System Verification and Validation Plan for Tandem

Team 11, Tandem
Justin Kwinecki
Zayed Umer
William Pol
Manav Patel
Musab Mondal

November 2, 2026

Revision History

Table 1: Revision history

Date	Version	Notes
Nov 2, 2026	0.1	First version, written against SRS version 0.2 and the Hazard Analysis.
Nov 2, 2026	0.2	Complete rubric coverage, fixtures, acceptance gates and learning plans.

Contents

1 Symbols, Abbreviations, and Acronyms **v**

2 General Information **1**
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 Custom Documents . . . 2
- 2.4 Relevant Documentation . . . 2

3 Plan **3**
- 3.1 Verification and Validation Team . . . 4
- 3.2 SRS Verification . . . 4
- 3.3 Design Verification . . . 5
- 3.4 Verification and Validation Plan Verification . . . 6
- 3.5 Implementation Verification . . . 6
- 3.6 Automated Testing and Verification Tools . . . 6
- 3.7 Software Validation . . . 7

4 System Tests **7**
- 4.1 Tests for Functional Requirements . . . 9
  - 4.1.1 Request Input . . . 9
  - 4.1.2 Cache Checks and Loading . . . 11
  - 4.1.3 Fallback and Failure Handling . . . 12
  - 4.1.4 Request Lifecycle . . . 14
  - 4.1.5 Configuration and Records . . . 15
  - 4.1.6 Isolation and Privacy . . . 16
- 4.2 Tests for Nonfunctional Requirements . . . 17
  - 4.2.1 Accuracy . . . 17
  - 4.2.2 Performance . . . 18
  - 4.2.3 Setup, Documentation and Reports . . . 19
- 4.3 Traceability Between Test Cases and Requirements . . . 20

5 Unit Test Description **21**
- 5.1 Unit Testing Scope . . . 22
- 5.2 Tests for Functional Requirements . . . 22
- 5.3 Tests for Nonfunctional Requirements . . . 22
- 5.4 Traceability Between Test Cases and Modules . . . 22

6 Appendix **23**
- 6.1 Symbolic Parameters . . . 23
- 6.2 Setup Walkthrough Record . . . 24

7 Appendix: Generative AI Use Disclosure **25**
7.1 AI Tools Used . . . 25
7.2 How and Where Used . . . 25
7.3 Verification of AI-Generated Content . . . 25

8 Appendix: Reflection **26**
8.1 What Went Well . . . 26
8.2 Difficulties and Resolutions . . . 26
8.3 Skills and Learning Approaches . . . 26

List of Figures

1 Verification order. Failed prerequisites block qualification. . . . 3

List of Tables

1 Revision history . . . ii
2 Symbols and abbreviations . . . v
3 Verification responsibilities . . . 4
4 Requirements and the tests that verify them . . . 20
5 Symbolic parameters . . . 23

1 Symbols, Abbreviations, and Acronyms

Table 2: Symbols and abbreviations

Symbol	Description
CI	Continuous integration
F1	Harmonic mean of precision and recall over answer words
FR, NFR	Functional and non-functional requirement (SRS)
GPU	Graphics processing unit
HA	Hazard Analysis (**?**)
HF	Hugging Face (Transformers library and model hub)
K, V	Key and value tensors of the KV cache
KL	Kullback–Leibler divergence between two next-token distributions
KV cache	Key-value cache: per-layer attention keys and values for processed tokens
MG, MIS	Module Guide and Module Interface Specification (**?**)
PoC	Proof of concept
SR	Safety or security requirement (SRS and HA)
SRS	Software Requirements Specification (**?**)
T	Test (test IDs start with T-)
TTFT	Time to first token

Other terms follow the SRS glossary. Symbolic constants, written in `TYPEWRITER_CASE`, are defined with their values in Appendix 6.1.

2 General Information

2.1 Summary

The software being tested is Tandem. When a semantic router switches a conversation from one large language model to another, the new model normally has to process (prefill) the whole conversation history again before it can answer. Tandem instead predicts the new model's KV cache from the previous model's KV cache with a trained translator, checks the predicted cache, and loads it, so the new model only processes the latest turn. If the predicted cache is unavailable or fails a check, Tandem falls back to normal prefill.

For the proof of concept, the source model is SmolLM2-360M-Instruct, the target model is Qwen2.5-0.5B-Instruct, and translation runs in one direction (source to target). The translator is trained on paired KV caches that both models produce for the same HotpotQA passages and questions (**?**). A small custom router chooses the model, and an inference runner built on Hugging Face Transformers runs it. Integration with vLLM is planned after the proof of concept.

2.2 Objectives

In order of priority, and following the goals in the SRS (**?**, Section 1.2) and the hazards in the HA (**?**, Section 5), this plan aims to build confidence that:

1. **Caches that fail checks never reach generation.** Every predicted cache that fails a check, and every unsupported case, falls back to normal prefill and records why (FR-04, FR-06, FR-07, SR-02). This is the most important quality, because a bad cache gives a fluent but wrong answer with no visible error (HA hazard H8).

2. **Translation accuracy is measured honestly** against normal prefill on held-out data that was not used for training or checkpoint selection (FR-14, FR-16, SR-09). This is the main risk the PoC must address.

3. **Time savings are measured including all overhead** (translation, checks and loading), with the same requests and settings in both arms (FR-11, NFR-04).

4. **Cached history is never shared between users without permission** (SR-01, SR-04, SR-12).

Out of scope.

- **External libraries** (PyTorch, Hugging Face Transformers, vLLM, the NVIDIA prompt classifier) are assumed to have been verified by their own teams. We test only how Tandem uses them.

- **Usability testing with end users.** Tandem has no user interface; its users are technical operators. Setup and documentation are checked by a structured walkthrough instead (T-DOC-1).

- **Factual correctness of model answers.** The reference is the target model's own output with normal prefill, not the true answer (SRS, Potential User Problems).
- **Router quality.** Whether the router picks the best model for a prompt is outside the SRS scope (A2). The router is tested only for returning a valid model and falling back when its classifier fails.
- **Multi-GPU and multi-server execution**, which the SRS leaves for future work (UC2).
- **Money cost.** GPU time is measured; cost in dollars is reported only if rates are known (SRS, Costs).

2.3 Custom Documents

Tandem's custom documents are a **Machine Learning (ML) Model Report** in the fall term and a **Performance Report** in the winter term.

- **ML Model Report** (**?**). It describes the translator, the paired dataset, the training procedure and the accuracy evaluation. The system test T-QUAL-2 defines the end-to-end pass limit; the report adds the component-level evaluation of the translator, which is not a system test:
  - the relative error per target layer ℓ, separately for keys and values, $e_\ell = \|\hat{K}_\ell - K_\ell\|_F / \|K_\ell\|_F$, where $\hat{K}_\ell$ is the predicted and K_ℓ the native tensor and $\|\cdot\|_F$ is the Frobenius norm (the absolute error is reported when $\|K_\ell\|_F = 0$), plotted against layer for the translator and an affine baseline;
  - a check that the training, selection and test splits share no source document;
  - two metamorphic checks that need no native reference: extending a history does not change the predicted cache at earlier positions, and predicting twice gives identical tensors.

  The report also presents the results of T-QUAL-2, their analysis and the failure cases. The report improves the project because the translator is its riskiest part, and it forces the team to justify the data splits, baselines and metrics before claiming any result.
- **Performance Report.** It will present the time, throughput and memory measurements of T-PERF-1 and T-PERF-2, with configurations, repeated trials and variation. The V&V Report will reference it when deciding whether NFR-04, NFR-07 and NFR-08 are met.

2.4 Relevant Documentation

- **Problem Statement and Goals** (**?**). States the goals that the objectives above are drawn from, and the custom documents.

- **SRS** (**?**). Defines the requirements every system test traces to, including the accuracy rules and the timing method (warm-up runs, paired trials and bootstrap confidence intervals) that the tests reuse.
- **Hazard Analysis** (**?**). Adds SR-09 to SR-14 and their fit criteria; the fallback, isolation and memory tests below implement those fit criteria.
- **Development Plan** (**?**). Lists the tools, coding standard and CI, which Section 3.6 refers to rather than repeating.
- **MG and MIS** (**?**). Not yet written. The unit tests in Section 5 will be derived from them.
- **ML Model Report** (**?**). Reports the results of the translator accuracy tests.
- **Same-model cache replay results** (`src/experiments/LOCAL_CACHE_RESULTS.md`). An experiment already run on Qwen2.5-0.5B-Instruct that measured TTFT for normal, warm and file-restored caches and checked that outputs were identical. It is the starting point for T-QUAL-1 and T-PERF-1.

Assessment basis. This revision follows the VnV Plan–Main, LO_Plan and LO_Reflect rubrics retrieved through Brightspace MCP for Avenue course 781906 (assignments 364168–364170). It supplies a plan, not passing test results. Detailed unit tests remain planned until the MIS, as recorded in issue 60.

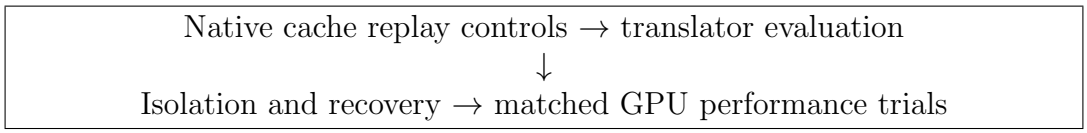


Figure 1: Verification order. Failed prerequisites block qualification.

3 Plan

This section describes who carries out verification and validation, how the documents are reviewed (SRS, design and this plan), how the implementation is verified, which tools are used, and how the team checks that it is building the right product.

3.1 Verification and Validation Team

Table 3: Verification responsibilities

Member	Verification role
William Pol	Owner of this plan and its traceability table. Leads SRS verification (Section 3.2), as he did not write the SRS. Writes and runs the fallback tests (T-FB-1 to T-FB-3) and cache memory test (T-LIFE-2) from the HA.
Zayed Umer	Same-model accuracy (T-QUAL-1) and the translator's component-level evaluation. Lead author of the ML Model Report.
Justin Kwinecki	Router and runner tests: input handling (T-IN-1 to T-IN-3), cache loading (T-CACHE-1 to T-CACHE-3), request lifecycle (T-LIFE-1) and configuration (T-CONF-1). Maintains the code CI workflow.
Manav Patel	Evaluations for the PoC demonstration: cross-model accuracy (T-QUAL-2), whose results feed the ML Model Report. Performance tests (T-PERF-1, T-PERF-2) and the reproducibility rebuild (T-DOC-2). Lead author of the Performance Report.
Musab Mondal	Isolation and privacy tests (T-ISOLATION-1 to T-ISOLATION-3), which test the security requirements he wrote in the SRS, and records (T-REPORT-1). As SRS author, answers the SRS review issues and applies the resulting changes.
Manoj Govindassamy (advisor, Cohere)	Validates the requirements and results from the inference-serving side (Sections 3.2 and 3.7).
Peer team and TA	Review the SRS, design and this plan, and file issues in the repository.

The setup walkthrough (T-DOC-1) is done by a member who did not write the README. Every test is reviewed by a second member who did not write it, and every pull request needs one approving reviewer other than its author.

3.2 SRS Verification

1. **Peer feedback received.** Team 1 filed issues #11–#16 and #38–#43. These issues remain open at this revision. Each will be answered in the repository and closed with a link to the commit that resolves it, or with a reason it is not changed.

2. **Checklist review.** William Pol, who did not write the SRS, goes through the course SRS checklist and, for every requirement, records in one issue whether it (a) is unambiguous, (b) has a fit criterion with a value or a named constant, and (c) has at least one test in Table 4. A requirement without a test is either given one or reworded.

3. **Task-based inspection.** Before reading this plan, the reviewer writes how they would test FR-16, NFR-04 and SR-13 from the SRS alone. Differences from this plan's tests point to ambiguous wording.

4. **Advisor review meeting** (30 minutes, recorded with the supervisor meeting issue template). The team presents the goals, FR-16, NFR-04, the proposed split of NFR-05 and the PoC results, then asks:

   (a) Is a router switching models in the middle of a conversation a realistic case in production serving?

   (b) Is an answer-quality limit (`QUALITY_LOSS_LIMIT`) an acceptable replacement for exact output match in the cross-model case?

   (c) What TTFT reduction would make the approach worth integrating?

   (d) Are the excluded items (router development, multi-server execution) the right ones to leave out?

   Answers that change a requirement are recorded as issues linked to the SRS.

3.3 Design Verification

The MG and MIS will be reviewed by the peer team, as for the other documents, and by an internal checklist review that checks that:

- every requirement in Table 4 maps to at least one module;
- each module hides one design decision, such as the cache format, the translator or the routing policy;
- the interface between the runner and the translator states its input (source cache, model pair, token alignment), its output (a target cache, or a reason it cannot translate), and that the runner, not the translator, performs the fallback to prefill;
- token positions, rotary position embedding (RoPE) handling, data type and device are specified at every point where a cache changes hands.

The team will also walk the advisor through the model-switch sequence (route, translate, check, load, prefill the suffix, decode) before Rev 0.

3.4 Verification and Validation Plan Verification

- **Peer review** of this plan by another team, due November 5.
- **Checklist review** against the course VnV Plan checklist by a member who did not write the section.
- **Seeded faults.** To check that the tests can actually fail, a member deliberately breaks one behaviour at a time on a throwaway branch: skipping the non-finite value check, dropping the last cached token, ignoring the sharing group, and appending output after a restart. The tests T-CACHE-1, T-CACHE-3, T-ISOLATION-1 and T-FB-2 must each fail on their seeded fault.

3.5 Implementation Verification

Dynamic verification uses the system tests in Section 4 and the unit tests planned in Section 5. Static verification uses:

- **Linting** with `ruff` in CI on every pull request.
- **Code review** on every pull request, by a reviewer other than the author, who checks: tests added or updated; model and dataset revisions pinned; tensor shapes, positions, data types and devices checked where caches are built or loaded; no prompts, caches or credentials committed; documentation updated.
- **Code walkthrough** before Rev 0 of the cache handoff path. The author explains the code line by line to two members, who check: the cache length equals the number of processed tokens (no token dropped or repeated); the attention mask covers cached and new tokens; RoPE is applied once, at the target model's positions; fallback is taken before any output on every failure path.

3.6 Automated Testing and Verification Tools

The language, linter, test framework, coverage tool and coding standard were chosen in the Development Plan (**?**, Sections 10 and 12) and are summarized here; the performance and analysis tools are added for this plan.

- **Language and environment:** Python 3.12, with dependencies pinned per script through `uv` lock files.
- **Test framework:** `pytest`, used as the runner for both the end-to-end system tests and the unit tests. It also runs the existing `unittest`-style tests in `test/`.
- **Linter and coding standard:** `ruff` for linting and formatting; PEP 8 as the coding standard.
- **Coverage:** `coverage.py` through `pytest-cov`, reporting line and branch coverage per module in CI. No target is set before Rev 0; after Rev 0 the cache-check and fallback modules must reach full branch coverage.

- **Planned CI:** GitHub Actions will run `ruff` and the tests that need no GPU or model download (routing, classifier scoring, synthetic translator tests) on every pull request. GitHub's runners have no GPU, so GPU tests run on allocated department GPUs; commit commands, manifests and summary results, and keep large outputs on Hugging Face. Current CI builds documentation; it does not yet establish these Python checks.
- **Performance tools:** `time.perf_counter` with `torch.cuda.synchronize` for timing; `torch.cuda.max_memory_allocated` for memory; the PyTorch Profiler and NVIDIA Nsight Systems to explain slow results.
- **Analysis:** NumPy for bootstrap confidence intervals with a recorded seed; Matplotlib for the plots named in each test.

3.7 Software Validation

Validation checks that the requirements describe something worth building.

- **Advisor review** of the requirements, as described in Section 3.2.
- **Rev 0 demonstration to the advisor** shortly after the scheduled Rev 0 demo. The team shows a model switch with and without Tandem and asks whether the measured TTFT reduction and quality loss would be acceptable in practice. Feedback is recorded as issues.
- **Public data.** HotpotQA (**?**) supplies the passages and questions, so others can repeat the accuracy tests.

4 System Tests

This section lists the system tests. Tests of functional requirements come first, grouped by area, followed by tests of non-functional requirements. Each test traces to the requirements in Table 4.

All tests are system level, as advised by the TA. Each test drives Tandem only through its external interfaces: requests sent to the runner (or, from Rev 0, the serving engine), the configuration, cancellation, and stored cache files. Each test observes only what Tandem returns or writes: output tokens, diagnostic next-token logits and log-probabilities, final statuses, request records, logs, exports and public cache-ownership metrics. GPU process memory is also recorded with `nvidia-smi`. No test calls internal functions, patches code or inspects tensors inside the process. Faults are introduced through a fault-injection setting in the test configuration, or by editing stored files. Checks of the translator as a component are reported in the ML Model Report, and unit tests are planned in Section 5.

Automated tests are end-to-end: each one starts Tandem as a whole and talks to it as a client would. They are written with `pytest`, which is only the test runner. The unit tests in Section 5 also use `pytest`, but call single modules directly.

Tests that need only the runner and the models (accuracy, cache loading and T-PERF-1) run for the PoC; tests marked as needing the serving integration run from Rev 0.

Common setup. Unless a test says otherwise, it uses:

- **Setup S**: source model SmolLM2-360M-Instruct, target model Qwen2.5-0.5B-Instruct, both at pinned revisions, 32-bit floating point, eager attention; the translator checkpoint under test, recorded by version; prediction enabled; greedy decoding.
- **Test set**: `TEST_PROMPTS` HotpotQA prompts (passage plus question) that share no source document with the prompts used to train the translator or to select its checkpoint, truncated to each length in `CONTEXT_LENGTHS`.
- **Planted-code fixture (F1)**: the prompt from the existing replay experiment. Project Cedar release-code notes padded with fixed filler to each length in `CONTEXT_LENGTHS`, with the code placed early, middle or late, followed by the question "What is the code?". It gives a small, readable check that history was carried over. The native output, not the planted code, is the oracle, since the native model itself misses the code in one of the nine cases.
- **Native reference**: the same request sent with prediction turned off, so the target model uses normal prefill.
- **Test trace**: a configuration option, used only with synthetic and public prompts, that adds each turn's rendered prompt token IDs, full-vocabulary pre-softmax logits at specified decoding positions, and reused/computed token counts to a restricted diagnostic record. Ordinary records exclude these diagnostic contents.
- **Native repeatability check**: before any accuracy test, the native reference is run twice on every test prompt. If corresponding diagnostic logits after identical preceding tokens differ by more than `SCORE_ATOL` + `SCORE_RTOL` × —value— or the greedy tokens differ, the accuracy tests are marked blocked until the cause (for example a non-deterministic kernel) is found, because the reference itself cannot be trusted.
- **Hardware**: one allocated department GPU. Run a two-prompt smoke test to measure memory before a full job. Local checks use only small synthetic inputs; model downloads and generation stay on the cluster.

Frozen inputs and split. The existing paired-KV experiment has 160 fitting prompts and 40 development prompts. Those 40 have informed checkpoint and design choices and are not final evaluation data. Record the exact revision and manifest hash from https://huggingface.co/datasets/tandemkv/paired-kv-qwen-smollm. For the final 50 prompts, group HotpotQA records connected by a shared context document, exclude groups overlapping any of the existing 200 prompts, sort the remaining IDs, then shuffle with seed 20261102. Freeze the first 50 eligible IDs before testing. Insufficient eligible data blocks the run. Reusing a final set to select another checkpoint makes it additional development data and requires a fresh final set.

Store the exact rendered prompt IDs, suffix boundary, attention masks, positions, model/tokenizer/checkpoint revisions and generation settings in the manifest. Construct length variants by trimming context only, keeping the question and chat delimiters intact;

report the achieved token lengths. The same prompt may tokenize differently in the two models. Every variant of a source document stays in its original split.

For F1, use the deterministic construction in `src/experiments/local_cache_poc.py`, function `prompt`, with seed 20261027, each declared length, and each of early/middle/late. Pin the source commit and tokenizer revision and save the resulting IDs. That function uses "Project Cedar's release code" and fixed archive-note filler; its native output is the oracle, not guaranteed recall of the planted code. For multi-turn cases, send the question again on each following turn and retain the actual previous assistant reply.

For T-IN-2, the ten user turns are the following JSON strings, decoded as UTF-8. Start with system text "Repeat the user's text" and retain actual assistant replies. Append the pinned model's chat delimiter string as plain text to the last turn:

```
["alpha", "caf\u00e9", "cafe\u0301", "\u4e2d\u6587",
 "\u0645\u0631\u062d\u0628\u0627", "\ud83d\ude00", "a\tb",
 "a  b", "line one\n\nline two", "literal delimiter: "]
```

Evidence and retention. Record pass, fail, blocked or inconclusive, with expected/observed outcomes, command, manifest and per-request records. Store large caches/checkpoints on Hugging Face and verify uploaded hashes before deleting local copies. Disable real-user tracing. Retain public/synthetic diagnostics through review, then delete temporary traces within 30 days after final grading unless course retention rules require otherwise. Normal exports exclude raw prompts, generated content, logits, KV and credentials.

4.1 Tests for Functional Requirements

The areas below follow the order of a request: checking the input, checking and loading the predicted cache, falling back on failure, finishing the request, recording what happened, and keeping users apart. Together they cover FR-01 to FR-16 and the safety requirements SR-01 to SR-06 and SR-09 to SR-14.

4.1.1 Request Input

These tests cover the checks made before any work starts (FR-01, FR-02, FR-03; SRS Section 9) and use the request fields and limits of the SRS Data Dictionary (**?**, Section 7.2).

T-IN-1: Invalid requests are rejected before allocation

Control	Automatic
Initial state	Setup S loaded; no requests running.
Input	Eight request cases: (1) no `request_id`; (2) `model_id` = `"llama-x"`; (3) output cap of 0 tokens; (4) a `request_id` already used in the run; (5) a prompt of `MAX_PROMPT_TOKENS` + 1 tokens; (6) a valid request with no cache; (7) a prompt of exactly `MAX_PROMPT_TOKENS` tokens; (8) `CONCURRENT_REQUESTS` + 1 simultaneous valid requests.

Output Requests 1–5 are rejected with the reason codes `missing_field`, `unknown_model`, `invalid_setting`, `duplicate_id` and `too_long`; no request-owned allocation is created for a rejected request; no tokens are generated. Requests 6 and 7 complete. In 8, the extra request is queued or rejected with a reason, as documented.

Derivation FR-01 requires rejection with a reason before resources are allocated, and valid cacheless requests to pass; NFR-09 requires the published limits to hold at their boundaries.

How performed An end-to-end test sends the requests to the runner and checks that rejected requests allocate no request-owned cache; process memory is recorded as supporting evidence, not an exact allocation oracle.

T-IN-2: The router's model and the full prompt are used

Control Automatic

Initial state Setup S; the router replaced by a stub that returns a fixed sequence of models.

Input Ten synthetic turns defined below, with the stub returning `smollm`, `qwen`, `qwen`, `smollm`, and so on. Its turns include accented Latin letters, Chinese, Arabic, emoji, tabs, repeated spaces and the models' own chat delimiter strings as plain text.

Output For each turn, the request record names the model the stub returned, at the pinned revision, and the test trace's prompt token IDs equal the native tokenization of the full history in that model's chat format.

Derivation FR-02 requires the router-selected model and the complete ordered prompt to be preserved; NFR-03 requires text to be preserved exactly.

How performed End-to-end test with the test trace enabled, comparing the recorded token IDs with `apply_chat_template` applied to the full history. The router is an external input (SRS assumption A2), so replacing it with a stub does not bypass Tandem.

T-IN-3: Only an exact history prefix is reused

Control Automatic

Initial state Setup S with a predicted cache for conversation H.

Input Six follow-up requests: H unchanged; H with one character of one message changed; H with two messages swapped; H with a different system instruction; H formatted with a different tokenizer revision; H followed by a new turn.

Output Reuse for "unchanged" and "new turn" only; the other four use normal prefill with reason `prefix_mismatch`.

Derivation FR-03: semantic similarity or equal length is not enough; only an exact prefix in the target model's format may be reused.

How performed End-to-end test reading the reuse decision and reason from the request record.

4.1.2 Cache Checks and Loading

These tests cover the checks before a cache is used (FR-04, SR-02, SR-11) and the loading of an accepted cache (FR-05). The faults come from the cache failure modes in the HA FMEA (**?**, Section 6).

T-CACHE-1: Each cache check rejects its fault

Control Automatic

Initial state Setup S with one valid predicted cache C for a test-set prompt, saved to the cache store.

Input A follow-up request that uses C, and eight more, each after one fault is made to the stored file of C: one layer removed; a wrong number of heads; one value set to NaN; translator version changed; target revision changed; a different sharing group; one byte changed (checksum mismatch, SR-11); covered length one token longer than the history.

Output C is reused. For each faulty file, the request record shows the cache rejected with its own reason code, the request completes with normal prefill, and its greedy output equals the native reference.

Derivation FR-04 requires each check to fail independently; SR-02 requires rejected caches never to enter generation; SR-11 requires a changed byte to be detected.

How performed End-to-end test, parametrized over the faults, editing the stored file between requests.

T-CACHE-2: An accepted cache is loaded and only the suffix is processed

Control Automatic

Initial state Setup S with the test trace enabled.

Input 20 test-set prompts at 512 tokens, each sent as a history turn followed by its question as the next turn: (a) both turns to Qwen (same-model reuse); (b) the history to SmolLM and the question to Qwen (predicted cache).

Output In both cases the record of the question turn shows the history as reused and the computed target tokens equal to the question's token count (plus `CANARY_TOKENS` if the SR-10 check is enabled), and generation completes.

In (a), greedy output equals the native reference for `GREEDY_TOKENS` tokens.

Derivation FR-16 requires prediction without target prefill over the history; FR-05 requires the suffix to be processed once. Same-model reuse is an exact oracle: loading the cache must not change the output, so any difference is a loading bug, not a translation error.

How performed End-to-end test reading the request records.

T-CACHE-3: No token is dropped or repeated across turns

Control Automatic

Initial state Setup S with the test trace enabled.

Input A 5-turn conversation on Qwen in which each turn reuses the cache left by the previous turn's answer.

Output For each turn, the record's reused plus computed token counts equal the token count of the full history, including the last generated token of the previous answer, and greedy output equals the native reference for that turn.

Derivation The SRS reuse rules say no prompt token may be skipped or processed twice. Hugging Face `generate` returns a cache that does not include its final generated token, so this case must be handled explicitly.

How performed End-to-end test reading the records at every turn; run on the cluster.

4.1.3 Fallback and Failure Handling

These tests cover native processing (FR-06), recovery from failures (FR-07, NFR-06) and the run-time check (SR-10, HA Section 7).

T-FB-1: Normal processing is used when reuse does not apply

Control Automatic

Initial state Setup S.

Input Five cases on test-set prompts: a turn on the same model; a first turn with no history; a missing source cache; a switch from Qwen to SmolLM (no translator for that direction); prediction switched off.

Output Each uses normal prefill, records the matching reason code, and gives greedy output equal to the native reference for `GREEDY_TOKENS` tokens.

Derivation FR-06 requires native processing to stay available and unchanged when reuse is unavailable or unsuitable.

How performed End-to-end test.

T-FB-2: Failures restart once before output, never after

Control	Automatic
Initial state	Setup S with the fault-injection setting enabled in the test configuration.
Input	Three requests, with the fault-injection setting making (1) translation fail, (2) cache loading fail, and (3) generation fail after the first output token.
Output	Cases 1 and 2: exactly one native restart, the same `request_id`, output equal to the native reference, one final status `completed`. Case 3: no restart, final status `failed`, and no output from a second attempt appended. In every case, the next 10 valid requests complete with output equal to their references.
Derivation	FR-07 allows at most one restart, before any output, and forbids combining output from two attempts. NFR-06 requires the server to keep serving after a failed request.
How performed	End-to-end test reading output events, final statuses and records.

T-FB-3: The run-time check catches a degraded cache

Control	Automatic
Initial state	Setup S with the SR-10 check enabled.
Input	For 20 test-set prompts, a follow-up request using the stored predicted cache, and a copy with Gaussian noise (seed 20261102) scaled so its perturbation norm equals the original norm at the last `CANARY_TOKENS` positions. Skip a zero-norm case with a recorded reason. Recompute integrity metadata after this synthetic corruption, so a checksum failure does not mask the canary behaviour.
Output	Every noisy cache falls back with reason `check_failed`. The number of clean predicted caches that pass is reported; it must be greater than zero for the check to be useful.
Derivation	SR-10 fit criterion. Peer review issue #41 questions whether SR-10 is consistent with FR-16. This test remains blocked until the SRS permits the bounded recomputation. On development data, measure detection among caches that independently fail the approved quality limit and false fallback among acceptable caches. Agree on numeric detection and false-fallback limits before final testing; a canary passing its own check is not evidence that output quality passes.
How performed	End-to-end test on the cluster.

4.1.4 Request Lifecycle

These tests cover output delivery (FR-08), memory release (FR-09), cancellation (FR-15) and memory held by caches (SR-03 in SRS Section 15.2; SR-14 in HA Section 7). They require the serving integration and run from Rev 0.

T-LIFE-1: Requests end cleanly

Control Automatic

Initial state Setup S in the serving integration, with the timeout set to `TEST_TIMEOUT`; one long request already active and its cache ownership and process-memory baseline M_0 recorded.

Input `LEAK_CYCLES` cycles, each with `CONCURRENT_REQUESTS` overlapping requests that ask the model to repeat a distinct 6-digit code. In each cycle one request is cancelled while queued, one is cancelled after 5 tokens, one is paused by the test configuration until the timeout, and one fails through the fault-injection setting. Use a barrier after cache acquisition to overlap them with the long request; do not rely on model verbosity to force a timeout.

Output Every output event carries its own request ID, in order, and each completed output matches its own native reference; correct recall of the planted code is not assumed. Each accepted request has exactly one final status: `cancelled`, `cancelled`, `failed` with reason `timeout`, `failed`, and `completed` for the rest. After every cycle, ended requests own no private cache allocations; active state remains valid and policy-retained caches are accounted for separately. Plot process memory against M_0, but allocator-reserved memory alone does not establish a leak.

Derivation FR-08, FR-09 and FR-15. The short timeout replaces the 120-second default so the test runs quickly.

How performed Script on the cluster, plotting memory against cycle number.

T-LIFE-2: Cache memory is protected and bounded

Control Automatic

Initial state Setup S in the serving integration; a request generating with predicted cache C.

Input (1) While the request is generating, other requests that fill cache memory so that C would be evicted, then a request whose history would replace C. (2) Predicted caches created for 200 synthetic F1 variants, each with a distinct history note numbered 1–200, and never used.

Output (1) The request's output is identical to an undisturbed run, and the serving engine's cache memory metric shows C freed only after the request ends. (2) Memory held by unused predicted caches never

exceeds `PREDICTED_CACHE_LIMIT`, and no unused cache is older than `PREDICTED_CACHE_TTL`.

Derivation SR-03 and the SR-14 fit criterion.

How performed End-to-end test; the cache memory metric is sampled every second for (2).

4.1.5 Configuration and Records

These tests cover settings (FR-13), approval of a setup (FR-14) and the records used for comparisons (FR-10, FR-11, FR-12).

T-CONF-1: Settings are checked and reuse needs evidence

Control Automatic

Initial state Tandem stopped.

Input (1) A translator for a model pair that is not approved; (2) prediction enabled with no translator checkpoint; (3) a valid configuration with no T-QUAL-2 record, a failing record, a passing record for another translator version, and a passing record for this one; (4) a stored cache reference used after the valid configuration is changed to a different target revision, and after Tandem is restarted.

Output (1) and (2) are rejected at start-up with a reason. In (3), only the last record, together with passing isolation and recovery evidence for this setup, enables reuse. Turning prediction off makes all requests use normal prefill. In (4), both uses are rejected with reason `stale_cache`.

Derivation FR-13; FR-14 with SR-09: missing or failed evidence prevents approval; SR-05: caches from an old configuration are never reused.

How performed End-to-end test.

T-REPORT-1: Requests are recorded and compared fairly

Control Automatic

Initial state Setup S and the comparison tool.

Input (1) One reused and one fallback request with known prompts. (2) A native run and a Tandem run of a 20-request workload, then a second pair in which one generation setting differs.

Output (1) Each record contains the run ID, request ID, source and target models, translator version, timings, computed prompt tokens, final status, reuse decision and reason. Computed prompt tokens equal the suffix length plus any explicitly approved canary recomputation for reuse, and the full prompt length for fallback. Report attempted prediction and restart work as additional cost. A timing that was not measured has a

missing-value reason, not a number. (2) The first comparison lists the same requests, routing decisions, arrival times and settings in both arms and contains every field listed in FR-12. The second is marked invalid, naming the differing setting.

Derivation FR-10, FR-11 and FR-12.

How performed End-to-end test reading the records and the tool's output.

4.1.6 Isolation and Privacy

These tests cover sharing between users (SR-01, SR-04, SR-12), stale caches (SR-05) and logs (SR-06), from SRS Section 15.3 and HA Section 7.

T-ISOLATION-1: No reuse across sharing groups

Control Automatic

Initial state Setup S.

Input The same history sent in sharing group A and then in group B; then again in group A; then a request in group B that claims group A in a request field.

Output Group B never reuses A's cache (reason `no_permission`); the second request in A reuses it; the claimed group is ignored.

Derivation SR-01 and SR-12: sharing is decided by the host, never by the request.

How performed End-to-end test.

T-ISOLATION-2: Another request's private text has no effect

Control Automatic

Initial state Setup S.

Input Fixture F1 as the shared history, branched into two concurrent requests in the same group: request R, which is held fixed, and a second branch B whose private suffix is "secret-A" in one run and "secret-B" in a second run.

Output R's returned log-probabilities are within `SCORE_ATOL` + `SCORE_RTOL` × —value— across the two runs, and its greedy tokens are identical, for `GREEDY_TOKENS` tokens. R's output never contains "secret-A" or "secret-B".

Derivation SR-04: shared history grants no access to later private text. Changing only B and holding R fixed means any difference in R can only come from B leaking into it.

How performed End-to-end test, run only when the native repeatability check passes.

T-ISOLATION-3: Logs contain no private content

Control	Automatic
Initial state	Setup S with default logging.
Input	20 requests whose prompts and system instructions contain the marker `SECRET_MARKER`.
Output	The marker appears 0 times in all logs, records and exports, and no exported file contains tensor data. Request IDs and reasons are present.
Derivation	SR-06.
How performed	End-to-end test searching every file written during the run.

4.2 Tests for Nonfunctional Requirements

Accuracy tests (SRS Section 12.3, HA SR-09) compare against the native reference, and their analysis is in the ML Model Report. Performance tests (SRS Sections 12.1 and 12.5) are experiments: their output is a table and a plot, and pass or fail is decided from confidence intervals, following the SRS timing method (**?**, Section 12.1) (`WARMUP_RUNS` warm-up runs, `TIMED_PAIRS` alternating timed pairs, `BOOTSTRAP_RESAMPLES` bootstrap resamples at `CONFIDENCE_LEVEL`).

4.2.1 Accuracy

T-QUAL-1: Same-model reuse gives identical output

Type	Dynamic, automatic
Initial state	Qwen2.5-0.5B-Instruct as both source and target.
Input/Condition	Fixture F1 and the test-set prompts at each length in `CONTEXT_LENGTHS`, sent as a history turn and a question turn, with the cache (a) kept in memory and (b) saved and restored from file between the turns.
Output/Result	Every corresponding finite diagnostic logit within `SCORE_ATOL` + `SCORE_RTOL` $\times$ —native value— of the native reference, and identical greedy tokens for `GREEDY_TOKENS` tokens or an earlier shared stop. Unexpected nonfinite values fail. Report absolute error where the native score is zero. Run this control separately for each base model; existing Qwen results do not qualify SmolLM.
How performed	Extension of the existing replay experiment, which already passed on 9 cases with a largest logit error of 5.44×10^{-5} (`LOCAL_CACHE_RESULTS.md`). Traces to NFR-05.

T-QUAL-2: Cross-model translation accuracy

Type Dynamic, automatic

Initial state Setup S; translator checkpoint frozen before the test set is first used.

Input/Condition Every test-set prompt at each declared context length, the passage sent to SmolLM and the question to Qwen, with (a) prediction on, (b) prediction off (native reference), (c) the question sent alone with no history, and (d) prediction on with the affine baseline translator configured.

Output/Result For (a), (c) and (d) against (b): first-token agreement, mean matching prefix length, exact-sequence agreement and mean next-token KL divergence. Compute $D_{KL}(p_{native} \| p_{translated})$ at temperature one after identical native preceding tokens, averaging within each prompt and then across prompts. Independent greedy generation is evaluated separately; teacher-path agreement is not sequence agreement. **Current SRS pass:** (a) meets the diagnostic-logit tolerances and identical greedy-token criterion of T-QUAL-1. A failed approximation is reported as failed, even if a diagnostic metric improves.

How performed Evaluation script on the cluster. Freeze the checkpoint and thresholds before final data is inspected. Report every length and bootstrap complete prompts, not correlated token positions. The HA's alternative answer-F1 loss limit is a proposal pending issue 39; it cannot replace NFR-05 until reviewed and adopted in the SRS. If adopted, use labelled, untruncated QA inputs with sufficient answer budget and a separately approved scoring protocol. Answer labels are unnecessary for the native-versus-loaded cache check. Traces to FR-14, FR-16, NFR-05 and proposed HA SR-09.

4.2.2 Performance

T-PERF-1: TTFT against context length

Type Dynamic, automatic, experiment

Initial state Setup S on one cluster GPU; no other jobs on that GPU.

Input/Condition Test-set prompts at each length in `CONTEXT_LENGTHS`, with the question placed at the start, middle and end, in four arms: native prefill; same-model warm cache; same-model cache restored from file; translated cache, timed from the switch and including translation, checks and loading.

Output/Result Median and 95th-percentile TTFT per arm and length with confidence intervals; computed prompt tokens per arm; a breakdown of the translated arm into translation, check, load and suffix prefill time; a plot of TTFT against context length for all four arms; and, not pass/fail, peak allocated memory per arm, cache size per token and GPU time per

completed request including translation. **Pass (NFR-04)** if, at some length, the translated arm processes fewer prompt tokens and the upper confidence bound of its median TTFT is below the lower bound of the native median, with T-QUAL-2 passing.

How performed Benchmark script; results and command committed and analysed in the Performance Report. Also verifies the SR-10 fit criterion that the check keeps TTFT below native.

T-PERF-2: Serving benchmark

Type Dynamic, automatic, experiment

Initial state Serving integration; Setup S.

Input/Condition (1) A workload of conversations with model switches at three arrival rates, with the same arrival times in both arms. (2) Same-model and no-history workloads, with Tandem installed and with the unmodified engine. (3) At the middle rate of (1), all predicted caches invalidated at once.

Output/Result Throughput and p95 TTFT per rate and arm, plotted as throughput against p95 TTFT. **Pass (NFR-07)** if in (1) throughput is higher and p95 TTFT rises by at most `REGRESSION_LIMIT`, with no extra failures. **Pass (NFR-08)** if in (2) the confidence bounds show median and p95 TTFT increases and throughput loss of at most `REGRESSION_LIMIT`. **Pass (SR-13)** if in (3) admitted requests stay within `REGRESSION_LIMIT` of the same load without Tandem, requests beyond `QUEUE_LIMIT` get the retry status instead of timing out, and reuse turns off once the fallback rate over `FALLBACK_WINDOW` requests exceeds `FALLBACK_RATE_LIMIT`.

How performed Benchmark script; from Rev 0; analysed in the Performance Report.

4.2.3 Setup, Documentation and Reports

T-DOC-1: Independent setup walkthrough

Type Static, manual

Initial state A clean machine or cluster account with the listed prerequisites.

Input/Condition A reviewer who did not write the instructions follows the README to install Tandem and run the basic example. From the example's comparison report alone, they then answer: which requests reused a cache; which fell back and why; the median TTFT of each arm; and whether each test passed or failed. Every reason code produced while running this plan's tests is looked up in the documentation.

Output/Result The reviewer completes every step without help, fills in the record in Appendix 6.2, and answers all four questions correctly; every code has

an entry saying what it means and what the operator should do. Any step that needed help is a failure and becomes an issue.

How performed Manual walkthrough before Rev 0; NFR-01, NFR-02, NFR-12, DR-01, DR-02.

T-DOC-2: The test setup can be recreated

Type Dynamic, manual

Initial state The recorded hardware, software versions and commands of a tested release.

Input/Condition A member rebuilds the setup on a different cluster node and reruns T-QUAL-1 and the 512-token case of T-PERF-1.

Output/Result T-QUAL-1 passes; TTFT medians are reported with any hardware or load differences.

How performed Manual; NFR-10.

4.3 Traceability Between Test Cases and Requirements

Table 4 maps each requirement to its tests. Some requirements are also checked by a review or report described in Section 3, noted in the last column.

Table 4: Requirements and the tests that verify them

Requirement	Tests	Also
FR-01	T-IN-1	
FR-02	T-IN-2	
FR-03	T-IN-3, T-IN-2	
FR-04	T-CACHE-1	Code walkthrough
FR-05	T-CACHE-2, T-CACHE-3	Code walkthrough
FR-06	T-FB-1, T-CACHE-1	
FR-07	T-FB-2	Seeded fault
FR-08	T-LIFE-1	
FR-09	T-LIFE-1	
FR-10	T-REPORT-1	
FR-11	T-REPORT-1	
FR-12	T-REPORT-1, T-PERF-1	
FR-13	T-CONF-1	
FR-14	T-CONF-1, T-QUAL-2	ML Model Report (data splits)

Requirement	Tests	Also
FR-15	T-LIFE-1	
FR-16	T-CACHE-2, T-QUAL-2	ML Model Report (per-layer error, metamorphic checks)
NFR-01	T-DOC-1	
NFR-02	T-DOC-1	
NFR-03	T-IN-2	
NFR-04	T-PERF-1	Performance Report
NFR-05	T-QUAL-1, T-QUAL-2	Advisor review
NFR-06	T-FB-2	
NFR-07	T-PERF-2	Performance Report
NFR-08	T-PERF-2	Performance Report
NFR-09	T-IN-1	
NFR-10	T-DOC-2	
NFR-12	T-DOC-1	
SR-01	T-ISOLATION-1	
SR-02	T-CACHE-1	Seeded fault
SR-03	T-LIFE-2	
SR-04	T-ISOLATION-2	
SR-05	T-CONF-1	
SR-06	T-ISOLATION-3	
SR-09 (HA)	T-QUAL-2, T-CONF-1	
SR-10 (HA)	T-FB-3, T-PERF-1	
SR-11 (HA)	T-CACHE-1	
SR-12 (HA)	T-ISOLATION-1	Documentation review
SR-13 (HA)	T-PERF-2	
SR-14 (HA)	T-LIFE-2	
DR-01	T-DOC-1	Checklist review
DR-02	T-DOC-1	

5 Unit Test Description

The unit tests will be written after the MIS (**?**) is complete, so this section gives the planned approach only.

5.1 Unit Testing Scope

The planned modules are the router, the inference runner, the model wrapper, the translator, the cache checks, the request records and the comparison tool. The cache checks and the fallback logic in the runner have the highest priority, because they implement objective 1. The NVIDIA classifier and the Hugging Face and vLLM libraries are external and are not unit tested.

5.2 Tests for Functional Requirements

Each access program in the MIS will get one test of normal behaviour and one test per edge case (black box, from the MIS), plus tests for each branch of the cache checks and fallback logic (white box). Tests that need a model use a tiny randomly initialized model with the same cache layout, so they run in CI without a GPU or downloads. Existing tests (`test/test_routing.py`, `test/test_classifier.py`) already follow this approach for the router. Test names will state the behaviour they check, so the test code documents the inputs and outputs.

5.3 Tests for Nonfunctional Requirements

Non-functional requirements are tested at system level in Section 4. Unit-level timing tests are not planned, because timings of single functions on a CI runner do not predict serving performance.

5.4 Traceability Between Test Cases and Modules

To be completed with the MIS.

6 Appendix

6.1 Symbolic Parameters

Values from the SRS are copied from it. Values for the Hazard Analysis constants and this plan's own constants are initial values for the PoC; they will be reviewed on development data before final tests, and are not changed after a final test fails (SRS, OI3).

Table 5: Symbolic parameters

Constant	Value	Meaning and source
SCORE_ATOL	0.001	Absolute logit tolerance (SRS)
SCORE_RTOL	0.001	Relative logit tolerance (SRS)
GREEDY_TOKENS	32	Generated tokens compared (SRS)
REGRESSION_LIMIT	0.05	Allowed fractional slowdown or throughput loss (SRS NFR-07, NFR-08)
CONFIDENCE_LEVEL	0.95	Confidence level for intervals (SRS)
BOOTSTRAP_RESAMPLES	2,000	Bootstrap resamples (SRS)
WARMUP_RUNS	3	Untimed warm-up runs (SRS)
TIMED_PAIRS	10	Alternating timed pairs (SRS)
LEAK_CYCLES	100	Completion/failure/cancellation cycles (SRS FR-09)
QUALITY_LOSS_LIMIT	0.05	Largest allowed drop in HotpotQA answer F1 (absolute) against native prefill (HA SR-09)
CANARY_TOKENS	16	Final history tokens recomputed by the run-time check (HA SR-10)
CANARY_LIMIT	0.5	Largest relative K/V error at those tokens (HA SR-10)
FALLBACK_WINDOW	100 requests	Window for the fallback rate (HA SR-13)
FALLBACK_RATE_LIMIT	0.5	Fallback rate that turns reuse off (HA SR-13)
QUEUE_LIMIT	32 requests	Requests that may wait for prefill (HA SR-13)
PREDICTED_CACHE_LIMIT	2 GiB	Memory for unused predicted caches (HA SR-14)
PREDICTED_CACHE_TTL	60 s	Lifetime of an unused predicted cache (HA SR-14)

Constant	Value	Meaning and source
TEST_PROMPTS	50	Held-out test prompts, document-disjoint from training and selection data
CONTEXT_LENGTHS	128, 512, 1,024 tokens	History lengths tested, matching the existing replay experiment
MAX_PROMPT_TOKENS	2,048	Longest supported prompt in the PoC
CONCURRENT_REQUESTS	8	Simultaneous requests in lifecycle tests
TEST_TIMEOUT	2 s	Request timeout used in T-LIFE-1
MEMORY_TOLERANCE	1 MiB	Allowed memory difference after a cycle
SECRET_MARKER	TANDEM-SECRET-7f3a	String searched for in logs

6.2 Setup Walkthrough Record

Tandem has no graphical interface, so no usability survey is planned. For T-DOC-1, the reviewer records:

1. For each README step: completed without help (yes/no), time taken, and any error message.
2. Which prerequisite, if any, was missing from the list.
3. Whether the basic example's output matched the README.
4. Whether the reviewer could tell, from the example output, if a cache was reused or the request fell back.
5. The single most confusing part of the instructions.

7 Appendix: Generative AI Use Disclosure

7.1 AI Tools Used

The preceding draft records Claude Code with Claude Opus 5.5 (Anthropic). This completion used OpenAI Codex (GPT-6 family; exact serving build not exposed), desktop host 26.1027.30824, build 22943, on November 2, 2026. Brightspace MCP supplied the Main, LO_Plan and LO_Reflect rubrics.

7.2 How and Where Used

- **Drafting:** Claude drafted this plan from SRS version 0.2, the Hazard Analysis, the course VnV template, checklist and lecture slides, the notes from the October 28 TA meeting, and the team's existing experiments and pull requests.
- **Consistency checking:** Claude checked that every SRS and HA requirement appears in the traceability table, and identified the SRS changes the tests depend on.
- **Merging an earlier draft:** Justin Kwinecki's draft (PR #61), written with OpenAI Codex, was compared with this plan. Its native repeatability check, zero-norm case, fixed-request isolation test and planted-code fixture were adapted into this version.

7.3 Verification of AI-Generated Content

Codex checked the live course rubrics, current SRS IDs, HA proposals and open issues 38–43; corrected test oracles and split handling; compiled the LaTeX; and inspected the rendered pages. It drafted the learning plans at Zayed's request using his stated interest in pretraining and efficient translation. Other members' learning choices remain proposed. No tests or stakeholder approvals are claimed by this editing work.

The requirement IDs were checked against the SRS and HA on the `main` branch, and the replay results against `LOCAL_CACHE_RESULTS.md`. The team reviews every test, the role assignments and the constant values before submission. The team is responsible for the accuracy, completeness and consistency of this document.

8 Appendix: Reflection

8.1 What Went Well

Separating cache replay, translation quality and serving performance gave the plan a clear order. Each stage now has its own evidence: native-cache controls establish whether the loading path works; prediction tests measure how much behaviour changes; matched timing trials establish whether translation saves time. Reusing the SRS IDs also made missing tests and conflicting requirements visible without adding unrelated experiments.

8.2 Difficulties and Resolutions

Our earlier QA checks mixed two questions: whether a base model answers a question correctly and whether loading a cache preserves that model's behaviour. A wrong native answer can still be replayed correctly. T-QUAL-1 compares native prefill with loaded native prefill on the same prompt, while T-QUAL-2 evaluates cross-model approximation separately. We will define the oracle before the next experiment so that a failed test points to the component being tested.

The other difficulty is that the SRS requires near-identical outputs while the HA proposes bounded quality loss and a runtime canary. Writing a convenient threshold here would hide that disagreement. This plan retains the current SRS check and ties the unresolved alternatives to issues 39 and 41. Resolving them on development data before final evaluation is necessary to make the eventual result credible. We also separated the repeatedly used 40 development prompts from a fresh final set to avoid overstating generalization.

8.3 Skills and Learning Approaches

The following individual plans are proposed for team adoption. Each compares two approaches and selects a concrete starting point; they do not claim that the study or team agreement has already occurred.

Zayed Umer: pretraining and efficient cache translation. The learning goal is to understand how pretrained representations and KV state affect translator design, then build a small translator whose quality and runtime justify its cost. Two approaches are a structured study of transformer/pretraining material followed by a small reproduction, or direct architecture and loss-function sweeps. The planned starting point is study plus reproduction: explaining the frozen base models, alignment and training objective should make later ablations interpretable. Evidence of learning will be one controlled comparison that changes one design choice and reports held-out output agreement, parameter count, memory and latency.

Justin Kwinecki: inference-engine cache integration. The skill is understanding cache positions, ownership and recovery during decoding. Two approaches are tracing an existing engine's cache API with a minimal example, or building a full serving adapter immediately. The planned choice is the minimal trace first because it isolates suffix-position

and lifecycle mistakes. The learning check is a native save/load example that passes T-QUAL-1, followed by a deliberately failed load that demonstrates the single-restart rule.

William Pol: test design and traceability. The skill is converting a requirement into a test with independent expected results. Two approaches are studying requirements-based testing examples and reviewing them against our SRS, or learning only by adding cases after bugs appear. The planned choice is the first approach, followed by a peer walkthrough, because it exposes missing or contradictory criteria before implementation. The learning check is a reviewed requirement-to-test matrix with a runnable input and oracle for each entry.

Musab Mondal: cache isolation and failure testing. The skill is reasoning about shared state, permission boundaries and cleanup. Two approaches are studying lifecycle/threat-model examples and constructing a small fault-injection harness, or relying on broad end-to-end stress runs. The planned choice is the focused harness because synthetic secrets and controlled interleavings make a failure attributable. The learning check is demonstrating rejection of cross-group reuse and safe cleanup while another request still holds the cache.

Manav Patel: GPU benchmarking and statistical analysis. The skill is measuring end-to-end cost without hiding prediction overhead or shared-device noise. Two approaches are studying GPU timing and paired comparisons using a small benchmark, or collecting a large timing run first and analyzing it afterward. The planned choice is the small benchmark first so the clock boundaries, cache policy and resampling unit can be checked cheaply. The learning check is a reproducible native-versus-predicted comparison with raw trials, uncertainty and an explanation of any inconclusive result.