# Assignment 4: Transformers vs RNNs

**CS 312 — Ungraded · Due: October 29, 1pm (quiz time)**

**Basic setting.** Unless a problem states otherwise, use the course next-token prediction setup on the provided DCLM-style data. Compare models under the same data, token budget, and evaluation protocol. Lower validation loss is better.

**Instructions.** Submit a single PDF report to Gradescope by the deadline above, with a maximum of 20 pages. This assignment is ungraded, but you may cite your report for partial credit on the quiz.

For each problem, include an analysis of the supplied experiments and any new experiments you ran, followed by a short, approximately one-paragraph synthesis of your interpretation. Report the comparisons, hyperparameters, best validation losses, and diagnostic evidence. Give plots clear axes and labels. Tune central hyperparameters for new variants and record divergent runs as well as successful ones.

**Code and provided runs.** Starter files live in `experiments/a4_stability/` in the course repository:

| Problem | Starter file | Suggested run counts |
|---|---|---|
| Problem 1 | `p1_transformer_depth.py` | Mostly diagnostics |
| Problem 2 | `p2_rnn_depth.py` | Mostly diagnostics |
| Problem 3 | `p3_attention_residuals.py` | $\sim$ 40 runs |
| Problem 4 | `p4_recurrent_updates.py` | $\sim$ 40 runs |

The W&B project public-alchemy-a4 supplies sweeps for **Problems 1 and 2 only**: five learning rates for each of 21 architectures (105 outcomes). You do not need to repeat these sweeps. No runs or checkpoints are supplied for Problems 3 and 4: train those experiments yourself, reusing P1/P2's additive and tanh baselines. The starter includes attention-residual and base GRU implementations. `sweeps.py` maps implementations to configs; `PROVIDED_VARIANTS` lists only the released P1/P2 models.

Since we are providing so many runs, we expect the bulk of your runs to be dedicated to investigating diagnostics and designing thoughtful ablations for Problems 3 and 4. Please spend a lot of time designing careful ablations for each of these architectures.

Checkpoints are in Modal's `cs312-shared-data` environment, volume `hard-dl-dclm-v1`, mounted read-only at `/root/shared_data/a4_stability`. Its `manifest.json` lists run names, learning rates, outcomes, W&B links, and checkpoint paths. Completed models have final weights and an `early/` checkpoint at step 100; divergent attempts may have no usable checkpoint. Names match W&B and disk, e.g. `a4-rnn-no-residual-lr0.001`.

From the repository root, measure the released default d8 on Modal with:

```
uv run python -m experiments.a4_stability.measure_default
```

Edit the script's `RUN` constant to select another model. Detailed diagnostics are measured separately from training and saved locally as JSON; see the folder's `README.md` for usage.

## Stable signal propagation

Transformers and RNNs must propagate useful signals through long chains of computation without vanishing, exploding, or collapsing. Transformers carry representations across layers; RNNs also carry hidden states across time. We will ask which architectural ideas make these paths trainable and which ideas transfer between the two axes.

Poor validation loss can reflect limited model capacity or unstable optimization. Pair loss comparisons with diagnostics of activations and gradients to investigate the difference. Our main tools are:

- Residual connections preserve an identity path and let the model learn an update instead of replacing the whole representation at each step.
- Normalization keeps feature scales predictable before or after an update, changing which learning rates train stably and whether deeper models optimize at all.
- Gates make the update data-dependent. They let a model decide how much of a proposed update should enter the state.
- Other interventions, such as carefully scaled initialization and explicit parameter-norm and gradient-norm constraints, can also improve stability. We will consider these modifications out-of-scope for this assignment.

### Roadmap

1. Analyze transformer signal propagation across depth and time, investigate attention sinks, and compare component ablations and gated variants.
2. Diagnose plain 1-layer and 4-layer RNNs, then improve the 4-layer RNN across depth using transformer components. Compare the resulting RNNs and transformers, including how much prior context they use.
3. Build a mental model for depth residuals, then test full and blocked attention residuals in the modern RNN and d8 transformer.
4. Ablate the GRU components and design new hidden-state update rules across time without adding another memory state.

## Setup

Use vocabulary size 4096, context length 1024, embedding/hidden width 512, and batch size 64. The supplied runs train for 600,000 sequences (614.4M tokens); use the same budget for new variants. Match data and evaluation across variants; report parameter counts when comparing architectures.

### One-layer RNN baseline

The 1-layer RNN embeds tokens and updates its only recurrent memory:

$$e_t = E[x_t], \qquad h_0 = 0, \qquad h_t = \tanh(W_x e_t + W_h h_{t-1} + b_h),$$

It predicts the next token with logits $\ell_t = W_o h_t + b_o$ and probabilities $\mathrm{softmax}(\ell_t)$.

### Four-layer RNN baseline

The 4-layer RNN stacks four recurrent layers. Let $z_t^{(0)} = e_t$. For layers $\ell = 1, \ldots, 4$,

$$h_t^{(\ell)} = \tanh\left(W_x^{(\ell)} z_t^{(\ell-1)} + W_h^{(\ell)} h_{t-1}^{(\ell)} + b_h^{(\ell)}\right), \qquad z_t^{(\ell)} = h_t^{(\ell)}.$$

Initialize each $h_0^{(\ell)} = 0$ and read logits $\ell_t = W_o z_t^{(4)} + b_o$ from the top layer. Problem 2 adds depth residuals, pre-norm, and MLPs; Problem 3 studies attention residuals across depth; Problem 4 changes the recurrent update across time.

The *plain* 4-layer RNN corresponds to not including any of the depth stabilization tricks from the transformer. The *modern* 4-layer RNN will copy over our core tricks from the class transformer: residuals, prenorm, and alternating MLP's.

### Eight-layer transformer baseline

Each block of the d8 causal LM has pre-norm residual attention and MLP updates:

$$a_\ell = x_\ell + \mathrm{Attn}_\ell(\mathrm{RMSNorm}(x_\ell)),$$
$$x_{\ell+1} = a_\ell + \mathrm{MLP}_\ell(\mathrm{RMSNorm}(a_\ell)).$$

The baseline uses eight heads, SwiGLU intermediate width 1792, and untied embeddings and output weights. It applies RoPE rotations and per-head RMSNorm to queries and keys, and final RMSNorm before the vocabulary projection. Problem 1 ablates these components and adds gates.

### Stability diagnostics

Keep the default RMS logger enabled during any new training. Its module activation, parameter, and parameter-gradient summaries are useful training statistics; measure activation and recurrent-state gradients separately using checkpoint probes.

Compare initialization, step 100, and the final checkpoint when available, using the same fixed held-out inputs across variants. Rebuild initialization with model seed 42. Use shared P1/P2 checkpoints, and save early and final checkpoints for your own P3/P4 runs. Record divergence when no usable checkpoint exists. Run probes in FP32 and evaluation mode, with gradients enabled for backward measurements.

Let $L$ be the mean next-token loss over the full sequence and $L_T$ the loss on only the final predicted token. Backpropagate each loss separately and plot, for each recurrent layer:

$$\left\|\frac{\partial L}{\partial h_t}\right\|_{\mathrm{rms}}, \qquad \left\|\frac{\partial L_T}{\partial h_t}\right\|_{\mathrm{rms}}, \qquad \|h_t\|_{\mathrm{rms}}, \qquad \Pr(|h_t| > 0.95).$$

- Hidden-gradient RMS for $L_T$ measures how much the final prediction depends locally on earlier states. Compare it with the gradient for $L$, which also receives contributions from predictions at earlier positions.
- Hidden-state RMS diagnoses whether the recurrent state keeps a stable activation scale.
- Tanh saturation fraction diagnoses whether the update has entered a low-derivative regime.

Compute RMS and saturation over batch and hidden dimensions at each position. Retain recurrent states on their temporal computation paths *before* stacking them for plotting.

For depth experiments, apply the same logic across layer index: compare layer-wise gradient norms, activation scales, and top-to-bottom gradient ratios.

## What matters across transformer depth?

We will investigate which components make the transformer trainable and how they affect signal propagation. The following intuitions motivate the experiments; they are hypotheses to test, rather than guarantees about validation loss.

### Components in the default transformer

- *Attention* mixes information across positions. Barring the softmax and QKNorm, the operation is linear. Attention accounts for approximately 24.2% of all parameters in our default d8 mode.
- *MLPs* transform each token independently through an expanded hidden representation.
- *Normalization* controls feature scales at different points, often thought to improve stability in high noise regimes. For a nice motivation, read Ioffe and Szegedy, 2017.
- *Residual connections* preserve an identity path through depth: each sublayer learns an update to the representation instead of replacing it. This creates a direct route for activations and gradients; see He et al. (2016).
- *Position encoding* gives attention access to token order beyond the causal mask.
- *Gates* make the scale of information flow input-dependent. The baseline already has a gate inside its SwiGLU MLP, but gates can additionally be introduced in attention, residuals, and norms.

### Additional variants

Let $x$ be a token's current depth-stream representation, $N$ the sublayer's RMSNorm, and $F$ its attention or MLP update. The baseline is $x' = x + F(N(x))$.

- **Post-norm** introduces normalization after the residual update:

$$x' = N(x + F(x)).$$

- **Gated residuals** multiply each branch update by a per-token, per-channel sigmoid gate:

$$g(x) = \sigma(W_g N(x) + b_g), \qquad x' = x + g(x) \odot F(N(x)).$$

  This borrows from the gated information flow in Highway Networks (Srivastava et al. (2015)).

- **Gated attention** multiplies each head output by a scalar gate before the output projection. For $y = N(x)$ and head output $A_j(y)$, use

$$g_j^{\text{attn}}(y) = \sigma(w_j^\top y + b_j), \qquad \text{Attn}^{\text{gated}}(y) = W_O \,\text{Concat}_j\left[g_j^{\text{attn}}(y) A_j(y)\right].$$

  For a study of output gating, stability, and attention sinks, see Qiu et al. (2025).

- **Gated norm** replaces each attention- and MLP-input normalization with

$$N^{\text{gated}}(x) = \sigma(W_n N(x) + b_n) \odot N(x).$$

### Problem 1: Diagnose transformer depth components

Use the supplied W&B sweeps and Modal checkpoints described on the first page.

Analyze how the components above affect stability across depth and information flow across the sequence. For each architecture, compare the five supplied learning rates and report the best final validation loss. Add an LR extension if needed to check a trend at the edge of the supplied range. Record divergent runs as well as successful ones.

(a) Analyze the supplied default class transformer runs.

(i) Measure activation RMS and activation-gradient RMS across depth, using the losses and measurement conventions from the Setup page.

(ii) Inspect attention patterns across sequence positions: how does attention mass vary with distance, how much goes to the current token, and do particular positions attract an unusual amount? Inspect individual heads as well as their average. Compare with uniform attention over each query's allowed causal prefix so the mask's preference for early positions is not mistaken for a learned attention sink.

(b) Analyze the following supplied variants. Each changes only the listed component from the baseline.

(i) **No residual connections:** use $x' = F(N(x))$ in both sublayers.

(ii) **No pre-norm:** use $x' = x + F(x)$ in both sublayers; keep QK normalization and final output RMSNorm.

(iii) **No QK normalization:** remove query/key RMSNorm; keep both input pre-norms.

(iv) **No RoPE:** remove query/key rotations; keep the causal mask.

(v) **No final output RMSNorm:** project the final depth stream directly to logits.

(vi) **Post-norm:** replace pre-norm with the post-norm rule above.

(vii) **Gated residuals:** add the per-channel gate to both sublayer updates.

(viii) **Gated attention:** add the per-head output gate above.

(ix) **Gated norm:** replace both sublayer input norms with the rule above.

Track validation loss and repeat the depth and sequence diagnostics for each variant. Which changes improve loss, and which diagnostics help explain the differences? Distinguish observed correlations from evidence for a causal mechanism.

## Train deep RNNs

RNNs are notoriously difficult to train for a host of reasons. The training behavior of plain 1-layer and deeper 4-layer RNNs illustrates these difficulties.

Since RNNs have not been given much love since the popularization of transformers in 2017, there is hope that we can improve their performance significantly by carrying over the last decade of stability tricks. The modern RNN combines the plain recurrence with the standard depth stabilization tricks of a transformer.

### Problem 2: Transfer depth-stabilizing tools to stacked RNNs

Use the supplied W&B sweeps and Modal checkpoints described on the first page.

The core question is whether the model can propagate useful signal across a long time horizon. Use the across-time stability diagnostics defined in the Setup page: hidden-gradient RMS for both the full-sequence loss $L$ and the last-token loss $L_T$, hidden-state RMS, and tanh saturation fraction. Plot these statistics as a function of time index $t$, then interpret whether the recurrence is vanishing, exploding, saturating, or roughly balanced.

(a) Compare the supplied learning-rate sweeps for the 1-layer and 4-layer plain RNNs. Report the best validation loss for each.

(b) Look at the model diagnostics suggested in the setup to visualize information flow across depth and sequence dimension.

(c) Analyze the supplied learning-rate sweep for the default *modern* 4-layer RNN.

(d) Compare the supplied ablation sweeps below with the modern baseline, and repeat the depth and temporal diagnostics. Add an LR extension only if needed to check the trend. Each variant changes one component.

(i) **No depth residuals:** remove the identity skips around both the recurrent branch and MLP. Keep the recurrence across time unchanged.

(ii) **No pre-norm:** remove the input norms of both branches.

(iii) **Add position encoding:** rotate each normalized recurrent input, $y_{\ell,t} \leftarrow R_t y_{\ell,t}$, before its input projection, using the RoPE frequency rule from Problem 1. This tests explicit position information; an RNN has no query/key pairs, so it is an analogy to transformer RoPE rather than the same operation.

(iv) **Add final output RMSNorm:** normalize $x_{5,t}$ before the vocabulary projection. The direction is reversed from Problem 1 because the RNN baseline has no final output norm.

(v) **Post-norm:** replace pre-norm with a norm after each depth residual sum, both for the recurrent branch and for the MLP.

(vi) **Gated depth residuals:** use separate per-channel gates on the recurrent and MLP branch outputs, as in Problem 1. Compute each gate from that branch's normalized depth input.

(vii) **Gated recurrent output:** split $h_{\ell,t}$ into eight equal channel groups and multiply group $j$ by $\sigma(w_{\ell,j}^\top y_{\ell,t} + b_{\ell,j})$ before adding it to the depth stream. This is the counterpart of the transformer's per-head attention gate; leave the MLP unchanged.

(viii) **Gated norm:** replace both branch-input LayerNorms with $N^{\text{gated}}(x) = \sigma(W_n N(x) + b_n) \odot N(x)$, with separate parameters for each branch.

There is no QK-normalization ablation for an RNN without queries and keys. For both output-gating variants, carry the *ungated* $h_{\ell,t}$ forward in time; the gate only changes what enters the depth stream. Compare each intervention's loss change relative to its own architecture's tuned baseline. Which interventions matter more for RNNs than transformers, and which diagnostics support that comparison?

(e) Choose a competitive RNN and transformer. Compare activation RMS and similarity across layers and sequence positions. For example, plot cosine similarity between representations at adjacent layers and between positions at several lags. Does high similarity imply that a layer or a token can be removed without affecting predictions?

(f) Design an experiment to compare how much prior context the models use. One option is to evaluate the same target tokens with successively shorter prefixes and plot the loss increase relative to full context. Reset RNN states when removing a prefix, preserve transformer position offsets, and control separately for the start-of-sequence token. Compare useful context with last-token gradient reach; explain why they need not agree.

## Route information across depth

*Attention residuals* replace additive depth updates with mixtures of earlier raw branch outputs. With embedding $u_0$, sublayer $j$ computes

$$\alpha_{ji} = \text{softmax}_{i<j}\left(q_j^\top K_j(u_i)\right), \qquad r_j = \sum_{i=0}^{j-1} \alpha_{ji} u_i, \qquad u_j = F_j(N_j(r_j)).$$

Here $K_j$ is a separate RMSNorm for scoring and $q_j$ is a learned query. The mixture combines unnormalized values at the same token.

*Blocked attention residuals* route over the embedding, sums of branch outputs from completed blocks, and the current block's partial sum when nonempty. Blocks contain $B$ sublayers, counting recurrent/attention and MLP branches separately.

EMA-style residuals over depth decay only the carried stream: $x' = \beta x + F(N(x))$. Half-and-half residuals mix the two preceding stream states, adding the branch separately:

$$x_1 = x_0 + F_0(N_0(x_0)), \qquad x_{\ell+1} = \tfrac{1}{2}x_\ell + \tfrac{1}{2}x_{\ell-1} + F_\ell(N_\ell(x_\ell)) \quad (\ell \geq 1).$$

Here $x_0$ is the embedding; each recurrent/attention or MLP branch is one depth step. Uniform routing sets $\alpha_{ji} = 1/j$ over prior sources.

### Problem 3: Attention residual case study

Train variants of both the *modern 4-layer RNN with MLPs* and *d8*; reuse the supplied additive baselines. Tune 2–3 learning rates per new variant, extending the range if needed. Keep branch norms and the RNN's temporal recurrence unchanged. Initialize attention-residual queries to zero and route once more before the usual output head.

For every variant, compare loss and plot statistics *across depth* at the Setup's checkpoint stages on fixed held-out inputs: activation RMS, activation-gradient RMS for $L$ and $L_T$, branch/input RMS ratios, and adjacent-depth cosine similarity.

(a) Train both models with fixed mixing patterns: half-and-half mixing, depth EMAs with $\beta = 0.1, 0.9$, and uniform routing. Compare with additive residuals and build a mental model using your depth statistics.

(b) Train both models with full attention residuals. Compare loss and depth statistics with the baseline, and inspect routing weights across depth.

(c) Train both models with blocked attention residuals, $B = 2, 4$. Compare with full routing. Do the results support your mental model?

(d) Explore your own residual scheme over depth and try to come up with a new variant that beats everything you have considered so far. Implement and train it, using your depth diagnostics to guide the design. Tune 2–3 learning rates, extending the range if needed, and compare with the supplied tuned baselines.

## Augment the RNN across time

In a residual pre-norm RNN with MLPs, each layer applies

$$a_{\ell,t} = x_{\ell,t} + u_{\ell,t}, \qquad u_{\ell,t} = \phi_\ell(h_{\ell,t-1}, \mathrm{Norm}(x_{\ell,t})).$$

$$x_{\ell+1,t} = a_{\ell,t} + \mathrm{MLP}_\ell(\mathrm{Norm}(a_{\ell,t})).$$

The recurrent state carried to the next token is $h_{\ell,t} = u_{\ell,t}$; the MLP acts on the depth stream.

**Example: GRU.** The GRU mixes the old hidden state with a tanh candidate. Its update gate $z_t$ sets the fraction of new information written; its reset gate $r_t$ controls which old-state coordinates enter the candidate's recurrent projection:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \qquad r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r),$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h),$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t.$$

### Problem 4: Design a better update across time

Use the modern 4-layer RNN from Problem 2, keeping its residual connections, LayerNorm, and SwiGLU MLPs fixed. Try to improve the model only by changing the hidden-state update $\phi_\ell$ inside each recurrent layer.

Your update $h_t = \phi(h_{t-1}, x_t)$ may use the current input, learned parameters, residual paths, gates, and normalization, but may carry only the hidden state across time: no separate cell state, attention cache, or other memory.

(a) Tune and train a modern GRU with the same depth residuals, pre-norm, and MLPs. Choose and implement your own ablations to investigate which recurrent components are most helpful. Compare tuned losses and checkpoint diagnostics.

(b) Propose, implement, and train a hidden-state update inside the same 4-layer residual pre-norm MLP RNN. Tune 2–3 learning rates, extending if needed. Assess its improvement over the supplied tanh baseline; also compare with the GRU.

(c) Use your diagnostics to explain the differences and guide improvements.