition sequence $\mathbf{x}$ directly into this latent space, yielding the predicted latent trajectory $\hat{\mathbf{y}}^{(z)} = B_{\text{pred}}(\mathbf{x})$. Third, the network is optimized end-to-end by minimizing the mean squared error loss function between the ground-truth encoded latent and the predicted latent trajectory, defined as $\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\left\|\hat{\mathbf{y}}^{(n)} - \mathbf{y}^{(n)}\right\|_2^2$. Finally, the predicted transformer output trajectory is passed through the pretrained decoder A_{dec} to reconstruct the full high-fidelity thermo-mechanical field distribution, expressed as $\hat{\mathbf{I}}^{(z)} = A_{\text{dec}}\left(\hat{\mathbf{y}}^{(z)}\right)$. For the final output to be compatible with latent space, we train end-to-end a final projection layer with the recursive models (i.e. $\mathbf{H} \in \mathbb{R}^{d_{token} \times L}$) to obtain the latent output $\mathbf{y}^{(\text{final})} \in \mathbb{R}^{1 \times d_{latent}}$.

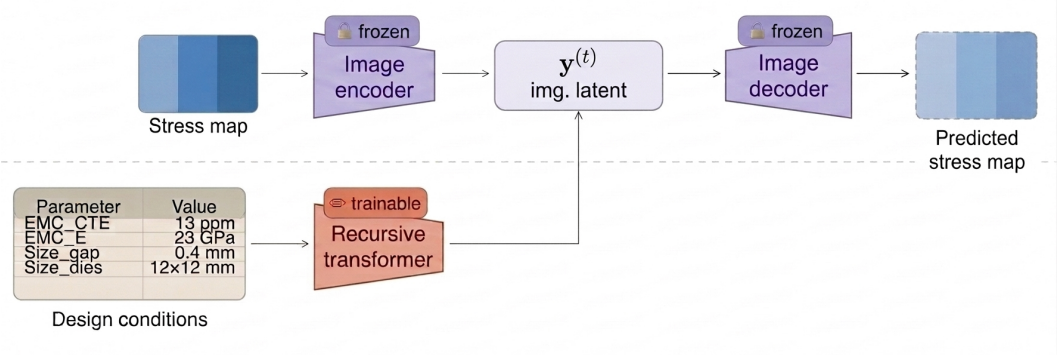


Figure 6: Predictive framework using Recursive transformer in latent space

A.2 DATASETS DESCRIPTION

We evaluate recursive models across three datasets with full factorial cases, $N = (factors)^{(levels)}$—the Stress10k and Warpage10k datasets ($N = 5^4 \times 16 = 10{,}000$ pairs) generated by ANSYS FEA simulation. Synthetic capacitor electrostatic field dataset generated by Successive-Over-Relaxation (SOR) numerical approach ($N = 22^2 \times 15 = 7{,}265$ pairs).

Stress10k, Warpage10k Lim & Lin (2025): Advanced packaging design requires understanding how material properties and geometric configurations influence thermo-mechanical behavior across spatial and depth-wise dimensions. Design variables –epoxy molding compound (EMC) coefficient of thermal expansion (CTE) and elastic modulus (E), die sizes ($Size_{dies}$), inter-die spacing (Gap_{dies}), and position within the package depth ($S^{(t)}$) –determine the resulting stress and warpage responses obtained from FEA.

$$\begin{aligned} \mathbf{x_{stress}} &= \left[\text{EMC}_{\text{CTE}},\ \text{EMC}_{E},\ \text{Size}_{\text{dies}},\ \text{Gap}_{\text{dies}},\ S^{(t)}\right] \in \mathbb{R}^{5\times 1}. \\ \mathbf{I_{stress}} &= \left\{\mathbf{I}^{(1)}, \mathbf{I}^{(2)}, \ldots, \mathbf{I}^{(16)}\right\} \in \mathbb{R}^{13\times 13\times 16} \end{aligned} \tag{5}$$

PINN Lim et al. (2023): The electrostatic field within an air-filled capacitor can be modeled by establishing collocation and boundary points across a two-dimensional grid. While the Laplace equation governs the internal collocation points, the system's boundary conditions define the perimeter. In this study, we implement five distinct boundary conditions (C1 through C5) in Lim et al. (2023)

governed by boundary parameters (a, b, d). Rather than treating all boundary metrics as static values, we allow the parameters to vary dynamically. Modifying these specific parameters significantly alters the resulting electrostatic field distribution, as demonstrated in Figure7.

$$\mathbf{x}_{PINN} = \left[\mathrm{a}, \mathrm{b}, d^{(t)}\right] \in \mathbb{R}^{3\times 1}.$$
$$\mathbf{I}_{PINN} = \left\{\mathbf{I}^{(1)}, \mathbf{I}^{(2)}, \ldots, \mathbf{I}^{(15)}\right\} \in \mathbb{R}^{13\times 13\times 15} \tag{6}$$

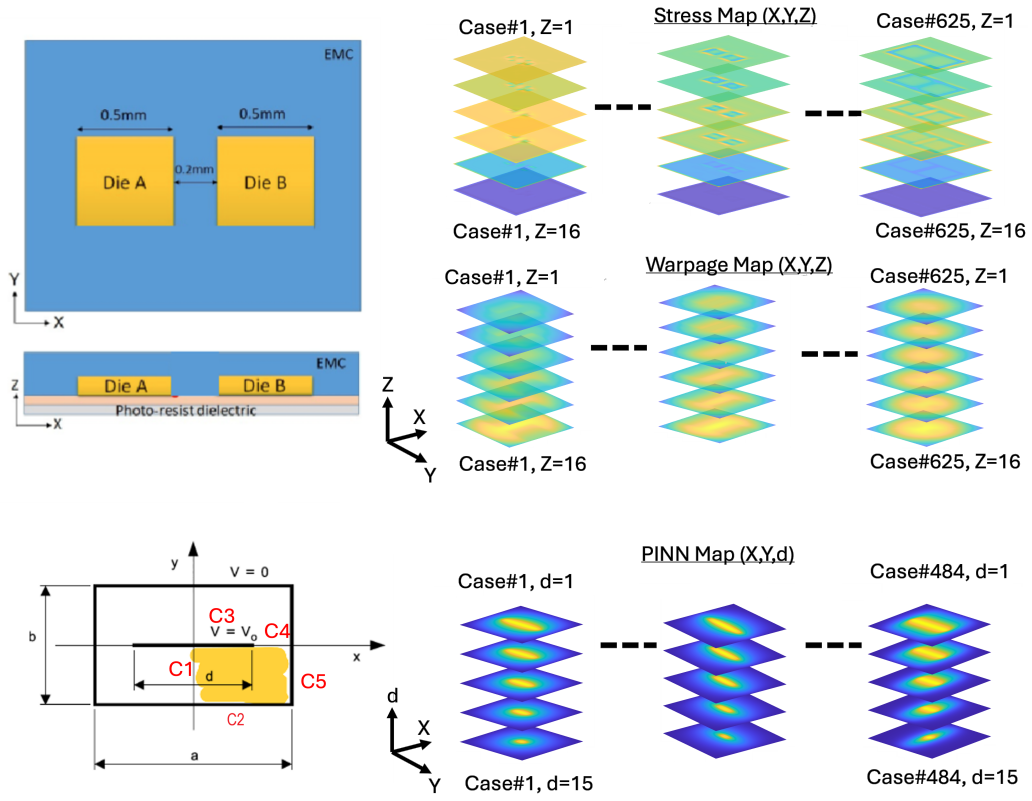


Figure 7: Datasets used in surrogate modeling. Top: Advanced Packaging Stress/ Warpage contours with varying FEA conditions. Bottom: Capacitor electrostatic fields with varying parameters in boundary conditions

A.2.1 STRESS10K/WARPAGE10K DATASET

In this paper, a two dies IC package representative geometry was adopted to demonstrate the methodology of using AI tool to facilitate AI prediction of stresses at different material interfaces inside the package. As shown in Figure 7, two identical rectangle dies were embedded in the package and different X-Y, Z-X views of the package structure are presented. Stress (MPa) occurs at the bond between different materials when subjected to thermal cycling (-55deg to 150deg). The X and Y axis refers to the plane while Z and X axis refers to the side view. Silicon (Si) dies are encapsulated by epoxy molding compound (EMC) with overmold on top of dies, i.e. dies are not exposed. Under the Si dies there is the underfill material layer. An RDL layer is at the bottom face of the package.

Train and test cases are generated based on a full factorial design with four variables, i.e. EMC modulus, EMC coefficient of thermal expansion (CTE), die size and gap size between the two dies. Table 5 lists the DOE of these four variables. 625 cases ($L = 5$, $P = 4$) were generated each layer for $Z = 16$ layers using ANSYS Mechanical for 10000 cases (8000 train and 2000 test). The stress result at each mesh point within the FEA model are exported together with the associated coordinates for each case. These results are subsequently used as the training and test data for AI stress predictor model development.

Table 5: DOE for each case using factorial design.

Parameters (P)	Range	Levels (L)
EMC Modulus [GPa]	5~30	5, 11, 17, 23, 30
EMC CTE [ppm/C]	5~20	5, 9, 12, 16, 20
Die Size [mm]	0.5~1.8	0.5, 0.8, 1.2, 1.5, 1.8
Gap between dies [mm]	0.2~1	0.2, 0.4, 0.6, 0.8, 1.0

We refer to the example ground truth stress images in Figure 8 using cases corresponding to different parameters (column) and different layer (row). In the top layer, there are more stress changes in the images as compared to the middle and bottom layer.

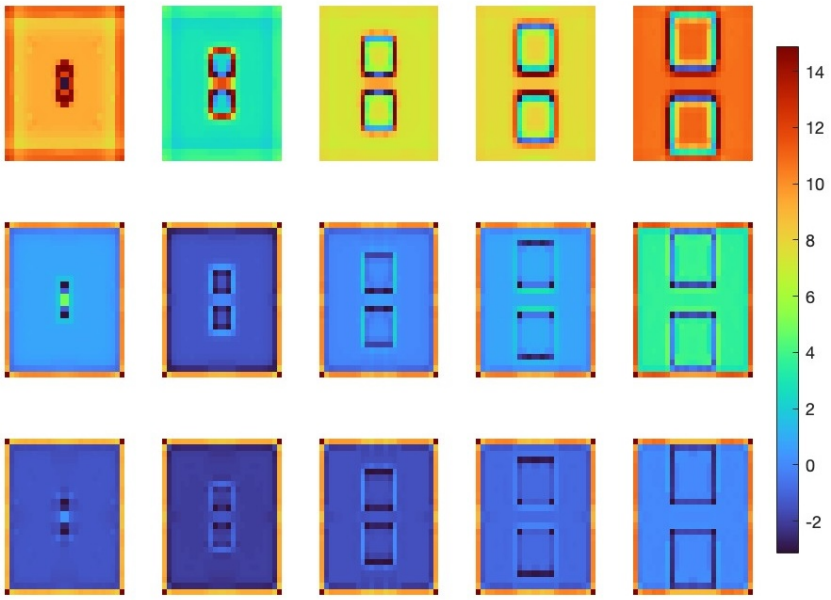
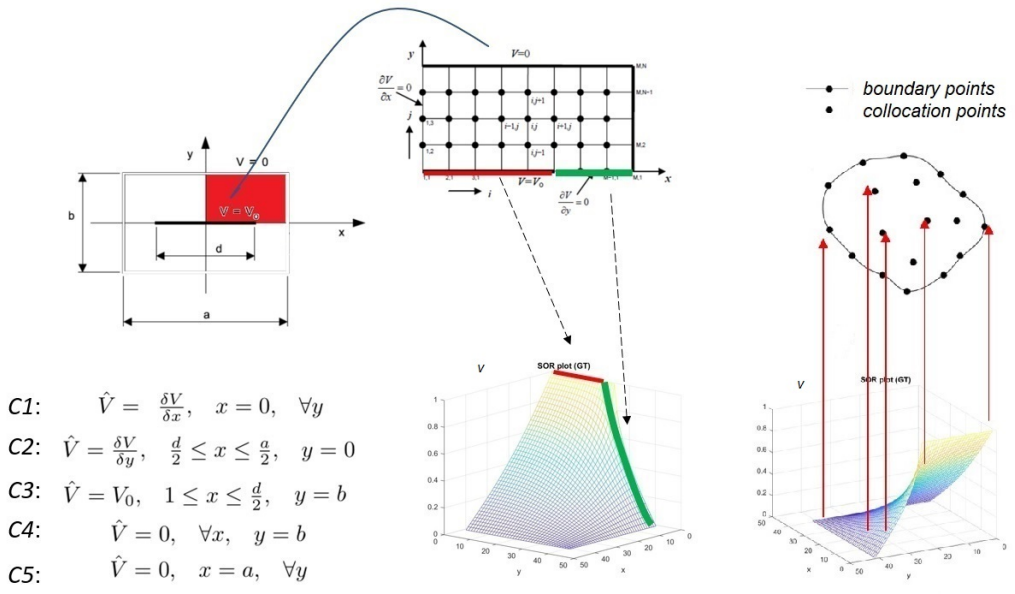


Figure 8: Stress images corresponding to different parameters (column) and at different layers (row).

A.2.2 CAPACITOR ELECTROSTATIC FIELD MODELING (PINN) DATASET

The electrostatic field of an air-filled capacitor Sadiku & Nelatury (2015) can be described using collocation points and boundary points on a 2D grid in Figure 9. The collocation points are described by Laplace equation and the boundary points by boundary conditions . We consider five boundary conditions $C1$ to $C5$ defined using parameters (a, b, d, V_o) along a 2D axis (x, y). Instead of traditionally fixing all the boundary parameters, we allow dynamic parameter a, b, d to vary, which changes the electrostatic field extensively as seen in Figure 7. In this work, we require a set of training data which comprises of electrostatic fields V and dynamic parameter a, b, d which can be obtained by commercial FEM softwares (e.g. ANSYS or MATLAB) or PDE solver (e.g SOR Nagel et al. (2011) or PINN Khan & Lowther (2022); Lim et al. (2022); Raissi et al. (2019)).



$$C1: \quad \hat{V} = \frac{\delta V}{\delta x}, \quad x = 0, \quad \forall y$$
$$C2: \quad \hat{V} = \frac{\delta V}{\delta y}, \quad \frac{d}{2} \leq x \leq \frac{a}{2}, \quad y = 0$$
$$C3: \quad \hat{V} = V_0, \quad 1 \leq x \leq \frac{d}{2}, \quad y = b$$
$$C4: \quad \hat{V} = 0, \quad \forall x, \quad y = b$$
$$C5: \quad \hat{V} = 0, \quad x = a, \quad \forall y$$

Figure 9: Modeling the boundaries of an electrostatic field (an air-filled capacitor) using five boundary conditions. In particular, a, b, d affects boundary conditions $C2$ and $C3$.

A.3 Evaluation Metric

To evaluate how well each predicted slice can be told apart from all other slices, we adopt a test-gallery retrieval protocol. Each predicted latent $\hat{\mathbf{y}}_i$, obtained from a design condition and depth, is used as a query and ranked against the ground-truth latents $\mathbf{y}_j$ of every slice in the test split, rather than compared against its own target alone. This is motivated by the observation that positive-pair similarity alone does not differentiate model quality: similarity scores between a prediction and its own target remain uniformly high across methods, regardless of true predictive performance (Lim & Puvvada, 2026).

Let $\mathbf{S} \in \mathbb{R}^{N \times N}$ be the cosine similarity matrix between the N predicted latents (rows, queries) and the N ground-truth latents in the test gallery (columns), with $\mathbf{S}_{ij} = \cos(\hat{\mathbf{y}}_i, \mathbf{y}_j)$. For query i, rank_i is the position of its true match $\mathbf{y}_i$ when row i is sorted in descending order. Since every query has exactly one true match in the gallery, rank_i is always defined.

Mean Average Precision (MAP) rewards models that place the correct match near the top of the ranking, with diminishing credit as rank increases. With a single relevant item per query, MAP reduces to the mean reciprocal rank:

$$\text{rank}_i = \begin{cases} k, & \text{if } \mathbf{S}_{i,k} \text{ is the } k\text{-th largest value in row } i \\ \infty, & \text{else} \end{cases} \tag{7}$$

$$\text{MAP} = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} \frac{1}{\text{rank}_i}, & \text{if rank}_i \text{ is found} \\ 0, & \text{else (not found)} \end{cases} \tag{8}$$

Recall@K reports the proportion of queries for which the true match is retrieved within the top-K candidates, providing a simpler, threshold-based measure of retrieval accuracy at varying levels of strictness. Together, these ranking-based metrics provide a more discriminative and deployment-relevant assessment of cross-modal alignment than positive-pair similarity alone, better reflecting the practical scenario in which a predicted stress contour must be correctly identified among many plausible candidates rather than validated against a single ground-truth reference.

$$\text{Recall@K} = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1, & \text{if rank}_i \le K\\ 0, & \text{if rank}_i > K\end{cases} \tag{9}$$

Reconstruction metrics. Retrieval measures whether predictions can be told apart; we also report how closely each decoded contour matches its ground truth. For test slice i, let $\hat{\mathbf{I}}_i = A_{\text{dec}}(\hat{\mathbf{y}}_i)$ be the decoded prediction and $\mathbf{I}_i$ the ground-truth contour, both 13×13 images with $P = 169$ pixels and values normalized to $[0, 1]$. We compute

$$\text{RMSE}_i = \sqrt{\frac{1}{P}\sum_{p=1}^{P}\big(\hat{\mathbf{I}}_{i,p} - \mathbf{I}_{i,p}\big)^2}, \qquad \text{Peak}_i = \max_p \big|\hat{\mathbf{I}}_{i,p} - \mathbf{I}_{i,p}\big|, \tag{10}$$

where RMSE measures average pixel error and Peak is the worst-case (L_∞) pixel error, which is sensitive to missed stress hotspots. SSIM_i is the structural similarity index (**?**) between $\hat{\mathbf{I}}_i$ and $\mathbf{I}_i$, which compares local luminance, contrast and structure and is therefore more sensitive to contour shape than pixel-wise errors. Each metric is averaged over all test slices in a run, and tables report the mean $\pm$ std of these averages across runs.

Because stress contours are dominated by smooth background regions with small localized features, pixel-wise metrics remain uniformly low across methods and separate them poorly; we therefore treat retrieval as the primary metric and reconstruction metrics as supporting evidence.

A.4 Baselines

Table 6: Method names used in Table 3, 4, 11.

Name	Description
RNN, $\mathbf{x}$ / $\phi_4(\mathbf{x})$	Recurrent baseline with the same depth signal and per-slice supervision as DEPTH, initialised from the raw inputs (1×4) or their degree-4 polynomial features (1×70).
vanilla	Standard (non-recursive) transformer; depth is given as an input token and each slice is predicted independently, with no state carried between slices.
UTkx	Universal Transformer baseline: the shared block is applied k times per slice ($k \in \{1, 3\}$), with depth given to the model as a token.
trm-z	Tiny Recursive Model-inspired baseline that recurses on a single latent state z.
trm-zy	TRM-inspired baseline with a latent state z and an answer state y; z is updated several times, then y once, per recursion.
DEPTH	Ours: one pass per slice, with the physical depth coordinate injected into the carried state and a supervised output at every slice.
-1b / -2b	Number of transformer blocks in the shared recursive unit (one or two).

A.5 Position encoding in UT

In the original UT, a fixed sinusoidal coordinate embedding $P^{(t)}$ is added to the state before every application of the shared block, $H^{(t)} = \text{RECUR}\big(H^{(t-1)} + P^{(t)}\big)$. It combines a position term, which identifies each token, and a timestep term, which tells the block which refinement loop t it is on. DEPTH's update, $H^{(z)} = \text{RECUR}\big(H^{(z-1)} + S^{(z)}W_s\big)$, has the same form but a different meaning. In UT, every step refines the same slice toward the same target, and $P^{(t)}$ only marks progress through those refinements. In DEPTH, every step is a new slice with its own target, and the learned signal $S^{(z)}W_s$ gives the physical depth that the carried state has reached.

A.6 Tiny Recursive Model-inspired recursion

Because the full TRM training procedure is tailored to iterative answer refinement on discrete reasoning tasks, and because all models in this work are implemented with hand-derived gradients, we implement a simplified variant, which we refer to as *TRM-inspired recursion*. It differs from the original in three respects:

1. **No gradient through the latent updates.** The original TRM backpropagates through the entire final recursion cycle, including all n latent updates of z. In our variant, gradients reach the shared block only through the final update of y; the latent updates of z are treated as constants during backpropagation. This substantially simplifies the hand-derived backward pass, but means z is not optimized directly.

2. **No deep supervision.** The original TRM repeats the full recursion over up to N_{sup} improvement steps, computing a loss at each step and carrying the detached (y, z) forward. We apply the loss once per forward pass, which keeps the training compute comparable to the other models in Table 11.

3. **No adaptive computation time (ACT).** TRM's halting head decides when to stop iterating over deep-supervision steps. Since our variant has no deep-supervision steps, ACT has nothing to act on and is omitted.

These simplifications remove mechanisms that likely contribute to TRM's performance, so our results should be read as characterizing a TRM-style recursion under a matched training budget, not as an evaluation of the full TRM training procedure.

Our trm-zy-1b uses RECUR to cap the parameters (weight sharing) while perform puzzle solving on Sudoku and ARC-AGI tasks. Within each iteration, RECUR_Z updates the latent feature $\mathbf{z}$ by feeding the input embeddings $\mathbf{E} \in \mathbb{R}^{L \times d_{token}} = [\mathbf{e}_1, \ldots, \mathbf{e}_L]$–where L is number of tokens and d_{token} is the embedding dimension–along with the previous latent feature $\mathbf{z}^{(n-1)}$, and the previous output feature $\mathbf{y}^{(n-1)}$. The updated latent feature $\mathbf{z}^{(n)}$ is then passed to RECUR_Y, which updates the output representation $\mathbf{y}^{(n)}$ while masking the input embeddings with zeros. Repeating this coupled update for N times defines the LATENT operation in Eqn.11. This progressively refine both $\mathbf{z}^{(n)}$ and $\mathbf{y}^{(n)}$. trm-z-1b is similar except it does not compute RECUR_Y.

$$
\begin{aligned}
\mathrm{LATENT} &\triangleq \mathrm{RECUR}_Y \circ (\mathrm{RECUR}_Z)^N \\
\mathbf{z}^{(n)} &= \mathrm{RECUR}_{\mathrm{Z}}\Big([\mathbf{e}_1, \ldots, \mathbf{e}_L, \mathbf{z}^{(n-1)}, \mathbf{y}^{(n-1)}]\Big), \\
\mathbf{y}^{(n)} &= \mathrm{RECUR}_{\mathrm{Y}}\Big([\mathbf{0}, \ldots, \mathbf{0}, \mathbf{z}^{(n)}, \mathbf{y}^{(n-1)}]\Big).
\end{aligned}
\tag{11}
$$

Finally to compute TRM (Eqn.12), LATENT is executed for T times and fed into another RECUR. Increasing either the inner N or the outer T loops increases the computation cost (FLOPs) without increasing the parameters through weight sharing.

$$
\begin{aligned}
&\mathrm{TRM}\big(\mathbf{E}, \mathbf{z}^{ini}, \mathbf{y}^{ini}\big) \triangleq \mathrm{RECUR} \circ (\mathrm{LATENT})^T \\
&= \mathrm{RECUR}\Big(\mathbf{E}, \Big[\big(\mathrm{RECUR}_Y(\mathbf{0}, \mathrm{RECUR}_Z(\mathbf{E}, \mathbf{z}, \mathbf{y}), \mathbf{y})\big)^N\Big]^T\Big).
\end{aligned}
\tag{12}
$$

A.7 HYPERPARAMETERS

Table 7: DEPTH hyperparameters on Stress10K/Warpage10k. All baselines are trained with the same optimizer, learning rate, batch size, iteration count and clipping.

Hyperparameter	**Value**
Data	
Design tokens L	4 (EMC CTE, EMC modulus, die size, die gap)
Slices per case Z	16, depth coordinate $s_z \in [0, 1]$
Train / test design cases	500 / 124
Latent dimension d_{latent}	48 (frozen pretrained autoencoder)
Architecture	
Token width d_{token}	24
Attention width d_{att}	16 (single head)
MLP hidden width	64, ReLU
Blocks in shared unit	1 (DEPTH-1b) or 2 (DEPTH-2b)
Skip scale α	1
Normalization	None
Biases	None
Output layer	Linear $W_f \in \mathbb{R}^{Ld_{\text{token}} \times d_{\text{latent}}}$, ReLU
Parameters	9,336 (1b) / 13,944 (2b)
Training	
Loss	$\frac{1}{2}\sum_{z=1}^{Z}\|\hat{\mathbf{y}}^{(z)} - \mathbf{y}^{(z)}\|^2$
Backpropagation	Full BPTT over all Z slices
Optimizer	Adam (β_1=0.9, β_2=0.999, ϵ=10^{-8})
Learning rate	10^{-3}
Batch size	4 design cases
Training iterations	152,000 (≈1,216 epochs)
Gradient clipping	0.1
Weight initialization	Xavier
Gating variants (Sec. 3.6)	
Momentum gate β_v	0.3
Highway gate β_h	0.88

Table 8: DEPTH hyperparameters on the PINN dataset. All baselines are trained with the same optimizer, learning rate, batch size, iteration count and clipping.

Hyperparameter	**Value**
Data	
Design tokens L	2 (boundary parameters a, b)
Slices per case Z	15, coordinate d normalized to $[0, 1]$
Train / test design cases	384 / 100
Latent dimension d_{latent}	48 (frozen pretrained autoencoder)
Architecture	
Token width d_{token}	24
Attention width d_{att}	16 (single head)
MLP hidden width	64, ReLU
Blocks in shared unit	1 (DEPTH-1b) or 2 (DEPTH-2b)
Skip scale α	1
Normalization	None
Biases	None
Output layer	Linear $W_f \in \mathbb{R}^{Ld_{\text{token}} \times d_{\text{latent}}}$, ReLU
Parameters	6,984 (1b) / 11,592 (2b)
Training	
Loss	$\frac{1}{2}\sum_{z=1}^{Z}\|\hat{\mathbf{y}}^{(z)} - \mathbf{y}^{(z)}\|^2$
Backpropagation	Full BPTT over all Z slices
Optimizer	Adam (β_1=0.9, β_2=0.999, ϵ=10^{-8})
Learning rate	10^{-3}
Batch size	4 design cases
Training iterations	152,000 ($\approx$1,583 epochs)
Gradient clipping	0.1
Weight initialization	Xavier

A.8 DEPTH FORWARD AND BACKWARD PASS

Algorithm 1 DEPTH (one block) forward pass and full BPTT over z-slices. ♦ per-slice supervision; ★ gradient carry across slices. $\mathrm{vec}(\cdot)$ is column-major flattening, $\mathrm{unvec}(\cdot)$ its inverse to $n \times d$, $\mathbf{1} \in \mathbb{R}^n$ is the all-ones vector and $\mathbb{I}[\cdot]$ the indicator.

Require: design tokens $x_{1:4}$; slice coordinates $s_{1:T} \in [0,1]$; targets $y_{1:T}$; shared weights $W_Q, W_K, W_V, W_o, W_1, W_2, W_f$; embeddings $W_{e,1:4}$; injection w_s; skip scale α
Loss: $L = \frac{1}{2}\sum_{t=1}^{T} \|\hat{y}_t - y_t\|^2$

Forward

$H_1 \leftarrow [\, x_1 W_{e,1};\, \ldots;\, x_4 W_{e,4} \,]$ ▷ design state h_0
for $t = 1$ **to** T **do**
  $E_t \leftarrow H_t + s_t\, \mathbf{1} w_s$ ▷ state-injection
  $Q_t \leftarrow E_t W_Q,\;\; K_t \leftarrow E_t W_K,\;\; V_t \leftarrow E_t W_V$
  $A_t \leftarrow \mathrm{softmax}_{\mathrm{row}}\big(Q_t K_t^\top / \sqrt{d}\big),\;\; Z_t \leftarrow A_t V_t$
  $Y_t \leftarrow Z_t W_o + \alpha E_t$ ▷ skip 1
  $P_t \leftarrow Y_t W_1,\;\; M_t \leftarrow \mathrm{ReLU}(P_t)$
  $H_{t+1} \leftarrow M_t W_2 + Y_t$ ▷ skip 2
  $u_t \leftarrow \mathrm{vec}(H_{t+1})^\top W_f,\;\; \hat{y}_t \leftarrow \mathrm{ReLU}(u_t)$ ▷ ♦ slice-t output

Backward (BPTT)

$\Delta W \leftarrow 0$ for all weights; $\;carry \leftarrow 0$
for $t = T$ **downto** 1 **do**
  $g_t \leftarrow (\hat{y}_t - y_t) \odot \mathbb{I}[u_t > 0]$ ▷ ♦ slice-t error
  $\Delta W_f \mathrel{+}= \mathrm{vec}(H_{t+1})\, g_t$
  $G \leftarrow \mathrm{unvec}(g_t W_f^\top) + carry$ ▷ ★ carry in
  $\Delta W_2 \mathrel{+}= M_t^\top G,\;\; \delta P \leftarrow (G W_2^\top) \odot \mathbb{I}[P_t > 0]$
  $\Delta W_1 \mathrel{+}= Y_t^\top \delta P,\;\; \delta Y \leftarrow G + \delta P\, W_1^\top$
  $\Delta W_o \mathrel{+}= Z_t^\top \delta Y,\;\; \delta Z \leftarrow \delta Y\, W_o^\top$
  $\delta V \leftarrow A_t^\top \delta Z,\;\; \delta A \leftarrow \delta Z\, V_t^\top$
  $\delta S \leftarrow A_t \odot \big(\delta A - \mathrm{rowsum}(\delta A \odot A_t)\big)/\sqrt{d}$ ▷ softmax
  $\delta Q \leftarrow \delta S\, K_t,\;\; \delta K \leftarrow \delta S^\top Q_t$
  $\Delta W_Q \mathrel{+}= E_t^\top \delta Q,\;\; \Delta W_K \mathrel{+}= E_t^\top \delta K,\;\; \Delta W_V \mathrel{+}= E_t^\top \delta V$
  $\delta E \leftarrow \delta Q\, W_Q^\top + \delta K\, W_K^\top + \delta V\, W_V^\top + \alpha\, \delta Y$
  $\Delta w_s \mathrel{+}= s_t\, \mathbf{1}^\top \delta E$
  $carry \leftarrow \delta E$ ▷ ★ carry out, $\partial L / \partial H_t$
$\Delta W_{e,k} \leftarrow x_k^\top\, carry_{k,:}$ for $k = 1, \ldots, 4$ ▷ all slices reach h_0
$W \leftarrow W - \eta\, \Delta W$

B ADDITIONAL RESULTS

B.1 EFFECT OF PER-SLICE SUPERVISION

Table 9 isolates the effect of per-slice supervision, holding state injection and the carry fixed. Supervising only the final slice collapses the model. Recall@1 falls from 0.48 to 0.0020, close to chance, and SSIM falls from 0.96 to 0.14. The failure is not confined to ranking; the reconstructed contours themselves degrade. This follows from Eq. (2). Without a local gradient at each step, slice z learns only from whatever gradient survives $Z - z$ applications of the block, and the depth signal injected at every step is never tied directly to a target. State injection supplies the depth, but only per-slice supervision teaches the recursion what each depth should produce.

Table 9: Final-slice vs. per-slice supervision, with state injection and the carry fixed. Mean ± std over 5 runs.

	R@1↑	R@5↑	MAP↑	RMSE↓	SSIM↑	Peak↓
Final-slice	0.0020±0.00071	0.0073±0.0018	0.0085±0.0016	0.35±0.092	0.14±0.052	0.62±0.0060
Per-slice	**0.48**±0.077	**0.80**±0.063	**0.62**±0.068	**0.013**±0.0018	**0.96**±0.0074	**0.057**±0.0086

B.2 UT PER-SLICE RECURSION SWEEP

Table 10: UT per-slice recursion sweep, with DEPTH. All models share one weight-shared block with identical parameter counts; only the number of block applications per design case varies. Best value per column in bold. Mean ± std over 5 runs. These are different run sets from Table3

	Blocks	R@1↑	R@5↑	MAP↑	RMSE↓	SSIM↑	Peak↓
UT-1x	16	0.13±0.089	0.36±0.17	0.25±0.12	0.046±0.017	0.88±0.045	0.12±0.027
UT-3x	48	0.39±0.092	0.74±0.064	0.55±0.079	0.018±0.0033	0.96±0.013	0.063±0.013
UT-5x	80	**0.48**±0.043	**0.81**±0.048	**0.62**±0.046	0.015±0.0053	**0.97**±0.0040	**0.051**±0.0060
UT-7x	112	0.037±0.0050	0.14±0.0080	0.099±0.0070	0.073±0.027	0.84±0.0080	0.16±0.014
UT-9x	144	0.00050±0	0.0025±0	0.0041±0	0.16±0.00020	0.54±0.0070	0.30±0.00010
DEPTH	16	**0.48**±0.077	0.80±0.063	**0.62**±0.068	**0.013**±0.0018	0.96±0.0074	0.057±0.0086

B.3 ADDITIONAL EXPERIMENTS - WARPAGE10K

Table 11: Retrieval performance on Warpage10K (Z=16). *Block/case* is defined as in Table 3. trm-zy-1b is averaged over the 7 of 24 runs that reached MAP > 0.60, selected on test MAP, so its values are an optimistic upper bound; all other models are averaged over all runs. Bold marks the best value per column within each tier.

Method	Params	Blocks/case	FLOPs/case	Recall@1	Recall@5	MAP
1-block						
vanilla-1b	10,584	16	2,973,984	0.233 ± 0.106	0.424 ± 0.147	0.328 ± 0.121
UT3x-1b	10,488	48	7,527,744	$\mathbf{0.629 \pm 0.064}$	$\mathbf{0.966 \pm 0.021}$	$\mathbf{0.772 \pm 0.048}$
trm-z-1b	11,640	256	17,601,024	0.526 ± 0.151	0.848 ± 0.113	0.667 ± 0.270
trm-zy-1b	17,496	496	62,133,024	0.532 ± 0.198	0.784 ± 0.183	0.647 ± 0.187
DEPTH-1b	**9,336**	16	**2,293,056**	0.607 ± 0.117	0.950 ± 0.034	0.753 ± 0.081
2-block						
UT1x-2b	15,096	32	5,207,424	0.537 ± 0.110	0.942 ± 0.016	0.708 ± 0.072
UT3x-2b	15,096	96	14,488,704	0.508 ± 0.287	0.871 ± 0.122	0.664 ± 0.217
DEPTH-2b	**13,944**	32	**4,136,256**	$\mathbf{0.652 \pm 0.097}$	$\mathbf{0.959 \pm 0.027}$	$\mathbf{0.783 \pm 0.067}$

Across the three datasets, DEPTH achieves the best MAP in five of the six tiers, the exception being Warpage10K (Section B.3) at 1 block, while having the fewest parameters and the lowest FLOPs per case in every tier. The baseline that comes closest in accuracy, trm-zy-1b, does so only at 27–34× DEPTH's training cost. The UT results also reproduce the pattern of Section 3.3: extra recursion per

slice buys limited accuracy, and on PINN and Warpage10K at 2 blocks it buys almost none or makes results worse. This is consistent with applying the same transformation repeatedly without a target for the intermediate states, which yields diminishing returns and can accumulate error. DEPTH avoids this by tying each recursion step to a physical slice with its own target.

Warpage10K. Warpage10K (Table 11) is the one dataset where a baseline leads in the 1-block tier. UT3x-1b is ahead of DEPTH-1b on all three metrics, by 0.019 MAP, although the gap is well within one standard deviation and UT3x-1b uses $3.3\times$ the FLOPs. In the 2-block tier, DEPTH-2b is clearly best, with a MAP of 0.810 against 0.708 for UT1x-2b. Here too, more recursion does not help UT: UT3x-2b is both less accurate and far less stable than UT1x-2b (Recall@1 std 0.287). Two reporting details apply to this table. trm-zy-1b converged 7 of 24 runs with average MAP 0.64 (due to TRM's difficulty with initialization), and DEPTH-2b over the 7 of 9 runs with MAP ≥ 0.7. Over all 9 runs, DEPTH-2b obtains 0.652 ± 0.097 Recall@1, 0.959 ± 0.027 Recall@5 and 0.783 ± 0.067 MAP, which still exceeds every other 2-block model.

C Extended related work

C.1 Survey on transformer compute reduction

Transformer compute reduction techniques and can be broadly categorized into seven paradigms. These include **Efficient Attention** Katharopoulos et al. (2020); Dao et al. (2022), **Token Reduction** Rao et al. (2021); Bolya et al. (2023), **Conditional Computation** Fedus et al. (2022); Du et al. (2022), **Architectural Optimization** Dehghani et al. (2019); Lan et al. (2020); Yang et al. (2024); Research (2025), **Model Compression** Han et al. (2015); Hinton et al. (2015); Hu et al. (2022), **Hardware Optimization** Dao (2024); NVIDIA (2023), and **State Space Models** Gu et al. (2022); Gu & Dao (2024). Although these approaches pursue the common objective of improving computational efficiency, they address different computational bottlenecks, ranging from reducing attention complexity and sequence length to redesigning network architectures or replacing attention-based models altogether. Among these paradigms, this work focuses on **Architectural Optimization**, where computational efficiency is achieved by redesigning the transformer itself. In particular, recursive weight sharing enables deeper computation without a proportional increase in parameters Dehghani et al. (2019); Yang et al. (2024); Research (2025), making it particularly attractive for resource-constrained EDA applications. Although numerous approaches have been proposed to reduce transformer computation, their suitability depends on the target application. Efficient attention mechanisms primarily benefit long-sequence models where self-attention dominates the computational cost Katharopoulos et al. (2020); Dao et al. (2022), while token reduction techniques rely on large token sets to achieve meaningful computational savings Rao et al. (2021); Bolya et al. (2023). Conditional computation and Mixture-of-Experts architectures improve scalability for large foundation models but introduce additional routing complexity and memory overhead Fedus et al. (2022); Du et al. (2022); AI (2024). Model compression and hardware-specific optimizations are generally applied after model design or target specific execution platforms Han et al. (2015); Hu et al. (2022); Dao (2024). In contrast, our application involves compact transformer surrogates operating on short input sequences under strict memory and computational constraints. Consequently, Architectural Optimization provides the most suitable design strategy, enabling computation to be reduced intrinsically through recursive weight sharing while preserving a compact, hardware-friendly architecture Dehghani et al. (2019); Yang et al. (2024); Research (2025).

AI USAGE DISCLOSURE

In this work, we used generative AI tools to provide feedback on experimental design and ablations, to assist in interpreting results, and to help debug the MATLAB implementation of the models. We have not used generative AI tools to generate synthetic data, formulate mathematical claims, or write proofs, and the remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to edit the paper for clarity and consistency, draft parts of the text, format tables, and check the manuscript for errors. All model implementations, experiments and reported numbers were produced and verified by the author; AI-suggested edits and code changes were reviewed and tested before use. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.