Correctness Testing for Systems Algorithms

Deep Dive into SGLang — Chapter 34

Yin Yang

Foundation Books Project

Where this chapter sits

This is the chapter that makes an *optimized* serving path safe to change.

- A faster scheduler, kernel, cache, or gateway is worthless if it quietly breaks request semantics.
- Ordinary output comparison is too weak: two paths can emit the same text while disagreeing about hidden runtime state.
- So correctness testing fixes a request, chooses an *oracle*, and records the *state evidence* that decides whether semantics were preserved.

Goal: by the end you can look at a test and say which request, mode, state ledger, output quantity, and tolerance it actually compares.

Roadmap

The correctness problem

The vocabulary that recurs

What to carry forward

The correctness problem

The failure a benchmark would miss

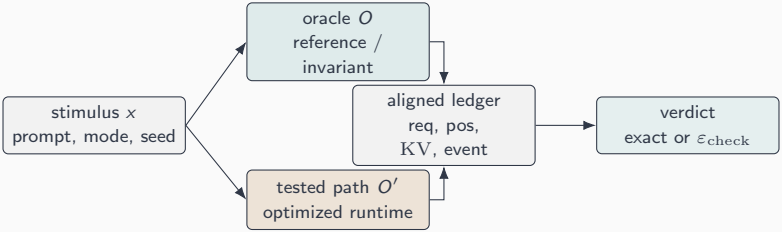
Request `qwen3-r1`: *"The capital of the United Kingdom is"*, temperature 0.

Field	Reference path	Optimized path
visible token	`London`	`London`
logical position	12	11
KV owner	`qwen3-r1`	another request
finish reason	`length`	`length`

The user sees a plausible answer. The test sees a broken serving transition. Final string comparison passes; the state ledger does not.

The oracle comparison harness

Every test in this chapter has the same shape.



Fix the stimulus, mode, ledger, and tolerance *before* reading the verdict. The final string is only one compared field.

The vocabulary that recurs

Five things every correctness test names

Oracle O the reference, invariant, or protocol expectation that issues the verdict.

Ledger $\mathcal{L}$ hidden state — position, KV owner, route, stream order — compared alongside the text.

Tolerance $\varepsilon_{\text{check}}$ exact equality, an `atol`/`rtol` band, or a required event order.

Alignment proof that both paths filled the ledger from the *same* semantic object.

Diagnostic D the record that localizes a regression to one surface.

A test that compares only "is the output nonempty?" names none of these — and catches almost nothing.

Coverage is distributed because state is distributed

Four suites protect four layers of serving state:

- **Registered runtime tests** — pin models, flags, workers, and fixtures (`test/registered/`).
- **Python test utilities** — launch servers, poll, send, compare to a reference (`python/sglang/test/`).
- **Kernel tests** — tensor shapes, dtypes, layouts, tolerances (`sgl-kernel/tests/`).
- **Gateway E2E tests** — API status, event order, tool calls, routing topology (`sgl-model-gateway/e2e_test/`).

The nearest precise oracle localizes the bug; an integration anchor proves the rule survives real topology.

From concept to code: the reference comparison

The chapter's oracle idea has a concrete home in the model-runner comparison.

from `python/sglang/test/runners.py`

```
def check_close_model_outputs(hf_outputs, srt_outputs,
        prefill_tolerance, decode_tolerance, rouge_l_tolerance, ...):
    rouge_l_scores = calculate_rouge_l(hf_outputs.output_strs,
                                       srt_outputs.output_strs)
    assert all(score >= rouge_l_tolerance for score in rouge_l_scores)
    ...
    assert torch.all(abs(hf_logprobs - srt_logprobs) < prefill_tolerance)
```

Three tolerances, one request: text overlap (ROUGE-L), prefill logprobs, decode logprobs — each an $\varepsilon_{\text{check}}$ tied to the same compared object.

What to carry forward

What to carry forward

1. Correctness testing changes the **evidence surface**; it does not loosen the runtime contract.
2. Two paths can share visible text and still **diverge in state** — position, KV owner, route, event, or number.
3. A useful **oracle** names request, mode, ledger, output quantity, and tolerance.
4. **Alignment** comes before the verdict: same object on both paths.
5. A test **inventory** is evidence only after each count is tied to an invariant, oracle record, fixture signal, and skip policy.

A benchmark asks how fast.

A correctness test asks whether it is still the same request.