

Lab 12 — Measure Inference and Sampling

Gavin · AI-assisted course design

Contents

1	Goal	3
2	Optional implementation reference	3
3	Part 1 — Predict before running	3
4	Part 2 — Establish a bounded specimen and environment	4
	4.1 Reuse the exact artifact	4
	4.2 Keep the software environment separate	4
5	Part 3 — Record what the model actually receives	5
	5.1 A small sampling matrix	5
6	Part 4 — Test caching on matched prefixes	6
7	Part 5 — Measure a defined workload repeatedly	7
	7.1 Choose timing boundaries	7
	7.2 Make length a controlled variable	7
	7.3 Summarize without hiding variation	8
8	Part 6 — Report memory honestly	8
9	Optional bounded extension — Inspect batching	9
10	Optional accelerator planning only	9
11	Analytical checks	9
12	Submission checklist	10
13	More Learning	10

1 Goal

Run one tiny, pinned language model on CPU and distinguish changes in token selection from changes in repeated computation. Preserve the inputs and all outcomes, test cache correctness on identical prefixes, and measure explicitly defined timing boundaries.

Prerequisites: Lesson 4.1, basic Python, and the model-loading discussion in Lab 05. A programming assistant can help implement the experiment, but the learner should explain its controls and inspect the records.

Estimated time: 60–90 minutes, plus initial dependency installation and artifact download if needed.

Deliverable: a reproducibility manifest, complete run records, a cache-comparison report, and a short explanation supported by actual results.

This page specifies an experiment to implement and verify. It is not a supplied, tested notebook, and it contains no measured Pythia outputs or timings. The analytical checks at the end do not establish that a model run succeeded.

2 Optional implementation reference

After writing the predictions below, you can use the maintainer’s bounded CPU runner. Save the artifact helper beside it; use the existing Lab 05 environment pinned by `residual-requirements.txt`, with CPython 3.13.13. The runner uses POSIX wall-clock timers and currently supports Linux and macOS. On Windows, the written experiment remains available, but this runner stops rather than using an unbounded fallback.

```
python measure_inference.py --output inference-run --plan
python measure_inference.py --artifacts pythia-artifacts --offline --output
inference-run
```

Omit `--offline` only when the six pinned files are missing and you intend the bounded public download. Use a new output directory. The runner preserves all ordinary generations, separate post-EOS profiling trials, raw first-step logits, full matched-prefix logit vectors, timing boundaries, effective configurations, and failures. Its declared cache tolerance is fixed before observing errors; a failed diagnostic is retained, not silently relaxed. Batching and accelerator extensions are not performed by this runner.

3 Part 1 — Predict before running

Write predictions, including uncertainty, before examining generated text:

1. Will repeated greedy runs return identical token IDs in the same environment?
2. Must every pair of different sampling seeds produce different continuations?
3. How might reducing temperature differ from reducing top-p?
4. Should correctly cached and uncached logits agree on the same prefix? Must every element be bitwise identical?
5. Will a cache necessarily improve wall time for this tiny model and short inputs?
6. Which memory categories will a parameter-byte count omit?

Do not replace these predictions with hindsight. Add a separate interpretation afterward.

4 Part 2 — Establish a bounded specimen and environment

4.1 Reuse the exact artifact

Use EleutherAI/pythia-14m at revision

94f7c35d5e9f2e9bac8ca839329f505b4d007d5d.

Reuse the verified Lab 05 download when available. Otherwise allowlist only:

- `config.json`
- `tokenizer.json`
- `tokenizer_config.json`
- `special_tokens_map.json`
- `generation_config.json`
- `model.safetensors`

The publisher commit records a 28,143,920-byte weight file with SHA-256 116a02532db461f91386a5b20f942ff2c8d4de7341e21b55caafc3d7b25f49a1. Verify that file and record hashes of the other five. Inspect remote file sizes before downloading; allow at most 35,000,000 bytes for the complete model/tokenizer bundle. This is an approximately 30-MB artifact, not a 30-MB software environment. Dependencies can consume substantially more space.

Do not fetch all training checkpoints, optimizer state, alternative `.bin` weights, or remote Python code. Do not silently substitute the deduplicated checkpoint or a newer branch tip. No Hugging Face account, hosted inference API, GPU, paid service, or private prompt is needed. Initial public artifact and package downloads require internet access; the measurement run should work offline.

4.2 Keep the software environment separate

Use an isolated virtual environment for this lab, outside the course website’s build dependencies. Preserve a lock file with exact tested versions of Python, PyTorch, Transformers, tokenizers, safetensors, the download client, and any measurement dependency. Record the operating system and CPU architecture.

Transformers v4.57.1 is the source/interface reference for this specification. It is not a claim that an installation has already been validated on the learner’s machine. Check a compatible supported package set, install it from official package sources, pin the working versions, and record any change from the reference. Recheck generation and cache interfaces when the version changes. Do not label an untested lock file “reproduced.”

Require these execution controls:

- CPU explicitly, with loaded floating model parameters in `torch.float32`.
- Built-in GPTNeoXForCausalLM, fast tokenizer, `trust_remote_code=False`, and safetensors only.
- Local-directory loading with `local_files_only=True` after download verification.
- Explicit `model.eval()` and `torch.inference_mode()`; no training, gradients, or optimizer.

- `attn_implementation="eager"`, no compilation, quantization, autocast, GPU, or automatic device placement.
- Fixed CPU thread configuration, preferably one intra-op and one inter-op thread for this small experiment; set before computation and record the effective values.

Inspect actual parameter devices and dtypes after loading. Record the architecture class, parameter count, unique parameter-storage bytes without double-counting shared storage, and any loading warnings. Separate checkpoint file bytes from loaded parameter bytes. If any required check fails, preserve the error and stop the dependent comparison.

Set a 15-minute execution limit after setup, a 30-second limit per generation or timing trial, and a maximum of 16 output selections per trial. These are safety bounds, not promised performance. Stop and record a timeout rather than enlarging the model or using a paid fallback.

5 Part 3 — Record what the model actually receives

Use two exact, harmless prompts:

- P1: The small red boat crossed the lake.
- P2: A notebook lay beside the window.

Supply plain text, with no chat template, role labels, system message, or added instruction. This specimen is a base model; a completion need not follow an instruction or be useful prose. A harmless prompt also cannot guarantee a harmless continuation from an unconstrained base checkpoint. Treat outputs as experimental data, and do not execute any text it produces.

Choose `add_special_tokens=False` explicitly. Record the exact UTF-8 source string, serialized string, token IDs, decoded token pieces, input length, attention mask, and special-token mapping. For this experiment the source and serialized strings should match. Verify rather than assume that relation. Store newline and whitespace characters unambiguously in JSON.

Use batch size one throughout the core experiment. Pass the attention mask. No padding is necessary. If the generation interface requires a pad ID, choose an existing tokenizer token deliberately, record its ID, and do not resize the vocabulary or add learned embeddings merely to silence a warning. Record the checkpoint's EOS ID and every stopping rule.

5.1 A small sampling matrix

Use `max_new_tokens=16`, `num_beams=1`, `num_return_sequences=1`, and `use_cache=True` for these runs. Retain normal EOS stopping and record actual generated length. Run four conditions for each prompt:

Condition	Token selection	Temperature	Top-p	Top-k
G	Greedy	Inactive	Inactive	Inactive
S1	Sampling	1.0	1.0	0, disabled
S2	Sampling	0.5	1.0	0, disabled
S3	Sampling	1.0	0.8	0, disabled

For G, perform three repetitions. For each sampling condition, reset the random state immediately before each run using seeds 17, 23, and 31. Reset any other random generators the implementation actually uses. Then rerun S1 with seed 17 once per prompt, from a fresh generation cache, to test same-seed repetition.

This yields 26 generation trials, requesting at most 416 new tokens in total. Retain every trial, including repeated outputs, early EOS, empty decoded text, incoherence, warnings, and failures. Do not generate more candidates and present only the most appealing examples.

Set other selection controls to neutral values: no repetition penalty beyond 1.0, token bans, sequence bias, forced token, minimum-length constraint, beam search, watermarking, or additional probability filter. Save the **effective complete generation configuration**, including defaults, rather than recording only this table. Avoid sample-only arguments that the installed library warns are inactive under greedy selection. The generation reference and processor implementation are the interface checks.

For every trial, save raw generated IDs separately from prompt IDs, the full returned sequence, and decoded continuation both with and without special tokens hidden. Do not use decoded character count as token count. Capture unprocessed first-step model logits from one already-budgeted trial per condition; do not substitute temperature-scaled or filtered generation scores. Copying and saving these values belongs outside the timed region and must not add an uncounted forward call. This helps establish that different sampling settings need not change the network’s initial logits.

Summarize exact-ID repeatability and distinct continuations. With three seeds, make no claim to have accurately estimated the full output distribution or identified a universally best temperature. A different seed can select the same tokens.

6 Part 4 — Test caching on matched prefixes

A cache comparison must isolate the computational path. If two free-running outputs diverge, their later logits answer different questions.

Take up to the first eight generated IDs from P1’s first greedy trial as a fixed continuation. Keep an early EOS if present and use the available shorter continuation; do not generate extra text to fill the trace. If the reference generation failed, mark this comparison unavailable instead of inventing a trace. Construct the two paths as follows:

1. **Uncached path:** pass the complete prompt plus preceding fixed continuation with `use_cache=False`, no prior cache, and its correct mask and positions. Save the last-position logits.
2. **Cached path:** run one fresh prompt prefill with `use_cache=True`; for later comparisons supply each preceding fixed token once, reusing that same returned cache. Extend masks and positions using the installed model’s documented contract. Save logits for the same logical positions.
3. Feed the same fixed continuation into both paths even when their argmax predictions differ. This is a matched-input test, not a test of free-running output quality.

For m available target tokens, compare prefixes from the prompt alone through the prompt plus $y_1, \dots, y_{m-1}$. The uncached path uses m calls; the cached path uses one prompt prefill and $m - 1$ one-token calls. Do not process the final target y_m after its preceding logits have been compared. Thus

$2m \leq 16$ model forwards suffice. If immediate EOS leaves $m = 1$, record that the test compared only prefill and did not exercise cache reuse.

Verify every compared vector has the vocabulary dimension expected from the loaded model, contains finite values, and corresponds to the same prefix. Inspect cache tensor dtype, device, shape, and logical length separately from the configuration's `use_cache` flag.

Before examining errors, declare an initial FP32 comparison criterion:

$$|a_i - b_i| \leq 10^{-5} + 10^{-4}|b_i|,$$

where b is the uncached reference. Record maximum absolute difference, the fraction of entries failing the criterion, argmax IDs, and the top-two logit margin. This is a diagnostic tolerance, not a universal guarantee about every backend. Do not silently relax it after a failure. First check masks, positions, evaluation mode, input slicing, stale caches, and implementation differences. Record any justified revised criterion separately.

Report both numerical agreement and token-choice agreement. Near a tie, small numerical differences can change argmax even if most logits are close. Compare real generated outputs from the timing trials below too, and mark their first divergence if there is one. The cache explanation and numerical-accuracy notes explain why mathematical equivalence is not a promise of bitwise identity.

7 Part 5 — Measure a defined workload repeatedly

7.1 Choose timing boundaries

Use a monotonic high-resolution wall clock. Record these phases separately:

- Artifact download or verified local reuse. Do not call local reuse a fresh download.
- Tokenizer and model loading, with declared start and end points.
- Input tokenization and output text decoding.
- Warmup, excluded from the measured sample but retained in the log.
- Prompt prefill forward time, ending when next-token logits are available.
- Local time to first selected token, including prefill and selection but excluding loading and tokenization.
- Subsequent selection timestamps and full generation-loop time.

This local time-to-first-token measure has no network or service queue. Do not label it an end-user service benchmark. If the available wrapper cannot expose internal phase boundaries, implement an instrumented ordinary generation loop and verify its selection behavior, or explicitly report the unavailable phase. Do not call a complete `generate()` duration “prefill.”

Keep expensive tensor copying, console printing, file writes, and memory inspection outside timed sections. Minimal timestamp collection still adds overhead; use the identical instrumented code for both cache conditions and disclose that overhead in the limitations.

7.2 Make length a controlled variable

Use P1 and a second timing prompt consisting of P1 repeated eight times with one separating space. Record the actual token lengths; do not assume the longer one has exactly eight times as many tokens.

For this timing experiment only, select exactly 16 greedy tokens and stop at that budget. Disable EOS termination without suppressing the EOS logit: if EOS is selected, retain it and continue the profiling loop. This deliberate fixed-length workload measures execution, not ordinary response completion. Record its different stopping rule prominently. Never pass these post-EOS strings off as normal responses.

For each prompt length, compare cache on and off. Use a fresh cache for every trial, identical dtype, eager attention, thread settings, inputs, and greedy selection. Record and hold fixed `logits_to_keep` and diagnostic-output settings across both paths. For the referenced GPT-NeoX interface, use `logits_to_keep=1` to request only the final-position vocabulary projection; verify that setting in the installed version rather than silently using full-prefix logits for one path. Disable attention/hidden-state/score collection in timing trials. Perform one warmup for each of the four prompt/cache combinations, then five measured trials per combination. Alternate cache-on and cache-off order within each prompt’s repetitions to reduce order effects. Save the exact schedule and every result.

The timing section uses four warmups and 20 measured trials: at most 384 token selections. Together with Part 3, the core cap is 50 generation trials and 800 token selections, plus Part 4’s maximum 16 matched-prefix forward calls. Do not add unrecorded “practice” runs to the measured sample.

If cached and uncached free-running IDs diverge, report that fact. Their fixed lengths still permit a limited runtime comparison, but it is no longer a strictly identical-prefix benchmark. Use the matched-prefix evidence to diagnose correctness; a separate fixed-token replay would be needed for a more tightly controlled speed comparison. Do not claim that repetition alone repairs this limitation.

7.3 Summarize without hiding variation

For each condition, report all five timings and their median, minimum, and maximum. Report prompt length and actual output length beside every number. Calculate the median local TTFT and, for 16 outputs, each run’s mean post-first-token interval:

$$\frac{t_{16} - t_1}{15}.$$

If reporting generation throughput, use $16/(t_{16} - t_0)$ and label that it includes prompt processing. Do not call its reciprocal inter-token latency. Report median throughput from the per-run values, or aggregate total tokens over total measured time; name the convention because these summaries differ.

A possible result is “no clear speed difference at this workload.” Another is a slower cached run. Both are valid observations if the procedure is correct. Do not infer performance for larger models, GPU kernels, long contexts, or concurrent serving from this tiny CPU test. The benchmarking guide provides useful checks on warmup and thread control.

8 Part 6 — Report memory honestly

Record loaded parameter-storage bytes and observed cache-storage bytes at declared prefix lengths. Count unique tensor storage where sharing or views would otherwise double-count memory. These are tensor payloads, not total process memory.

If collecting process RSS or peak RSS, identify the tool, platform, units, baseline, and whether the value is current or a process-lifetime high-water mark. Peak RSS cannot simply be reset by dropping a Python variable. A rising process peak across sequential trials does not establish the cache increment for each condition. Do not claim an isolated cache-memory delta from an uncontrolled before-and-after measurement.

Use the accounting in Lesson 4.1 to identify omitted categories: temporary activations, logits, workspace, runtime libraries, allocator retention, and loading copies. You may report those as unmeasured. An honest partial accounting is more useful than a falsely precise “total memory” figure.

9 Optional bounded extension — Inspect batching

After the core experiment, compare the two original prompts singly and as a two-row batch, then reverse the batch order. Use greedy decoding, at most 16 new tokens per row, and no more than six additional generation calls, including any warmup. Keep all records; this is a correctness demonstration, not a statistically meaningful throughput benchmark.

Establish an explicit existing pad token and left padding for this generation interface. Save token IDs, masks, positions or their preparation rule, request IDs, and output-row mapping. Compare final nonpadding prompt logits and generated IDs with singleton results. Explain any numerical differences without assuming wrong outputs belong to the wrong model.

10 Optional accelerator planning only

On an already available accelerator, a future experiment could compare supported dtypes or kernels while preserving the same inputs, output-length rule, numerical checks, and synchronized timing boundaries. It would also need device-memory measurements and a complete hardware/software record. Weight quantization would be a separate intervention with its own calibration and metadata.

This lab does not provision hardware, install accelerator stacks, open accounts, rent compute, use inference APIs, or authorize spending. No accelerator experiment is needed for completion.

11 Analytical checks

i Expected reasoning, not run results

- A full 16-token ordinary generation uses one prompt prefill and 15 later forwards. First-token selection is not an additional model forward.
- With logits $(\log 8, \log 4, \log 2, 0)$, temperature one gives $(8, 4, 2, 1)/15$. Temperature one-half gives $(64, 16, 4, 1)/85$.
- Under the lesson’s smallest-prefix-reaching-the-threshold convention, temperature one-half followed by top- $p = 3/4$ leaves only the first token. Applying the nucleus first leaves two candidates with final probabilities $4/5, 1/5$.
- Different seeds need not produce different outputs. A distribution can concentrate on the same choices, and a filter may leave a single candidate.

- Parameter bytes omit cache, temporary tensors, workspace, and runtime overhead. A weight-file size is not a runtime memory measurement.
- Neither “cache on is faster” nor “all logits are bitwise identical” is a required empirical outcome. Correctness and timing must be evaluated separately.

12 Submission checklist

Include:

1. Original predictions and a separate revised explanation.
2. Complete experiment source, its file hash or revision, exact commands, environment lock, hardware/thread record, and warnings.
3. Checkpoint revision, allowlisted file hashes and sizes, loading options, observed devices/dtypes, and the full effective generation configuration.
4. Exact source and serialized inputs, token IDs, masks, seeds, stopping rules, output IDs/text, and all trial records.
5. Cache-prefix alignment checks, tolerance, numerical errors, token agreement, and cache shapes/bytes.
6. Timing boundaries, schedule, warmups, all measured samples, declared summaries, and any unmeasured quantities.
7. One supported conclusion about selection, one about computation, and two limitations. Do not infer truthfulness or general assistant quality from the continuations.

A clear failure report with its evidence is preferable to invented timings or a silently changed checkpoint.

13 More Learning

- Pinned Pythia artifact commit: verify the specimen and weight checksum.
- Transformers generation configuration: inspect effective selection and stopping controls.
- PyTorch reproducibility: understand what seed control can establish.
- PyTorch benchmarking: study repeated measurements and runtime-aware timing.