

Compare Training Evidence

Gavin · AI-assisted course design

Contents

1	Goal	3
2	Part 1 — Record predictions first	3
3	Part 2 — Establish the source manifest	3
	3.1 Base specimen	3
	3.2 Instruction-tuned specimen	4
	3.3 Optional bounded source helper	4
4	Part 3 — Build an evidence matrix	4
5	Part 4 — Separate the package from its weights	5
6	Part 5 — Design a benign behavioral comparison	6
	6.1 Define two different comparisons	6
	6.2 Optional actual model comparison	7
7	Part 6 — Explain what the comparison could establish	7
8	Optional CPU arithmetic experiment	7
	8.1 Run the optional completed arithmetic reference	8
9	Reference checks	8
10	Submission rubric	9
11	More Learning	9

1 Goal

Investigate a documented base/instruction-tuned pair without loading either model. Build a defensible account of what changed, what the sources establish, and what remains unknown. Then design a benign behavioral comparison that could test your predictions later.

Prerequisite: Lesson 3.2 — Post-Training, Alignment, and Safety Learned into Weights, including its distinction between documentation, observation, and inference.

Time: about 60–90 minutes. The optional arithmetic experiment adds 20–30 minutes.

Requirements: a browser or saved public source copies, a text editor, and ordinary arithmetic. The optional experiment uses an existing Python 3 installation and its standard library. The core requires no account, paid API, GPU, model weights, tokenizer vocabulary download, or machine-learning package.

Expected artifact: `submission.md` with a source manifest, evidence matrix, predictions, comparison protocol, and a short conclusion. Preserve any optional code and its actual output separately.

! Documentation is not an inference run

Do not present imagined answers as model outputs. Complete the Lab by reading small text files; ignore model-loading commands and deployment buttons. Running a model is an optional, separately resourced extension, not a requirement for a complete submission.

2 Part 1 — Record predictions first

Before consulting the reference answers, write your predictions about these statements:

1. The instruction-tuned model must have more Transformer layers than the base.
2. A chat template proves that a checkpoint received assistant training.
3. Using each package's defaults produces a controlled weights-only comparison.
4. A family-level training report must identify the exact training data for each released checkpoint.
5. An instruction-tuned model will answer every harmless prompt correctly.

Give a reason and confidence level for each. Keep the original predictions when you revise them.

3 Part 2 — Establish the source manifest

The specimens are **Qwen/Qwen2.5-0.5B** and **Qwen/Qwen2.5-0.5B-Instruct**, official Qwen repositories. They are deliberately fixed specimens, not a claim that this family is the newest available model.

Use the exact revisions below. Read or save only the named small text files. A repository page may include dynamically generated usage examples outside the versioned file; those are not part of the pinned evidence.

3.1 Base specimen

Revision: 060db6499f32faf8b98477b0a26969ef7d8b9987

- `README.md`: checkpoint-specific description and training-stage statement
- `config.json`: architecture and model settings

- `tokenizer_config.json`: special tokens and chat serialization
- `generation_config.json`: packaged generation defaults
- `LICENSE`: repository license

3.2 Instruction-tuned specimen

Revision: 7ae557604adf67be50417f59c2c2f167def9a775

- `README.md`
- `config.json`
- `tokenizer_config.json`
- `generation_config.json`
- `LICENSE`

Also read Section 4 of the Qwen2.5 Technical Report, version 2. Use the actual Qwen2.5 report rather than assuming that the older paper identifier displayed in repository metadata describes this release’s recipe.

For every source, record its title, exact URL, revision or paper version, access date, and sections or keys inspected. The two repository license files identify Apache License 2.0. If you retain copies, retain the applicable license and notices; do not assume the paper has the same license. This Lab does not redistribute checkpoint files or third-party diagrams.

If a link fails: try the repository’s pinned file view or save the small text source using its Raw link. Do not replace the revision with `main` without documenting the change. Do not download `model.safetensors`, `tokenizer.json`, `vocab.json`, or `merges.txt` for the core.

3.3 Optional bounded source helper

The standard-library text-source helper retrieves only the ten named files above, including both license files, at their fixed revisions. It checks exact byte sizes and SHA-256 fingerprints, retains original bytes, and writes a source manifest. The total is 47,706 bytes. It does not download model weights or tokenizer vocabulary, execute templates, or run inference.

```
python3 read_training_evidence.py --plan
python3 read_training_evidence.py --output training-evidence
python3 read_training_evidence.py --offline --output training-evidence
```

Use a dedicated directory. Altered or unexpected files fail verification. Reading the saved text and paper is still your evidence-handling task; the helper does not supply model-performance results.

Paper-only alternative: if the checkpoint pages are unavailable or require access you do not have, read the public Qwen2.5 report and the InstructGPT paper, version 1, Sections 3 and 5. Complete the training-method and evaluation portions using those papers. Mark checkpoint fields and serialization questions “not directly verified.” Use the supplied reference values below only as course-provided data. This alternative can earn full credit for evidence handling; it is not direct inspection of the pair.

4 Part 3 — Build an evidence matrix

Use these columns:

Claim	Evidence and exact location	Scope	Category	What it does not establish
Example: a setting is present	A pinned file and key	One package revision	Configuration	That every runtime uses it

Use five categories:

- **Documentation:** an author’s statement about training, ancestry, intended use, or evaluation
- **Configuration:** a value or template present in the named file
- **Derived:** your calculation or inference from explicitly identified evidence
- **Observed:** something measured in an actual run, with its setup recorded
- **Unknown:** a question these inspected sources do not settle

Reading a configuration is an observation of a file, but use **Configuration** here so that it cannot be mistaken for observed model behavior.

Include at least twelve rows addressing:

1. The base-model relationship stated in the Instruct card’s metadata
2. The checkpoint-specific training-stage descriptions
3. Architecture class, layer count, hidden width, attention heads, and KV heads
4. Whether those principal dimensions match
5. At least two differing configuration fields
6. Whether each tokenizer configuration contains a chat template
7. Whether each template inserts a default system message if one is absent
8. Whether its template describes tool-call serialization
9. The saved sampling choice and stop-token settings
10. Training methods described by the family report
11. Which training details are linked specifically to this released revision, and which are not
12. Actual performance on your planned prompts

Split rows when one statement combines categories. “The card says post-training, therefore I observed better instruction following” is not an acceptable row. A field absent from one file is not automatically false at runtime: defaults and implementation matter.

For the training account, identify the signal and updated object at each stage. Distinguish demonstrated responses, preferred/dispreferred pairs, and reward-scored policy samples. The report describes offline DPO and online GRPO; preserve its terminology while explaining that the procedures are different. Do not rewrite every reference to RL as PPO.

5 Part 4 — Separate the package from its weights

Write a 150–250 word explanation of why a comparison using the two packages “as shipped” is not a clean intervention on weights alone.

Address these questions:

- If principal architecture dimensions match, must learned parameter values match?

- Does finding a tool-call branch in a template prove reliable tool use or actual tool access?
- If generation defaults differ, which output differences might arise without invoking training as the explanation?
- Does changing a system message or stop rule retrain the model?
- What additional evidence would establish a checkpoint-specific recipe rather than a family description?

Inspect the complete default-message strings but paraphrase their difference in your submission. You do not need to copy the whole template or execute it. Explain where role separators, the generation prefix, and the end-of-turn marker fit in the eventual token sequence.

6 Part 5 — Design a benign behavioral comparison

Use the same four user requests for both specimens:

1. “Return exactly the lowercase word blue, with no punctuation.”
2. “Sort these integers in ascending order and return only a JSON array: 9, 2, 5.”
3. “The note says: Mira packed three apples and two pears. According to the note, how many pears did Mira pack? Answer with one digit.”
4. “The note says: Mira packed three apples and two pears. According to the note, what color was the bag? If the note does not say, answer unknown.”

Before running anything, predict for each specimen:

- the relative likelihood of following the exact format;
- plausible failure types;
- whether you expect task accuracy and format compliance to move together;
- the strength of the evidence supporting your prediction.

A prediction may be “uncertain.” Do not invent exact responses or percentage success rates from a model card. Do not add dangerous requests to test refusal; this Lab’s safety-related skill is evidence discipline and appropriate uncertainty.

6.1 Define two different comparisons

A — Package comparison: use each checkpoint’s intended documented input preparation and saved defaults, while keeping the user requests fixed. The base card does not recommend a conversational recipe; label any chosen completion wrapper for it as an experimenter choice. This asks how the two configured packages behave under the stated preparation. Record all differing settings and any inserted system messages.

B — More controlled comparison: specify a common textual completion prefix, equal generation budget, decoding rule, stopping interpretation, and precision/runtime. Record the exact serialized inputs and, if actually running, token IDs. This reduces some confounds but may put one checkpoint in a less suitable format. It does not magically measure context-independent capability.

A stronger design crosses checkpoint and input format: each checkpoint receives a plain completion format and an explicitly specified chat format. That lets you inspect a checkpoint-by-format interaction

instead of attributing every difference to training. If tokenizer mappings differ, document them rather than claiming identical strings are necessarily identical token inputs.

Define scoring before viewing outputs. For the four requests above, the canonical exact outputs are blue, [2,5,9], 2, and unknown. For task 2, valid JSON with different whitespace should count as content/structure success; record whether you separately require exact text. Score:

- task correctness;
- instruction/format compliance;
- unsupported information;
- successful termination within the budget.

State a sample count per condition. Greedy decoding provides a useful fixed decoding rule but repeating it is not equivalent to independent random trials. For sampled decoding, preserve seeds and settings, report all attempts, and avoid strong population claims from four hand-written prompts. Add held-out paraphrases only after documenting how they were selected.

6.2 Optional actual model comparison

Proceed only if you already have a suitable runtime and sufficient resources, or separately choose to set them up. Record model/tokenizer revisions, runtime version, hardware, precision, serialized inputs, tool access, and all generation settings. Load one checkpoint at a time if appropriate. A “0.5B” label does not specify working memory or guarantee fast CPU inference.

Do not substitute a hosted assistant of unknown provenance and call it the base model. If you cannot run the pinned pair, submit predictions and the protocol with the runtime results marked **not run**. That is a complete core submission.

7 Part 6 — Explain what the comparison could establish

Write a conclusion of 200–300 words answering:

1. Which facts are directly documented for this pair?
2. Which proposed behavioral differences remain predictions?
3. Which exact training or data details remain unknown?
4. Why would one refusal, one fluent answer, or one error not identify a training method?
5. What is the smallest additional experiment or source that would improve your conclusion?

Include one tempting overclaim and rewrite it. For example, replace “The instruction-tuned checkpoint is safer” with a statement that identifies the actual evaluated behavior, test set, scoring rule, and uncertainty. If you ran nothing, your conclusion must say so.

8 Optional CPU arithmetic experiment

This miniature experiment trains a two-response probability model, not an LLM. It requires no external downloads. It tests the mechanics of a loss rather than assistant capability.

Let a single parameter z define $p(A) = \sigma(z)$ and $p(B) = 1 - p(A)$. Start at $z = 0$. Use a fixed reference with probability $1/2$ for each response, one label preferring A, and a learning rate of 0.1.

Implement 20 gradient-descent updates for each objective:

- SFT on A: $L = -\log \sigma(z)$, with derivative $\sigma(z) - 1$
- DPO on A over B: $L = -\log \sigma(\beta z)$, with derivative $\beta[\sigma(\beta z) - 1]$

Run DPO once with $\beta = 1$ and once with $\beta = 1/2$. Use stable scalar functions such as `math.log1p` where useful. Print the step, parameter, both probabilities, loss, and reference-relative margin. Check that probabilities sum to one and the first update increases z .

Predict first: with $\beta = 1$, the two objectives and updates coincide in this special two-outcome model. Explain why that does not make SFT and DPO equivalent for arbitrary language-model datasets. SFT sees a target; DPO depends on a comparison and a reference. This construction makes those distinctions algebraically collapse.

At step zero both losses are $\log 2$. The first SFT update gives $z = 0.05$; the first DPO update with $\beta = 1/2$ gives $z = 0.025$. These are analytical checks, not a report that your program ran. Retain your actual output and investigate any disagreement.

8.1 Run the optional completed arithmetic reference

After recording your predictions, download `compare_scalar_losses.py` and run it locally:

```
python3 compare_scalar_losses.py --output scalar-results.json
```

Use a new filename; an existing output is rejected. The JSON contains all 21 snapshots from step zero through 20 for SFT and both DPO settings. Inspect the derivatives and checks rather than treating this scalar exercise as an LLM behavioral evaluation.

9 Reference checks

Read after preserving your initial work.

- The Instruct metadata explicitly names Qwen/Qwen2.5-0.5B as its base. The cards identify pretraining for the base and pretraining plus post-training for Instruct. This is documented ancestry, not a checksum-level training log.
- Both configs specify Qwen2ForCausalLM, 24 layers, hidden width 896, 14 query heads, and two KV heads. Those facts establish matching named dimensions, not identical parameter values.
- The config end-token ID is 151643 for the base and 151645 for Instruct. `max_window_layers` also differs, while both set `use_sliding_window` false. Do not infer active sliding-window behavior from the first field alone.
- Both tokenizer configurations contain chat templates, including tool-call handling. Their default system strings differ. This is a direct counterexample to “chat template implies instruction-tuned checkpoint.”
- The base generation file sets `do_sample` false; Instruct sets it true with temperature 0.7, top-p 0.8, and top-k 20. Instruct also records a repetition penalty of 1.1. Runtime overrides can change all of these.
- In the generation files, the base has end-token ID 151643; Instruct accepts 151645 and 151643. Keep generation settings separate from the model config’s single end-token field.
- Section 4 describes SFT, offline DPO, and online GRPO. It does not provide a full data manifest and optimizer-state lineage tied to these repository hashes.

- No core-Lab step measures outputs. Any statement that one model followed these prompts better must remain a prediction unless accompanied by a real run record.

10 Submission rubric

Area	Points	Full-credit evidence
Source identity	4	Exact revisions or paper-only route, access dates, inspected locations, license distinction
Evidence matrix	5	At least twelve well-scoped rows; unknowns and documentation separated from runtime observation
Experimental design	5	Saved predictions, four benign tasks, specified formats/settings, predeclared scoring and limitations
Interpretation	4	No recipe inferred from a name; no safety guarantee or hidden-capability claim from an output
Revision and clarity	2	Original predictions retained, corrections explained, run/not-run status explicit

The optional arithmetic experiment is additional practice, not required for the 20-point core. A careful paper-only submission can receive full credit by making its evidence limitations explicit.

11 More Learning

- Revisit the DPO paper’s Section 4 after the optional experiment and locate the reference-relative log ratio.
- Compare the DeepSeek-R1 report’s training stages with Qwen’s reported stages. Similar assistant behavior would not make the recipes identical.