Hazard Analysis
Tandem

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

Table 1: Revision History

Date	Developer(s)	Change
Oct 19, 2026	William Pol	First draft of all sections; see the disclosure appendix

Contents

1 Introduction **1**

2 Scope and Purpose of Hazard Analysis **1**

3 System Boundaries and Components **1**

4 Critical Assumptions **2**

5 Hazards **3**

6 Failure Mode and Effect Analysis **4**

7 Safety and Security Requirements **7**

8 Roadmap **8**

9 Appendix — Generative AI Use Disclosure **9**

9.1 AI Tools Used 9

9.2 How and Where Used 9

9.3 Verification of AI-Generated Content 9

10 Appendix — Reflection **10**

1 Introduction

This document analyzes the hazards of Tandem, which predicts a target model's key-value (KV) cache from a source model's KV cache and loads the prediction into a large language model (LLM) serving engine. When a semantic router sends the next turn of a conversation to a different model, Tandem lets the new model start from the predicted cache instead of processing the whole conversation history again (prefill). The prediction is approximate, so much of this analysis concerns how a wrong prediction is caught before it reaches the user. The Software Requirements Specification (SRS) describes the requirements this analysis refers to.

A *hazard* is a condition of Tandem that, together with a condition in its environment, can lead to a loss. Tandem does not control physical equipment, so, as the course allows, we treat harm broadly: incorrect answers, disclosure of private information, unavailable service, wasted computing resources, and misleading performance claims.

Changes to the template. We added a hazard list (Section 5) before the Failure Mode and Effect Analysis (FMEA) so that the table, the SRS and the roadmap can all refer to the same hazard identifiers. The FMEA table adds a hazard column for that traceability.

2 Scope and Purpose of Hazard Analysis

The purpose of this analysis is to find how Tandem can fail, decide how each failure is detected and mitigated, and turn the mitigations into requirements. Following the as low as reasonably practicable (ALARP) principle, the goal is not zero risk but risk reduced as far as is practical within the capstone.

The losses considered are:

L1. **Incorrect response.** An application receives an answer that differs from what the selected model would have produced, and nothing marks it as unusual.

L2. **Privacy loss.** One user's conversation, instructions or answer is revealed to another user, directly or indirectly.

L3. **Lost service.** A valid request fails, times out, or slows other requests beyond acceptable limits.

L4. **Wasted resources.** Graphics processing unit (GPU) memory or time is spent on work that is never used.

L5. **Misleading results.** Evaluation reports claim a speedup or quality level that the system does not deliver, leading stakeholders to adopt it wrongly.

The analysis covers Tandem on a single server, as the SRS scopes it, including the development and evaluation of the KV cache predictor. It excludes the semantic router's choice of model, the application that stores the conversation, training of the base models, and operation across several servers, because these are outside the product boundary in the SRS. Failures at those boundaries are still considered where they affect Tandem.

3 System Boundaries and Components

The components below come from the context and work partitioning in the SRS. They are used to brainstorm hazards, not as a design.

Inside Tandem.

C1. **Request validation:** checks request fields, sizes and settings before work begins.

C2. **KV cache predictor:** estimates the target model's keys and values from the source model's KV cache, using versioned parameters and recorded token, layer and head alignment (SRS FR-16). Its parameters may be fitted on paired source and target caches.

C3. **Compatibility checker:** decides whether a candidate cache may be used, using the checks in the SRS formal specification (model setup, exact prefix, permission, readiness and resources).

C4. **KV loader:** places an accepted cache into the serving engine's KV memory and prefills the remaining prompt.

C5. **Native prefill fallback:** processes the full prompt normally when reuse is off, unavailable or unsuitable.

C6. **Decoding and output streaming:** returns generated tokens in order under the correct request identifier (ID).

C7. **Request lifecycle:** handles completion, failure, cancellation and timeout, and releases memory.

C8. **Configuration and tested-setup registry:** stores the approved model/cache setups and the reuse on/off control.

C9. **Logging and metrics:** records per-request status, reuse decisions and measurements.

C10. **Benchmark tools:** run matched workloads with and without Tandem and produce comparison reports.

Outside Tandem, but sources of hazards.

C11. **Semantic router:** selects the target model for each turn.

C12. **Application:** supplies the conversation, the new message and the sharing group of each request.

C13. **Serving engine and GPU:** the vLLM engine [**?**], its memory manager, the source and target model files, the source model's KV cache and the hardware.

4 Critical Assumptions

The SRS lists assumptions A1–A6. Only the following affect which hazards are analyzed:

- **A1 (trusted cache provider).** The SRS assumes the initial cache provider runs in the operator's trusted environment and that users cannot upload caches. Because Tandem's own predictor now produces the cache, deliberate tampering is less likely, but a cache can still be corrupted by a software fault, a storage error or a mismatched predictor version. We therefore do *not* rely on A1: H9 remains in the analysis, and stronger mitigations are placed on the roadmap for any future release that accepts caches from outside.
- **A4 (a useful predictor exists).** The SRS states that a predictor which is both accurate enough and cheaper than prefill is a proof-of-concept question. We do not assume it succeeds: H8 and H15 cover a predictor that is inaccurate or too slow, and in both cases the native prefill path must remain correct.
- **A2 (router and application inputs).** The router's model choice and the application's conversation are taken as given. Tandem still validates them (H1, H6), because a wrong input must not cause a wrong cache to be used.
- **A3 (adequate server).** The server can run the selected models normally. Running out of memory is still analyzed (H6, H12, H14).
- **A5 (data for the predictor).** Public or synthetic conversations are available for fitting and testing the predictor. If the same conversations are used for both, results look better than they are; this is analyzed as part of H11.

No assumption states that a component never fails.

5 Hazards

Table 2 lists the hazards.

Table 2: Hazards and the losses they can cause

ID	Hazard	Losses
H1	An incompatible cache (wrong model, tokenizer, format, positions or precision) is used and changes the response.	L1
H2	One request uses or reveals another request's private history, instructions or answer.	L2
H3	An old or incomplete cache is used.	L1
H4	Cache data is changed or removed while a request still uses it.	L1, L3
H5	Logs or measurement files reveal private data.	L2
H6	Memory or other resources run out.	L3
H7	Output from a failed reuse attempt is mixed with a restarted answer.	L1
H8	A compatible, tested predicted cache still lowers answer quality, because cross-model prediction is approximate.	L1, L5
H9	A cache is corrupted or deliberately altered before it is loaded.	L1, L2
H10	The time difference between reuse and normal processing reveals information about other users' conversations.	L2
H11	Evaluation reports overstate the benefit or hide failures.	L5
H12	Predicted caches that are never used consume memory and GPU time.	L4, L3
H13	The model, tokenizer, alignment rules or predictor version change between predicting a cache and using it.	L1
H14	Many reuse attempts fail at once and the fallback overloads the server.	L3
H15	Predicting, checking and loading the cache takes longer than native prefill, so reuse slows the request instead of speeding it up.	L3, L4

Three hazards need context.

H8, approximate prediction. Unlike same-model prefix caching, predicting one model's KV cache from another's does not reproduce the target model's own cache exactly. Published results show that the quality loss can be large and depends on the direction of transfer. For example, one cross-family method reports a question-answering F1 score falling from 45.6 to 20.6 when transferring from a 4-billion-parameter model to a 14-billion-parameter model, while some transfers in the opposite direction stay close to the target's own results [**?**]. A cache with the right shape and a passing compatibility check is therefore not necessarily good enough, and a predictor that scores well on the conversations it was fitted on may not do so on new ones.

H9, altered cache. Retained KV state can change a model's behaviour even when the text that produced it is no longer part of the request [**?**]. A cache that has been corrupted or altered could therefore steer the response without any visible change to the prompt.

H10, timing side channel. Shared prefix caches in LLM serving systems have been shown to leak information through timing: a cache hit returns its first token sooner, which lets an observer infer what other users have sent [**?**]. Tandem adds another fast path, so the same risk applies.

6 Failure Mode and Effect Analysis

Table 3 analyzes how each component can fail. Each row gives the recommended action and the SRS requirement that implements it.

Table 3: Failure Mode and Effect Analysis

ID	Component	Failure mode	Effects	Causes	Detection	Recommended action	Req.	Hazard
F1	Request validation	Accepts a malformed or oversized request	Out-of-memory error or wrong output; other requests slowed	Missing size or setting checks	Rejection tests with invalid inputs (T-ADM)	Reject before allocating resources; check sizes against the configuration	FR-01, SR-07	H6
F2	Compatibility checker	Accepts a cache predicted for a different model, tokenizer, format or position layout	Garbled or subtly wrong answer	Missing metadata treated as a pass; equal cache sizes treated as a match	Each check forced to fail separately (T-COMP, T-MATCH)	Treat missing information as a failed check; require every check to pass	FR-03, FR-04, SR-02	H1
F3	Compatibility checker	Accepts a cache from before a model or configuration change	Wrong answer	Version not recorded; restart without invalidation	Change the model or configuration, then reuse an old cache (T-CONFIG)	Record versions with each cache; invalidate on restart or change	SR-05, FR-13	H3, H13
F4	KV cache predictor	A tested, compatible predicted cache lowers answer quality	Plausible but wrong answers, worst when predicting a larger model from a smaller one	Approximate prediction; conversation unlike the fitting data	Held-out prediction error and output tests (T-PRED, T-QUAL); run-time check on the last history tokens	Approve each setup only within a quality-loss limit; at run time, recompute the last few history tokens on top of the predicted cache and fall back when they disagree	SR-09, SR-10, FR-14, FR-16	H8
F5	KV cache predictor	Predicted cache is corrupted before loading	Steered or wrong answer	Storage error; software fault; untrusted source in a future release	Integrity check fails	Record a checksum and the predictor version with each cache; reject on mismatch	SR-11, SR-02	H9
F6	KV cache predictor	Tokens, layers or heads aligned wrongly between source and target	Keys and values placed at the wrong positions; garbled answer	Tokenizers split text differently; alignment rules not updated for a new model pair	Alignment checks in T-PRED; run-time check on the last history tokens	Record alignment with each cache; reject a cache whose alignment does not match the configured rules	FR-16, SR-02, SR-10	H1, H8
F7	KV cache predictor	Produces invalid tensors (wrong shape, non-finite values)	Request crashes or produces nonsense	Numerical overflow; input outside the fitted range	Forced invalid outputs (T-FAIL)	Validate shape and values before loading; fall back to native prefill	FR-07, FR-16	H1, H6
F8	KV cache predictor	Prediction and loading take longer than native prefill	Slower first token than without Tandem	Short shared history; large predictor; slow loading path	TTFT benchmarks against native prefill (T-PERF)	Use prediction only where tests show a gain; otherwise use native prefill	NFR-04, NFR-08	H15
F9	Compatibility checker	Reuses a cache from another request's sharing group	Private history exposed	Sharing group not checked; request names its own group	Cross-group isolation tests (T-ISOLATION)	Sharing group assigned by the host; permission check required	SR-01, SR-04	H2
F10	KV loader	Reuse is measurably faster than normal processing	An observer infers other users' conversations	Caches reused across users	Review of sharing settings; timing tests	Reuse only within one sharing group; disable cross-user sharing by default; document the remaining risk	SR-12	H10
F11	KV loader	Cache evicted or overwritten while in use	Corrupted output for an active request	Reference-counting fault; eviction policy	Repeated lifecycle tests (T-LIFE)	Pin caches while any request uses them	SR-03, FR-09	H4
F12	KV loader	Loading fails after output has started	Partial answer	GPU error; memory shortage	Forced failure tests (T-FAIL)	End with a failure status; never append a restarted answer	FR-07, SR-08	H7

ID	Component	Failure mode	Effects	Causes	Detection	Recommended action	Req.	Hazard
F13	Native prefill fallback	Fallback is not triggered or fails	A valid request fails	Logic fault; fallback disabled	Missing-cache and unsupported-pair tests (T-NATIVE)	Fall back on any failed check; keep the normal path always available	FR-06, NFR-06	H1, H6
F14	Native prefill fallback	Many requests fall back at the same time	Delay spike and timeouts for all requests	A configuration change or predictor fault invalidates many caches at once	Fallback-rate and queue metrics	Admit prefills only within the engine's budget; queue a bounded number and reject the rest with a retry status; turn reuse off when the fallback rate exceeds a limit	SR-13, NFR-08	H14
F15	KV cache predictor	Predicted caches are never used	Memory pressure, evictions and wasted GPU time	Cache predicted for a model the router did not select	Memory and prediction metrics	Cap the memory for predicted caches and expire unused ones	SR-14	H12, H6
F16	Request lifecycle	Cancelled or timed-out request keeps its memory	Memory leak leading to out-of-memory errors	Missing release path	Repeated cancellation tests (T-LIFE, T-CANCEL)	Release resources on every final status	FR-09, FR-15	H6
F17	Output streaming	Tokens attached to the wrong request	A user receives another user's answer	Request IDs mixed under concurrency	Overlapping-request tests (T-STATE)	Keep one ordered output stream and one final status per request	FR-08	H2
F18	Logging and metrics	Raw prompts, answers or cache data written to logs	Private data exposed	Debug logging left on	Scan of logs after tests (T-PRIVACY)	Log only IDs and status or reason codes	SR-06	H5
F19	Benchmark tools	Baseline differs from the Tandem run, or failures and preparation cost are left out	Overstated benefit	Different settings; selective reporting	Configuration comparison (T-EVAL, T-REPORT)	Run matched workloads; report failures, runs with no gain, and cache preparation cost	FR-11, FR-12	H11
F20	Configuration registry	An untested setup is enabled	Wrong or low-quality answers	Operator error	Attempt to enable without evidence (T-QUAL)	Keep reuse off for setups without passing test records	FR-13, FR-14	H1, H8
F21	Benchmark tools	Predictor evaluated on conversations it was fitted on	Reported accuracy higher than in real use	Data split done after fitting, or overlapping conversations	Check of the recorded data split (T-QUAL, T-REPORT)	Split conversations into fitting and held-out groups before fitting; report the split	FR-14, FR-12	H11, H8

7 Safety and Security Requirements

The existing SRS requirements SR-01 to SR-08 mitigate H1–H7 and are referenced in Table 3. The analysis found the following new requirements, which should be added to the SRS security and robustness sections. Their fit criteria use the constants in Table 4.

SR-09 — Limit the quality loss of each cross-model setup (H8)
Tandem shall enable a cross-model setup only if, on its declared test set, which shares no conversations with the data used to fit the predictor, the selected model's task scores with the predicted cache are no more than `QUALITY_LOSS_LIMIT` below its scores with native prefill. *Fit criterion:* T-QUAL compares both runs on the same test set; a setup above the limit stays disabled.

SR-10 — Check predicted caches at run time (H8)
For each reuse, Tandem shall recompute the last `CANARY_TOKENS` tokens of the shared history with the target model, using the predicted cache for all earlier positions, and compare the recomputed keys and values with the predicted ones at those positions. If the difference exceeds `CANARY_LIMIT`, it shall use native prefill. The check costs about `CANARY_TOKENS` tokens of prefill, not the whole history. It detects badly wrong predictions but does not prove a cache correct; SR-09 provides the comparison with native prefill. *Fit criterion:* a test injects a degraded cache and checks that the request falls back and records the reason; T-PERF shows that the check keeps time to first token below native prefill for supported setups.

SR-11 — Check cache integrity (H9)
Each predicted cache shall carry a checksum and a record of its source model and predictor version; Tandem shall reject a cache whose checksum or record does not match. *Fit criterion:* a test alters one byte of a cache and checks that it is rejected.

SR-12 — Limit timing leaks (H10)
Tandem shall reuse a cache only within the sharing group it was predicted for, with reuse across users disabled by default; the documentation shall state the remaining timing risk. *Fit criterion:* T-ISOLATION shows no reuse across groups; a documentation review finds the risk statement.

SR-13 — Limit fallback load (H14)
Tandem shall admit fallback prefills only within the engine's configured prefill budget, queue at most `QUEUE_LIMIT` further requests, and reject any beyond that with a status telling the caller to retry, rather than letting them time out. It shall also turn cross-model reuse off automatically when the fallback rate over the last `FALLBACK_WINDOW` requests exceeds `FALLBACK_RATE_LIMIT`, so no prediction work is wasted, and report this to the operator. *Fit criterion:* a test invalidates all caches at once under load; admitted requests stay within `REGRESSION_LIMIT` of the same load without Tandem (NFR-08), excess requests receive the retry status instead of timing out, and reuse is turned off.

SR-14 — Bound memory for predicted caches (H12)
Memory held by predicted but unused caches shall not exceed `PREDICTED_CACHE_LIMIT`, and unused caches shall expire after `PREDICTED_CACHE_TTL`. *Fit criterion:* a test predicts caches that are never used and checks that memory stays within the limit.

Changes needed in the SRS.

- Add SR-09 to SR-14, their constants, and H8–H15 to the traceability table, which resolves open issue OI6. FR-16 should also trace to H8, H13 and H15, not only H1 and H3.
- **NFR-05 conflicts with H8.** NFR-05 requires every reuse setup to match the native scores within `SCORE_ATOL` and to produce identical greedy tokens. The SRS itself states that cache prediction may be approximate, so no cross-model setup could pass, and FR-14 would never enable reuse. NFR-05 should keep the exact check for same-model reuse and use SR-09 and SR-10 for cross-model reuse. [TODO: agree on this change with the SRS author]
- Note in A1 that H9 is still analyzed (Section 4).

Table 4: New constants (values to be set by the team)

Constant	Meaning
QUALITY_LOSS_LIMIT	Largest accepted drop in task score for a cross-model setup. [TODO: choose a value with the team and advisor]
CANARY_TOKENS	Number of final history tokens recomputed for the run-time check. [TODO: choose a value]
CANARY_LIMIT	Largest accepted difference between the recomputed and predicted keys and values for those tokens. [TODO: choose a value]
FALLBACK_WINDOW	Number of recent requests used to compute the fallback rate. [TODO: choose a value]
FALLBACK_RATE_LIMIT	Fallback rate above which reuse is turned off. [TODO: choose a value]
QUEUE_LIMIT	Number of requests that may wait for prefill before further requests are rejected. [TODO: choose a value]
PREDICTED_CACHE_LIMIT	Memory allowed for predicted but unused caches. [TODO: choose a value]
PREDICTED_CACHE_TTL	Time after which an unused predicted cache expires. [TODO: choose a value]

8 Roadmap

Table 5 shows which requirements are planned within the capstone and which are left for later. Requirements for the capstone are the ones needed for a correct, safe single-server demonstration.

Table 5: Roadmap for the safety and security requirements

Requirements	When	Reason
SR-01 to SR-08	Capstone	Needed for any correct reuse; already in the SRS.
SR-09, SR-10	Capstone	Cross-model prediction is approximate (H8), so quality checks are part of the core product.
SR-11 (checksum and source)	Capstone	Cheap to build and detects corruption.
SR-12 (sharing-group limit)	Capstone	Configuration and documentation only.
SR-13, SR-14	Capstone, after the core path works	Needed for load tests, but depend on the reuse path.
Signed caches and caches uploaded by users	Future	A1 excludes untrusted sources in the first release; accepting them would need authentication of the cache source.
Timing side-channel defences, such as padding response times	Future	Costs part of the speedup; SR-12 reduces the risk for now.
Hazards of operation across several servers	Future	Outside the SRS scope.

9 Appendix — Generative AI Use Disclosure

9.1 AI Tools Used

Claude Code, running Claude Opus 5.5 (Anthropic).

9.2 How and Where Used

- **Drafting:** Claude drafted all sections of this document from the team's SRS draft, the course Hazard Analysis lecture, the template and the Hazard Analysis checklist. This includes the hazard list, the FMEA table, SR-09 to SR-14 and the roadmap.
- **Consistency checking:** Claude compared the SRS requirement IDs and draft hazard labels with this document, and identified the conflict between NFR-05 and approximate cross-model prediction.
- **Literature:** Claude found the three papers cited for H8, H9 and H10, and checked their titles, authors and the reported numbers on the arXiv pages and the HeteroFold repository.

9.3 Verification of AI-Generated Content

The citations were checked against the arXiv abstract pages, and the H8 figures against the results table published with the HeteroFold code. The requirement IDs were checked against the SRS draft on the `SRS` branch. The team reviewed every hazard, FMEA row and new requirement before submission. The team is responsible for the accuracy, completeness and consistency of this document.

10 Appendix — Reflection

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   After meeting with our supervisor, the team had a clear idea of the project's direction. Identifying the hazards and losses went well because, by this point, we also understood the technology well from reviewing the state-of-the-art papers on KV cache reuse and cross-model transfer, so we could see where the system could fail and who would be affected.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   The hardest part was coordinating between the writer of the SRS and the writer of this Hazard Analysis, who worked at the same time and had to agree on the functional, non-functional and security requirements. We resolved this by using the SRS identifiers throughout this document and collecting every change the SRS needs, including the new requirements and the NFR-05 conflict, in Section 7 so that the SRS author could review and apply them in one place.

3. Which of your listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? For the latter ones (ones you thought of while doing the Hazard Analysis), how did they come about?

   H1–H7 were identified before this deliverable, while writing the SRS. They came from discussions within the team and with our supervisor, and from research papers on KV cache reuse, which showed when a cache can safely be reused. H8–H15 came up during this analysis. H8 came from reading published cross-model transfer results, which showed large and direction-dependent quality loss. H10 came from work on timing leaks in prefix caching. H11 to H14 came from asking what happens when many requests fail at once, or when predicted caches are never used. H15 came from the SRS assumption that a predictor cheaper than prefill is still an open question.

4. Other than the risk of physical harm (some projects may not have any appreciable risks of this form), list at least 2 other types of risk in software products. Why are they important to consider?

   Privacy risk: software can reveal information users expected to stay private, as in H2, H5 and H10, and such a leak cannot be undone. Risk of misleading results: software that produces confident but wrong output, as in H8 and H11, can lead people to make poor decisions, because nothing tells them the output is unreliable.