

Trace a Residual Stream

Gavin · AI-assisted course design

Contents

1	Goal	3
2	Part 1 — Predict a fresh residual sequence	3
3	Part 2 — Compare three different changes	3
4	Part 3 — Optional implementation of the toy	4
5	Part 4 — Optional trained-model observation	4
5.1	Use a bounded public specimen	4
5.2	Local setup and commands	4
5.3	Implementation requirements	5
5.4	Observe and interpret	6
6	Checkpoints and solutions	6
7	Submit an explanation	7
8	More Learning	7

1 Goal

Track the difference between a residual state and an update. Predict how changing an early branch affects a later calculation. Optionally record clearly identified residual boundaries in a small trained Transformer.

Prerequisite: Lesson 2.1.

Time: 45–60 minutes for the hand trace. Allow additional setup time for the optional real-model path.

Core requirements: paper or a text editor. The first path is fully offline and requires no API, account, paid compute, GPU, installed machine-learning library, or model download. A hand-worked submission completes the core Lab.

Expected artifact: a short `submission.md` or equivalent note containing predictions, calculations, explanations, and limitations. Distinguish analytically checked expectations from outputs you actually observed in software.

2 Part 1 — Predict a fresh residual sequence

Use the lesson’s hand-constructed functions with a fresh input:

$$x = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}, \quad F(x) = \begin{bmatrix} -x_1 \\ 2 \\ 0 \end{bmatrix}, \quad G(y) = \begin{bmatrix} y_2 \\ -y_2 \\ 2y_1 \end{bmatrix}.$$

Calculate $y = x + F(x)$ and $z = y + G(y)$. The functions are affine or linear update rules chosen for arithmetic; neither implements attention or a Transformer MLP. There is no normalization in this toy.

Before calculating, predict:

1. Will the first coordinate of y depend on the first coordinate of x ?
2. Must each addition increase the stream’s norm?
3. If you remove F , can you recover the changed output by subtracting the old $F(x)$ from the old z ?

Then make a trace with five labeled vectors: input, first update, intermediate stream, second update, and final stream. Check that every vector has shape 3×1 .

Calculate their Euclidean norms and the total change $z - x$. Explain why the norm of the total change need not equal the sum of the two update norms. A diagram of three-dimensional arrows is optional; coordinate arithmetic is sufficient.

3 Part 2 — Compare three different changes

Reset to the same fresh input for every case.

1. **Suppress the first branch:** replace its output with zero, and recompute the second branch from its changed input.
2. **Suppress the second branch:** retain the first update, then replace only the second update with zero.
3. **Parallel arrangement:** evaluate both branches on the original input, producing $z_{\text{parallel}} = x + F(x) + G(x)$.

Record a prediction for each before doing its arithmetic. Explain why case 3 changes the architecture rather than simply renaming an intermediate vector.

Use the scalar readout $s = [0, 0, 1]z$, which selects the final third coordinate. Compare the two baseline update norms with the effect of suppressing each branch on s . Does the larger update necessarily have the larger effect? Limit your conclusion to this specified calculation.

Finally, give the toy two positions x and $\tilde{x} = [-2, 0, 3]^T$. Arrange them as rows of a 2×3 array. Apply the same update rules separately to each row, transposing your individual-vector results as needed. Explain why these rules never move information from one position into the other.

4 Part 3 — Optional implementation of the toy

A standard-library Python implementation can reproduce the trace on a CPU. Implement checked vector addition, the two update rules, Euclidean norm, and independent intervention cases. Reject mismatched vector lengths rather than silently truncating them.

Retain copies of intermediate vectors. Confirm that running an intervention does not mutate the baseline input or stored baseline results. Compare every intermediate vector with your hand trace before checking just the readout.

If you use the supplied runner or a coding assistant implements this specification, inspect the arithmetic and save the actual run output, Python version, and command. Analytical answers below are not evidence that your script has run.

5 Part 4 — Optional trained-model observation

5.1 Use a bounded public specimen

Use EleutherAI/Pythia-14M at revision 94f7c35d5e9f2e9bac8ca839329f505b4d007d5d. This immutable publisher commit contains the final checkpoint. Its config specifies six blocks, width 128, and `use_parallel_residual: true`.

Download only `config.json`, `tokenizer.json`, `tokenizer_config.json`, `special_tokens_map.json`, `generation_config.json`, and `model.safetensors` from that revision. The weights file is 28,143,920 bytes, about 28.1 MB; tokenizer and configuration files are additional. Inspect the planned total before downloading. Do not download every training checkpoint, optimizer states, `.bin` weights, or custom Python files. Framework installation can consume substantially more space than these weights.

The model card records a February 2026 replacement of the formerly hosted deduplicated model. Pinning this revision avoids silently mixing those histories. Use the model for local inspection, not as a reliable assistant or factual source.

5.2 Local setup and commands

The toy runner supplies an optional standard-library implementation of Part 3. Save it locally and run `python3 trace_residual_toy.py` after making your hand trace. Its JSON output records actual Python version and independent cases. Compare the intermediate arrays with your calculations.

For the trained-model route, save the observation runner and pinned requirements in a new directory. Use a separate environment from Lab 02: these libraries require a newer Tokenizers version. This

setup was tested with CPython 3.13.13, PyTorch 2.9.0, Transformers 4.57.1, Tokenizers 0.22.1, Safetensors 0.6.2 and Hugging Face Hub 0.35.3. The download client is CPython 3.13.13's standard-library `urllib.request`; model inference uses the explicit eager attention backend.

On macOS with Python 3.13 installed:

```
python3.13 -m venv .venv-residual
.venv-residual/bin/python -m pip install -r residual-requirements.txt
.venv-residual/bin/python inspect_residual_stream.py --plan
.venv-residual/bin/python inspect_residual_stream.py --artifacts pythia-artifacts
--output run-01
```

On Linux, install the CPU-only PyTorch wheel before the requirements, avoiding optional CUDA packages:

```
.venv-residual/bin/python -m pip install torch==2.9.0 --index-url https://
download.pytorch.org/whl/cpu
.venv-residual/bin/python -m pip install -r residual-requirements.txt
```

On Windows use `py -3.13 -m venv .venv-residual`, replace the Python executable with `.venv-residual\Scripts\python.exe`, and use the CPU-wheel command before installing requirements. Package installation can require substantially more disk space than the specimen files.

The plan prints exactly six files totaling **30,264,046 bytes**, including the 28,143,920-byte weights file. The runner verifies each fixed byte size and SHA-256 hash; it rejects extra artifact files and altered caches. Initial downloads retrieve only that scope, without an account. Text is fixed to the harmless sentence below and processed locally. Afterward repeat into a fresh directory without downloading:

```
.venv-residual/bin/python inspect_residual_stream.py --offline --artifacts pythia-
artifacts --output run-02
```

Open `manifest.json`, `observations.json` and `captures.json` in each result directory. These record versions, provenance, token IDs/pieces/positions, full cloned raw states and branch updates, selected-position metrics, separate final normalization, actual maximum numerical errors, repeated-input comparisons, and hook-removal checks. Undefined cosines are JSON `null`. The runner refuses to overwrite nonempty output directories. Retain unexpected observations; do not replace them with the analytical toy answers. The implementation checks are measured on the machine that runs it; no expected trained-model norms are supplied.

5.3 Implementation requirements

The following requirements describe the supplied runner and any independent implementation. Inspect them alongside your actual run; published requirements are not your measured results.

- Use CPU execution and float32 model computation, an explicit evaluation mode, inference mode without gradients, and `use_cache=False`. No inference API, GPU service, account, or payment is needed. Initial artifact and dependency downloads require internet access.

- Use Transformers’ built-in GPTNeoXForCausalLM implementation and a fast tokenizer with `trust_remote_code=False`; load only safetensors weights. After the allowlisted download, load from that local directory with `local_files_only=True` so the observation run needs no network. Never enable remote custom code to solve a loading error.
- Pin and record tested versions of Python, PyTorch, Transformers, tokenizers, safetensors, and the download client. The inspected architecture reference is Transformers v4.57.1; record any implementation-version change and recheck its boundaries.
- Use a short nonprivate input, such as `The small red boat crossed the lake`. Record actual token IDs, decoded pieces, and positions before selecting a token. Do not assume words coincide with tokens.
- Record the first block’s input, each of the six raw block outputs, and the separate output of the final LayerNorm. Clone captured tensors and remove observation hooks afterward.

At each block, this checkpoint adds attention and MLP updates calculated from separately normalized versions of the same input. Capture branch outputs at their addition boundaries as well if verifying that identity. Source inspection must establish the hook locations and whether a module returns a tensor or a tuple; do not guess from a generic hook recipe.

Do not label a returned `hidden_states` tuple as seven raw residual states without checking the implementation. In this architecture, final normalization changes the last returned boundary. Keep the raw last-block output and final normalized output separately labeled.

5.4 Observe and interpret

For one selected token position, report each raw state’s norm, consecutive raw-state difference norm, and cosine similarity where both norms are nonzero. Mark undefined cosines explicitly. Keep final-normalization changes separate from block updates.

Verify expected shapes $(1, n, 128)$ and, if capturing branches, the parallel addition identity within a declared floating-point tolerance. Record maximum error rather than reporting only “passed.” Repeat the unchanged input as a reproducibility check.

Describe one observed pattern and two explanations consistent with it. Explain what additional evidence would distinguish those explanations. Do not label a large update as new factual knowledge or successful reasoning. No expected trained-model numbers are provided: actual observations must come from your recorded run.

6 Checkpoints and solutions

i Analytically derived answers

The first intermediate coordinate is always $y_1 = x_1 - x_1 = 0$. For the fresh input:

- $F(x) = [-1, 2, 0]^T$ and $y = [0, 4, -1]^T$.
- $G(y) = [4, -4, 0]^T$ and $z = [4, 0, -1]^T$.

- Norms of input, first update, intermediate stream, second update, and final stream are $\sqrt{6}$, $\sqrt{5}$, $\sqrt{17}$, $\sqrt{32}$, and $\sqrt{17}$.
- $z - x = [3, -2, 0]^T$, with norm $\sqrt{13}$.

Suppressing the first branch gives $G(x) = [2, -2, 2]^T$ and output $[3, 0, 1]^T$. Subtracting the baseline first update from the baseline output would give $[5, -2, -1]^T$, which is a different calculation.

Suppressing the second branch leaves $y = [0, 4, -1]^T$. The parallel arrangement gives $[2, 2, 1]^T$.

The baseline readout is -1 . Suppressing the smaller first update changes it to 1 ; suppressing the larger second update leaves it at -1 . Update norm therefore does not rank these intervention effects.

For the second position $\tilde{x}$, the first update is $[2, 2, 0]^T$, its intermediate state is $[0, 2, 3]^T$, its second update is $[2, -2, 0]^T$, and its final state is $[2, 0, 3]^T$. Stack these results as rows; neither computation references the other row.

7 Submit an explanation

Include original predictions, corrected arithmetic, baseline and intervention traces, readout comparison, and the two-position explanation. If you executed code, attach its actual outputs and environment details. If you performed the optional model observation, include checkpoint revision, download scope, exact measurement boundaries, shape and numerical checks, observations, and limitations. A complete hand trace remains a valid core submission.

8 More Learning

- Lesson 2.1 explains the geometry and architecture choices.
- Pythia's pinned configuration and GPT-NeoX source establish the optional specimen's actual wiring.
- Reading the Original Transformer Paper connects this trace to a different historical architecture. Explain the differences before transferring your observation code to it.