

Lab 6.2 Build a Reusable LLM Microscope

Gavin · AI-assisted course design

Contents

1 Goal	3
2 Part 1 — Inventory the evidence before integrating it	3
3 Part 2 — Define a small command contract	4
4 Part 3 — Specify the run schema and bounds	5
5 Part 4 — Verify the specimen and test failure paths	8
6 Part 5 — Run one bounded integration demonstration	9
6.1 Freeze the imported direction and exact prompts	9
6.2 Use the exact fourteen-attempt schedule	10
6.3 Validate before interpreting	10
7 Part 6 — Regenerate analysis without the model	11
8 Part 7 — Meet the acceptance criteria and review export	11
9 Optional closing investigation — Ask Your Own Question	12
10 More Learning	13

1 Goal

Connect existing model inspection and intervention work into a small, reproducible command-line pipeline:

verified model → **named capture** → **declared intervention** → **saved evidence** → **repeatable analysis**.

Reuse the measurements from Labs 5.1 and 5.3, then perform a fixed fourteen-attempt integration demonstration. The required outcome is an auditable instrument and an honest report. Successful semantic steering is not an acceptance criterion.

Prerequisites: Lesson 6.2, the experimental-design habits of Lesson 6.1 and Lab 6.1, and completed, locally saved core measurements from Lab 5.1 and Lab 5.3. The bounded specimen originates in Lab 2.2; retain Lab 4.1’s distinction between measured runtime and performance claims and Lab 4.4’s explicit operation boundaries.

Time: allow 2–3 hours for integration, tests, and explanation, depending on the state of your earlier code. This is a planning estimate, not a measured execution result.

Requirements: the existing verified Pythia-14M cache and Python environment, local CPU, and ordinary files. No new model weights, inference API, account, GPU, paid compute, AWS provisioning, website, or front end is required.

Expected artifact: learner-implemented source, a frozen plan, verified imported evidence, a bounded fresh run, independently regenerated analysis, and a 400–600-word report. The commands below are the **implementation contract**. A reference implementation is available in Part 2; inspect and validate it against your own recorded inputs. No measured outputs or runtime guarantees are supplied here.

i Reference implementation verification status

The supplied runner has passed synthetic integration, bookkeeping, import validation, replay/failure, interruption, supervision, saved-only analysis, and export tests. Fixture results establish the tested code paths; they are not real-model measurements.

The fourteen-attempt Pythia demonstration has **not run** for this implementation. Same-environment numerical replay, real-model observer invariance, intervention effects, and hook restoration remain unverified. Part 7’s measured-integration requirements are mandatory and remain unmet; fixture tests do not waive them. You can study and test the instrument without claiming completion of the measured Lab. A missing compatible local cache or prerequisite measurement must remain an explicit blocker in your report.

2 Part 1 — Inventory the evidence before integrating it

Create a read-only inventory of your completed Lab 5.1 and Lab 5.3 runs. Identify the original files containing configuration, prompts and tokens, runtime versions, source identity, checks, numerical arrays, and reports. Preserve originals. Do not rerun either full Lab as an automatic import fallback.

For Lab 5.1, require the eight core prompts' actual final-position captures: seven raw residual states, separate final-normalized state, and six pairs of branch contributions. Require their observer-invariance, restoration, residual-addition, and output-reconstruction checks. Exclude the optional probe from this integration unless it is separately labeled and excluded from the core summaries.

For Lab 5.3, require the completed core record: discovery vectors, frozen predictions, frozen direction and dose, exact fixtures and tokenization, chronological attempts, final logits, checks, and metrics. Verify its planned 82 attempts, including the deliberate cleanup exception, and establish completeness from the schedule, arrays, and recorded checks. Preserve an original declared status if present; if absent, record it as missing and label the importer's completeness assessment derived. Earlier Labs do not require a standalone status file. Import the original four-review primary comparison and all controls, not just the favorable arms.

If earlier file layouts differ, write a small explicit adapter that maps their fields into the schema below. It must preserve meanings and identify original filenames and hashes. Do not infer a missing boundary from an ambiguous array shape or create historical predictions after observing results. A new adapter is conversion code and belongs in the source manifest.

Hash files now, and compare with any hashes recorded by their original runs. Distinguish `matched_original_manifest` from `recorded_at_import`. The latter establishes identity from this import onward, not integrity since the original experiment. Retain evidence of when predictions were frozen; a fresh timestamp cannot replace it.

If required earlier measurements, source information, or checks are missing, report the exact prerequisite as blocked. You may complete schema and fixture tests, but label the integrated measured demonstration `not_run`. Implementing the original Lab later is separate work under that Lab's plan and limits. Never fabricate activations, treat answer-key values as measurements, or silently add 82 forwards to this Lab's budget.

3 Part 2 — Define a small command contract

Use one entrypoint, for example `microscope.py`, and a few functions or small modules. Reuse inspected capture and intervention code rather than rewriting the numerical experiment while changing its packaging. No generic plugin loader, arbitrary module execution, remote service, database, scheduler, or web interface is needed.

The reference runner implements the following commands. Download `microscope.py`, `microscope_io.py`, `microscope_imports.py`, `microscope_analysis.py`, and `microscope-plan.example.json` into one directory. Keep the reused `inspect_activations.py`, `contrast_experiment.py`, `inspection_common.py`, `inspect_residual_stream.py`, and `lab19-fixtures.json` beside them. Copy the example plan to a learner-owned file and replace its question and prediction placeholders before preparing a run.

Uppercase paths are placeholders to replace with verified local paths. The exact existing environment and import checks still apply; installing a different stack or substituting fixture arrays does not satisfy them.

```
python microscope.py prepare --plan microscope-plan.json --lab17 LAB17_RUN --lab19 LAB19_RUN --out NEW_RUN
python microscope.py verify --run NEW_RUN --model-dir LOCAL_CACHE
python microscope.py demonstrate --run NEW_RUN --model-dir LOCAL_CACHE
python microscope.py analyze --run NEW_RUN --out ANALYSIS_DIRECTORY
python microscope.py export --run NEW_RUN --analysis ANALYSIS_DIRECTORY --allowlist reviewed-export.json --out NEW_EXPORT
```

- `prepare` validates the plan, creates a new directory, imports a bounded snapshot of approved evidence, records hashes, and freezes input and prediction records. Copy the selected prompts' actually saved token fields into the frozen inputs, labeling separately any derived fields; `verify` retokenizes and compares without rewriting them. It performs no model load or forward. Refuse an existing destination.
- `verify` checks the imports, arrays, manifests, local model files, configuration, source identity, and actual environment. It tokenizes the two fresh-demo inputs locally. It performs no model forward. Write a verification record; do not label the fresh demonstration completed.
- `demonstrate` alone may load the model and perform the exact fourteen attempts in Part 5. It requires successful verification, demonstration state `not_run`, and zero prior forward attempts, then rechecks frozen hashes before starting. It has no sweep, generation, download, install, retry, or cloud fallback.
- `analyze` reads saved evidence with no model load, no tokenizer download, and no forwards. It writes a new analysis directory, records its own source hash, and refuses overwrite. Repeated analysis must not change raw records.
- `export` requires the selected analysis directory and copies only explicitly reviewed raw/analysis artifacts into a new local directory. Its allowlist entries name a source kind (`raw` or `analysis`) and a relative path; absolute paths, traversal, escaping symlinks, and unlisted sources are rejected. Verify that the analysis manifest names this raw run and its finalized manifest hash. Verify source hashes, forbid source/output overlap, and refuse an existing destination. Export neither uploads nor publishes anything.

Keep normal Python validation in these functions. Accept only supported schema versions, boundary names, arms, and numeric limits. Reject unknown configuration fields and non-finite parameters. Do not use `eval`, dynamic imports named by input data, shell commands supplied by a plan, or execution of model output. These safeguards do not turn the Python process into a security sandbox.

The `verify` and `demonstrate` commands use the existing environment. The `analyze` command may import NumPy to read numeric arrays, but must not load the model. A command that only prints a cached report has not demonstrated that its metrics can be regenerated.

4 Part 3 — Specify the run schema and bounds

Use schema identifier `llm-microscope/1`. Give every run a unique `run_id`. The exact filenames below form the minimal contract; additional documentation is allowed within the storage limit.

File or directory	Required meaning
<code>plan.json</code>	Frozen question, exact schedule, metric definitions, tolerances, limits, and expected checks
<code>environment.json</code>	Actual versions, OS, architecture, CPU, effective threads, dtype, device, and flags
<code>artifact-manifest.json</code>	Approved model/tokenizer files, revision, sizes, and SHA-256 values
<code>source-manifest.json</code>	Entrypoint, local adapters/helpers, imported runner identity, and source hashes
<code>imports.json</code> and <code>imports/</code>	Source run IDs, any original status, separately derived completeness, field mappings, file hashes, and copied evidence
<code>inputs.json</code> and <code>predictions.json</code>	Exact demo texts, token records, prior-evidence disclosure, prospective integration predictions, and frozen hashes
<code>attempts.jsonl</code>	Append-only start and finish events, including expected exceptions and unfinished attempts
<code>captures.npz</code> and <code>logits.npz</code>	Fresh numeric measurements with a metadata key for each array
<code>metrics.csv</code> and <code>checks.json</code>	Unrounded scalar measurements, comparisons, tolerances, and eligible/missing status
<code>status.json</code>	Stage states, scientific-outcome label, attempted/finished counts, elapsed processing time, and reason for stopping
<code>files.sha256.json</code>	Final integrity map of all finalized run files except itself

Before finalization, use frozen-input hashes to protect plan, predictions, imports, and source. Successful preparation and verification leave demonstration `not_run`; verification gates its one permitted execution. When the raw run reaches its terminal exit, write final status and checks first, then `files.sha256.json`, excluding itself. A prerequisite block leaves demonstration `not_run` and records the failed preparation/verification stage. A failed or partial demonstration retains its terminal state and available evidence.

A finalized raw run rejects further `verify` or `demonstrate` mutation. Repairs require a new run, preserving the original. A crashed, unfinalized directory is not eligible for reentry or analysis as a completed run; a read-only recovery report may identify its unfinished events, without resuming inference. `analyze` and `export` require and verify final hashes, write outside the raw root, and never rewrite its status or checks. Each analysis manifest binds its run ID to the raw run ID and SHA-256 of `files.sha256.json`. A hash manifest is not a self-authenticating signature.

Every stored measurement identifies `run_id`, `origin_run_id`, `prompt`, `arm`, `attempt`, `module path`, `boundary`, `raw/normalized status`, `block index`, `token ID` and `position`, `source shape`, `stored shape`, `dtype`, and `provenance`. Use `measured`, `derived`, or `fixture` for value provenance, with named parent arrays for derived values. Input prompts have separate provenance such as `authored_synthetic`; a synthetic prompt can produce measured activations. Fixture arrays cannot satisfy measured-run requirements.

Statuses are separate from provenance. Stage state is one of `not_run`, `running`, `completed`, `failed`, or `partial`; include a reason. An expected cleanup exception has attempt outcome `expected_exception`, `not_completed_logits`, and has no invented logits array. A run can finish successfully with that planned test and an unsupported steering hypothesis. An unexpected error marks the dependent stage failed. A timeout, cancellation, or interruption marks it `partial`; leave missing comparisons unavailable.

Enforce these core limits:

- CPU only; batch size one; two demo prompts, each 1–64 tokens; no padding or truncation.
- Exactly 14 planned and at most 14 attempted top-level model forwards. Increment the counter before invoking the model, including the expected exception. Refuse a fifteenth call.
- At most 15 minutes of cumulative automated processing across `prepare`, `verify`, and `demonstrate`, excluding installation and time spent writing the plan. Include hashing, copying, tokenization, and model loading in that cumulative 900-second allowance. Record actual per-command processing durations and carry their total forward. A parent watchdog enforces each active command's remaining allowance, including a stalled load or forward; also check between operations. Each offline analysis or export has a separate 60-second limit recorded in its own manifest; it must not update the finalized raw run.
- At most 50,000,000 bytes in the raw run, including copied imports but excluding pre-existing dependencies and weights. Each separate analysis directory is capped at 5,000,000 bytes; the reviewed export is capped at 50,000,000 bytes. Preflight sizes; reject an over-budget snapshot rather than copying the entire model cache. Bound decompressed array sizes as well as archive file sizes.
- No generation, training, fitting a new direction, extra random vectors, layer search, dose search, dependency installation, or network calls in the pipeline.

Use numeric NPZ arrays with `allow_pickle=False` and `max_header_size=10000`. Require an exact permitted array-key inventory, numeric dtypes, and shapes from the checked source schema and import mapping; a self-declared manifest alone cannot authorize arbitrary shapes. Captured vectors and logits remain `float32`; derived direction arrays retain their documented `float64` dtype. Reject object arrays, duplicate keys, unexpected members, and unsupported shapes.

Before allocation, inspect archive metadata and bounded NPY headers. Cap each decompressed member at 20,000,000 bytes and all numeric payloads in the raw run at 40,000,000 bytes, within the 50,000,000-byte total disk ceiling. These bounds accommodate the required `float32` logits even when one Lab 5.3 member stacks its successful passes. Calculate expected bytes from validated shapes/dtypes, then enforce actual streamed decompression caps as well as advertised sizes; reject mismatches and decompression bombs. Close archives after use. Validate filesystem paths against selected roots, reject traversal and escaping symlinks, and import only inventoried files. Do not load arbitrary files merely because a manifest requests them. NumPy loading documentation

A forced worker termination may prevent hook cleanup from running. Record `cleanup_unverified` for that attempt and discard the worker; never claim that a watchdog verified hook restoration. Preserve existing logs. There is no automatic resume or retry of a partially executed demonstration. A repair requires a separately reviewed new run; it does not reset a hidden counter inside this one.

5 Part 4 — Verify the specimen and test failure paths

Use EleutherAI/pythia-14m at revision 94f7c35d5e9f2e9bac8ca839329f505b4d007d5d. Reuse exactly these approved files: `config.json`, `tokenizer.json`, `tokenizer_config.json`, `special_tokens_map.json`, `generation_config.json`, and `model.safetensors`.

The previously verified six-file artifact totals 30,264,046 bytes. Its 28,143,920-byte `safetensors` weights have SHA-256 116a02532db461f91386a5b20f942ff2c8d4de7341e21b55caafc3d7b25f49a1. Verify all six local hashes against the saved artifact manifest. The publisher revision identifies the specimen. Missing cache is a blocker, not permission to download a replacement.

The existing tested reference stack is CPython 3.13.13, PyTorch 2.9.0, Transformers 4.57.1, tokenizers 0.22.1, `safetensors` 0.6.2, and huggingface_hub 0.35.3. Record actual versions, including build suffixes, and NumPy's actual version. Require compatibility with the original measured runs and source; stop the claimed same-environment replay if packages or computational settings differ. Do not silently install another stack.

Before loading, set `HF_HUB_OFFLINE=1` and `TRANSFORMERS_OFFLINE=1`. Use the local directory, `local_files_only=True`, `trust_remote_code=False`, `safetensors` only, built-in GPTNeoXForCausalLM, and the fast tokenizer. Offline flags are loading controls, not a general network sandbox. Transformers offline setup

Set CPU and float32 explicitly; require `model.eval()`, `torch.inference_mode()`, eager attention, seed 19, deterministic algorithms, and one intra-op and one inter-op thread set before computation. Pass `use_cache=False`, `output_attentions=False`, `output_hidden_states=False`, and `logits_to_keep=1` for each forward. Assert actual parameter dtype/device, six blocks, width 128, vocabulary 50304, and parallel residuals. No autocast, quantization, compilation, or automatic device placement.

Require logits shape (1,1,50304) and finite values. The observation capture from Lab 5.1 keeps only final-position vectors. The block-2 intervention uses the tuple-returning layer in Transformers 4.57.1's implementation. Inspect source and assert the actual runtime structure before using it.

Before any model execution, implement fixture-only tests for these cases:

1. Changed imported file bytes fail a recorded hash check. Change a temporary test copy, never an original run.
2. Missing required array, wrong shape, duplicate measurement key, non-finite value, or object array is rejected.
3. An ambiguous boundary or mismatched model/source identity cannot silently import.
4. A fixture-only run cannot be labeled measured or satisfy core completion.
5. An unknown arm, excessive token count, attempted fifteenth call, or unapproved path is rejected before a model call. A fake callback counter suffices for these limit tests. Add bounded-header,

oversized/decompression-bomb, and escaping-symlink fixtures; they must fail without allocating their claimed large arrays or reading outside approved roots.

6. An existing output directory is not overwritten. Absent measurements remain explicitly missing and are excluded from eligible denominators.
7. Repeated offline analysis leaves all raw-file hashes unchanged.

Save actual test results with `fixture` provenance. These tests can prove properties of the implemented bookkeeping on these cases; they cannot establish real-model numerical agreement. Neither lesson text nor a coding assistant's statement that a test passed replaces its recorded execution.

6 Part 5 — Run one bounded integration demonstration

6.1 Freeze the imported direction and exact prompts

Import Lab 5.3's block-2 discovery direction u and dose $\delta = 0.05R$ without changing orientation or scale. Recompute d , u , and R in float64 from its six saved float32 discovery vectors using Lab 5.3's formulas, and compare with the saved values using `atol=1e-10`, `rtol=1e-10`. Require finite values, a nondegenerate direction, and the original frozen-prediction hashes. A mismatch blocks demonstration; do not substitute a coordinate vector or a new direction.

Preserve the original intervention's cast and addition order from its recorded source. Record the actual float32 additions used for both signs and their norms. A matching width alone does not make a vector compatible.

Use exactly Lab 5.3's two first held-out review prompts, including one newline and no trailing whitespace:

```
[
  {"id": "H1+", "text": "The concert was wonderful and the musicians were talented.
\nOverall, it was"},
  {"id": "H1-", "text": "The concert was awful and the musicians were unskilled.
\nOverall, it was"}
]
```

Tokenize with `add_special_tokens=False`, no padding, and no truncation. Compare exact text, IDs, pieces, and positions with the fields actually saved in Lab 5.3. Derive lengths and selected final positions from those IDs, and the all-ones no-padding mask from each length; mark them derived when absent from the original record. From the original runner source, record whether it passed an explicit attention mask or used the unmasked default, and preserve that computational convention for replay. A derived mask is not evidence that an explicit mask was originally passed. Assert matching final token IDs. Check that " good" and " bad", with their leading spaces, each encode as one distinct token. Do not replace a multi-token metric with its first token.

These prompts have already been used and inspected. The new run is a **replay and integration check**, not a fresh held-out test of sentiment generalization. Freeze predictions about observer invariance, zero control, cleanup, numerical replay, and the expected signed effects while disclosing access to prior results. Do not call those expectations a preregistration of the original Lab 5.3 hypothesis.

6.2 Use the exact fourteen-attempt schedule

For H1+, then H1-, run these six attempts in order, resetting to unchanged input and model values each time:

Arm	Operation
Baseline	Unhooked final-position logits
Observe	Lab 5.1 read-only capture of all twenty final-position vectors
Zero	Lab 5.3 intervention path with an explicit zero vector at block 2
Positive	Add the frozen $+\delta u$ at block 2, final position only
Negative	Add the frozen $-\delta u$ at the same boundary
Restore	Unhooked baseline after all hooks have been removed

This consumes twelve attempts. Attempt 13 uses H1+ and a temporary block-2 hook that raises one specifically named test exception. Catch only that expected exception outside the hook context; verify handle removal. Attempt 14 reruns unhooked H1+ and checks its baseline. Do not attempt further model calls.

For every ordinary hook, require exactly one invocation at each expected boundary, shape $(1, n, 128)$, and stored selected-vector shape $(128,)$. Observation hooks detach and clone, then return `None`. Intervention hooks clone the whole residual tensor, change only $[0, n-1, :]$, and return $(\text{replacement},) + \text{output}[1:]$. Assert exact equality of every earlier row at that boundary. Record original and replacement final rows. Never modify the original output in place.

Use context-managed handles removed in `finally`, with registered-hook counts checked before and after every context. Per-prompt buffers must start empty. Preserve the expected exception's event and cleanup outcome without inventing a successful measurement for it. PyTorch hook reference

6.3 Validate before interpreting

For each prompt, compare its observed, zero, and restored logits against its fresh unhooked baseline using `atol=1e-6`, `rtol=1e-6`. Check attempt 14 against the fresh H1+ baseline. Record maximum absolute errors and pass/fail values. Also compare fresh baseline and signed-arm logits with their corresponding imported Lab 5.3 arrays using the same criterion; this is numerical replay under the verified source/runtime contract.

From each observed pass, check Lab 5.1's parallel identity $r_{\ell+1} = m_{\ell} + a_{\ell} + r_{\ell}$ in the implementation's addition order for every block, final normalization, and reconstruction of final logits from both h_f and r_6 . Use Lab 5.1's `atol=1e-6`, `rtol=1e-5` for these structural checks. Remove hooks before the diagnostic LayerNorm/head calls. These are bounded module readouts, not additional top-

level Transformer forwards; log them separately. Allow exactly two final-LayerNorm calls and two output-head calls per observed prompt for these checks, with no intermediate lens sweep.

A failed invariant stops subsequent interpretation and marks the stage failed. A non-finite value or unexpected exception stops execution. Do not loosen a tolerance, omit an arm, or continue with a plausible-looking plot. Preserve partial arrays and counts. No numerical result, effect sign, token ID, or elapsed time is prescribed as an observed answer.

7 Part 6 — Regenerate analysis without the model

Close the measurement worker. Run the proposed analyze command from saved files. Derive metrics in float64 from preserved float32 measurements. It must produce:

1. A geometry plot or compact table covering all eight imported Lab 5.1 core prompts and seven raw boundaries, with final normalization separate.
2. A paired-effect plot or table for all four imported Lab 5.3 review prompts under every original arm. Preserve its primary mean and separate off-task controls. Show all signs and missingness; do not reselect examples.
3. A separate fourteen-attempt integration summary, including observer/zero/restoration errors, replay errors, expected-exception cleanup, actual forward count, processing time, and storage use.
4. Fresh signed-arm values of $y = z_{\text{good}} - z_{\text{bad}}$, within-prompt Δy , actual perturbation norm and relative norm, and full-vocabulary $D_{\text{KL}}(p_0 \parallel p_a)$ calculated with stable `log_softmax`. A two-token-renormalized KL is not a substitute.
5. A 400–600-word report distinguishing imported scientific evidence from new integration evidence, naming failures and limitations, and giving a narrow causal conclusion if checks permit one.

For any undefined cosine, store `null` plus its reason, not zero. Keep unrounded metrics in machine-readable output. The fresh two-prompt mean is descriptive replay only; do not replace the original four-review endpoint with it or count imported rows twice as independent observations.

Run analysis again into a second new directory without a model path and compare numeric tables and substantive report content. Timestamps or file metadata may differ; measurements must not. Verify raw-run hashes remain unchanged. Do not require binary-identical compressed archives or plots when metadata can vary, and do not mislabel repeated analysis as a repeated experiment.

8 Part 7 — Meet the acceptance criteria and review export

The core measured Lab is accepted only when all of the following are evidenced:

- Source and proposed commands are implemented; recorded tests actually run.
- The exact local checkpoint, source, runtime, imports, predictions, and tokenization are identified and validated.
- Imported evidence retains its true provenance and original checks; fixture-only values are not used as measurements.
- The fourteen-attempt schedule finishes with thirteen real logit outputs, one expected exception, no extra forwards, and all required integrity, invariance, structural, replay, and cleanup checks passing.
- Storage/time limits and the actual counts are recorded; failed and partial states remain inspectable.
- Offline analysis regenerates all required comparisons without loading a model or changing raw data.

- The report separates execution success, observed effects, and interpretation limits. It does not claim unseen generalization or a universal sentiment mechanism.
- The export preview uses an explicit allowlist and contains no unintended private data.

A correctly reported blocker is an honest partial submission, but it does not satisfy the measured integration criterion. Conversely, passing instrumentation with zero or reversed steering effects can satisfy the Lab. Explain which case you obtained.

Before export, inspect the exact candidate filenames and contents. Allow only reviewed source, synthetic fixtures, sanitized configuration/environment summaries, necessary arrays, plots, and the report. Exclude credentials, complete environment-variable dumps, unrelated files, personal prompts, absolute home paths, and unpublished private experiments. Avoid distributing model weights merely because the local cache exists; identify the publisher source and review applicable licenses for any redistribution.

The export must include the selected analysis manifest and verify the hash of every copied source. Its manifest lists source kind, relative source path, destination path, and hash. The export command creates a local folder only. Publishing or sharing it is a separate decision about the recipient and data. Nothing in this Lab requires changing website content, deploying a service, or making results public.

Checks to explain back

- The planned count is $2 \times 6 + 2 = 14$ attempts. Thirteen return logits; the intentional exception does not.
- Twenty float32 vectors of width 128 occupy 10,240 raw bytes per observed prompt; array metadata and logits are additional.
- Reusing Lab 5.3's direction does not fit another direction. Replaying its inspected prompts does not create an untouched test set.
- A zero edit exercises replacement code; a read-only observer exercises a different path. Both require comparisons.
- A file hash can reveal changed bytes relative to a reference; it cannot prove an honest experiment or authenticate an undocumented history.
- A partial run has missing evidence. Replacing missing values with zeros changes the claimed result.
- The specimen is dense. Routing records are not_applicable; there is no real MoE measurement in this core.

9 Optional closing investigation — Ask Your Own Question

Curriculum v6's learner Research Capstone can begin as an extension of this Lab. Choose one modest instructional question, such as whether a fixed intervention effect persists across one new wording family, or whether two explicitly defined summaries tell different stories about the same saved measurements. A careful replication or negative result is a useful outcome.

Write a short new plan: question, predicted result, specimen, inputs, comparison, confounders, metric, numerical checks, evidence that would count against the prediction, and a stopping rule. Keep exploration separate from any untouched evaluation set. Reuse saved evidence when the question permits it; new observations require a separate bounded plan and honest provenance.

This optional investigation does not add a new activity type or silently extend the fourteen-attempt core. It does not require novel research, a larger model, or cloud compute. If a chosen question genuinely needs resources beyond the approved local setup, pause for approval of the specific compute, cost, and lifecycle plan. Budget alerts are not hard spending caps, and stopping an instance does not remove every associated cost. AWS Budgets; EC2 stopping guidance

Finish with the question you investigated, what the evidence established, what remained unresolved, and one better next question. The aim is a small investigation that another learner can inspect and understand.

10 More Learning

- Lesson 6.2: revisit the full provenance chain from specimen to conclusion.
- PyTorch numerical accuracy: explain why declared tolerances and actual environment details matter.
- Python SHA-256 interfaces: implement exact-byte integrity checks without treating them as proof of authorship.
- Towards Best Practices of Activation Patching: connect implementation choices to the scientific question they can answer.