Software Requirements Specification for Tandem: Predicting Target-Model KV Caches from Source-Model KV Caches

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

November 3, 2026

Contents

1 Purpose of the Project **vi**
- 1.1 User Business . . . vi
- 1.2 Goals of the Project . . . vi

2 Stakeholders **vii**
- 2.1 Client . . . vii
- 2.2 Customer . . . vii
- 2.3 Other Stakeholders . . . vii
- 2.4 Hands-On Users of the Project . . . vii
- 2.5 Personas . . . viii
- 2.6 Priorities Assigned to Users . . . viii
- 2.7 User Participation . . . ix
- 2.8 Maintenance Users and Service Technicians . . . ix

3 Mandated Constraints **ix**
- 3.1 Solution Constraints . . . ix
- 3.2 Implementation Environment of the Current System . . . x
- 3.3 Partner or Collaborative Applications . . . x
- 3.4 Off-the-Shelf Software . . . x
- 3.5 Anticipated Workplace Environment . . . x
- 3.6 Schedule Constraints . . . x
- 3.7 Budget Constraints . . . xi
- 3.8 Enterprise Constraints . . . xi

4 Naming Conventions and Terminology **xi**
- 4.1 Glossary of All Terms, Including Acronyms, Used by Stakeholders involved in the Project . . . xi

5 Relevant Facts And Assumptions **xiii**
- 5.1 Relevant Facts . . . xiii
- 5.2 Business Rules . . . xiv
- 5.3 Assumptions . . . xiv

6 The Scope of the Work **xvi**
- 6.1 The Current Situation . . . xvi
- 6.2 The Context of the Work . . . xvii
- 6.3 Work Partitioning . . . xvii

6.4 Specifying a Business Use Case (BUC) xix

7 Business Data Model and Data Dictionary **xix**
7.1 Business Data Model xix
7.2 Data Dictionary xx

8 The Scope of the Product **xxi**
8.1 Product Boundary xxi
8.2 Product Use Case Table xxi
8.3 Individual Product Use Cases (PUC's) xxiii

9 Functional Requirements **xxiii**
9.1 Functional Requirements xxiii

10 Look and Feel Requirements **xxix**
10.1 Appearance Requirements xxix
10.2 Style Requirements xxix

11 Usability and Humanity Requirements **xxx**
11.1 Ease of Use Requirements xxx
11.2 Personalization and Internationalization Requirements xxx
11.3 Learning Requirements xxx
11.4 Understandability and Politeness Requirements xxx
11.5 Accessibility Requirements xxxi

12 Performance Requirements **xxxi**
12.1 Speed and Latency Requirements xxxi
12.2 Safety-Critical Requirements xxxi
12.3 Precision or Accuracy Requirements xxxii
12.4 Robustness or Fault-Tolerance Requirements xxxii
12.5 Capacity Requirements xxxiii
12.6 Scalability or Extensibility Requirements xxxiii
12.7 Longevity Requirements xxxiv

13 Operational and Environmental Requirements **xxxiv**
13.1 Expected Physical Environment xxxiv
13.2 Wider Environment Requirements xxxiv
13.3 Requirements for Interfacing with Adjacent Systems xxxiv
13.4 Productization Requirements xxxiv

13.5 Release Requirements xxxv

14 Maintainability and Support Requirements **xxxv**
14.1 Maintenance Requirements xxxv
14.2 Supportability Requirements xxxv
14.3 Adaptability Requirements xxxv

15 Security Requirements **xxxv**
15.1 Access Requirements xxxv
15.2 Integrity Requirements xxxvi
15.3 Privacy Requirements xxxvi
15.4 Audit Requirements xxxvii
15.5 Immunity Requirements xxxvii

16 Cultural Requirements **xxxvii**
16.1 Cultural Requirements xxxvii

17 Compliance Requirements **xxxvii**
17.1 Legal Requirements xxxvii
17.2 Standards Compliance Requirements xxxviii

18 Open Issues **xxxviii**

19 Off-the-Shelf Solutions **xxxix**
19.1 Ready-Made Products xxxix
19.2 Reusable Components xxxix
19.3 Products That Can Be Copied xxxix

20 New Problems **xxxix**
20.1 Effects on the Current Environment xxxix
20.2 Effects on the Installed Systems xl
20.3 Potential User Problems xl
20.4 Limitations in the Anticipated Implementation Environment That May Inhibit the New Product xl
20.5 Follow-Up Problems xl

21 Tasks **xl**
21.1 Project Planning xl
21.2 Planning of the Development Phases xl

22 Migration to the New Product **xli**
22.1 Requirements for Migration to the New Product xli
22.2 Data That Has to be Modified or Translated for the New System xli

23 Costs **xli**

24 User Documentation and Training **xlii**
24.1 User Documentation Requirements xlii
24.2 Training Requirements xlii

25 Waiting Room **xliii**

26 Ideas for Solution **xliii**

27 Appendix — Generative AI Use Disclosure **xliii**
27.1 AI Tools Used xliii
27.2 How and Where Used xliv
27.3 Verification of AI-Generated Content xliv

28 Appendix — Reflection **xlv**

Revision History

Date	Version	Notes
Oct 18, 2026	0.1	Draft requirements for the current Tandem scope.
Oct 19, 2026	0.2	Make source-to-target KV cache prediction central to the scope, requirements, evaluation and workflow diagram.
Oct 19, 2026	0.3	Shorten supporting sections and requirements; mark host-provided security features not applicable and consolidate duplicate checks.
Oct 20, 2026	0.4	Present functional and quality requirements as bullet points; replace most mathematical notation with plain-language checks.
Oct 20, 2026	0.5	Remove the traceability matrix and draft test-settings table; replace remaining setting references with plain-language limits.

1 Purpose of the Project

1.1 User Business

Tandem predicts a target large language model's key-value (KV) cache from a source model's KV cache to avoid repeating prefill for shared conversation history after a model switch. The intended benefits are lower GPU compute, serving cost and time to first token (TTFT).

vLLM Semantic Router selects the target model. Tandem validates and loads the prediction; the engine prefills the uncached suffix and decodes. Unsupported or failed predictions use native processing. The project includes predictor development, any fitting/training, paired KV data and engine integration on one GPU server; base LLM weights stay frozen. Accuracy and end-to-end benefits remain to be demonstrated.

1.2 Goals of the Project

Tandem addresses repeated prefill after model switches:

G1: Predict target KV from source KV. Develop and integrate a predictor that lets supported target models reuse the shared history without repeating prefix prefill.

G2: Reduce compute and serving cost. Lower GPU time per completed request after counting prediction, validation and cache-loading overhead.

G3: Support economical model switching. Preserve the benefit of routing suitable requests to cheaper models by reducing repeated history processing.

G4: Improve latency and throughput. Lower time to first token (TTFT) and increase completed requests per second within the agreed latency limits.

Matched native/Tandem tests measure KV error, output effects, actual prompt work, GPU time, latency and throughput. Cost claims state rates and assumptions; router-selection savings are reported separately. Output accuracy, privacy and native fallback remain required.

2 Stakeholders

2.1 Client

A potential client is a multi-model application company such as Cursor, whose team could help identify useful model combinations, acceptable response delays and cost-saving goals; this is a proposed adoption scenario, not a confirmed partnership.

2.2 Customer

A potential customer is an inference provider such as Together AI or an internal AI infrastructure team that could deploy Tandem to serve more requests with its GPUs; adoption of the planned open-source software does not require a licence purchase.

2.3 Other Stakeholders

Other people who could benefit include:

- **Independent application developers looking to host local LLM:** could provide faster AI features without needing to build their own serving optimization.
- **Inference Data Center Operators:** could make better use of limited GPU capacity and see clear information about cache reuse and failures.
- **Researchers:** could use the test tools and results to study when sharing cached calculations is useful and when it is ineffective.

2.4 Hands-On Users of the Project

Engine developers integrate the predictor; operators configure approved predictors and resource limits; researchers evaluate results; application developers use the existing serving interface. Technical users need Linux, LLM-serving and GPU knowledge. vLLM Semantic Router selects the target model, and application end users need no knowledge of Tandem.

2.5 Personas

These are potential adoption scenarios, not confirmed partnerships or company-supplied requirements; Alex Morgan and Sofia Nguyen are fictional.

Cursor (AI Application Company). Cursor offers multi-model AI development tools. In this scenario, coding conversations retain instructions, code and earlier responses across model switches. The application team wants cheaper model choices without repeated history prefill erasing the savings. Tandem would require a compatible backend with KV access; a hosted API alone is insufficient. The team needs documented model pairs, unchanged prompts and routing, measured benefits and native fallback.

Together AI (Inference Hosting Provider). Together AI offers dedicated inference. This scenario represents operators managing GPU capacity, queues and customer isolation. They would configure approved predictors, model pairs and limits, monitor reuse and failures, and disable prediction when needed. Adoption requires accuracy/privacy checks and gains that include prediction/loading overhead; production-stack compatibility needs separate assessment.

Alex Morgan (Inference-Engine Maintainer). Alex maintains a vLLM deployment using Python and GPU tensor operations. Frequent model and dependency updates can break cache integration. Alex needs versioned interfaces and tests covering prediction, validation, loading, suffix prefill and decoding, plus diagnostics for unsupported caches and reliable native fallback.

Sofia Nguyen (LLM Inference Researcher). Sofia studies efficient LLM serving on shared laboratory GPUs. She compares predicted and native KV state and measures when prediction overhead outweighs saved prefill. She needs held-out conversations, matched routing/settings, repeated trials, output checks, cache/fallback records and reproducible hardware/software details. Measured GPU time and estimated monetary savings must remain distinguishable.

2.6 Priorities Assigned to Users

Output accuracy and privacy take priority, followed by reliable fallback, lower inference cost and latency, higher capacity and reproducible evidence; these priorities apply across the four personas.

2.7 User Participation

Potential application developers, operators, maintainers and researchers can review integration examples, recovery scenarios and benchmarks; record actual participants and feedback separately from the hypothetical personas.

2.8 Maintenance Users and Service Technicians

Alex Morgan represents integration maintenance and Together AI represents operational support; both need versioned setup instructions and diagnostics, while the hosting organization maintains hardware, access and the operating system.

3 Mandated Constraints

3.1 Solution Constraints

The initial project has these limits:

C1: Run text-generation requests on one server.

C2: Use the target model selected by vLLM Semantic Router, or the same recorded routing decision when replaying a benchmark.

C3: Use existing trained source and target LLMs with frozen base-model weights. Developing a KV cache predictor, including fitting or training its parameters if the chosen method requires it, is part of Tandem.

C4: Reuse only an exact matching start of the conversation, using a model pair and KV prediction method that have passed the required tests.

C5: Keep the serving engine's normal prompt processing available whenever reuse is not possible.

These limits keep the project manageable and allow a fair comparison with the serving engine running without Tandem.

3.2 Implementation Environment of the Current System

Use a Linux GPU server with sufficient model memory, with vLLM as the planned engine and SGLang as an alternative; record exact model, tokenizer, engine and GPU-software versions before testing. CPU checks do not establish GPU performance, and the first model pair remains open (OI1–OI2).

3.3 Partner or Collaborative Applications

The application sends conversation messages and generation settings through the existing serving interface; vLLM Semantic Router selects the target. The engine supplies source KV through an internal cache reference, not HTTP tensor uploads. Tandem predicts and validates target KV, loads accepted state and prefills the uncached suffix. The full prompt remains available for checks and fallback. Native target KV references are computed offline for development/evaluation, not as hidden shared-prefix prefill during serving. The application retains conversation storage and access control; benchmarks replay routing decisions. The predictor and first supported pair remain open (OI1).

3.4 Off-the-Shelf Software

Reuse the serving engine, Python, PyTorch, Hugging Face Transformers, tokenizers and GPU libraries; record versions/licences and rerun compatibility tests after relevant updates. Predictor training is in scope; base-LLM training is not.

3.5 Anticipated Workplace Environment

Develop locally and use a shared Linux GPU server for model experiments, recording hardware and competing workloads for repeatable performance tests.

3.6 Schedule Constraints

The Development Plan schedules the SRS/Hazard Analysis for October 19, 2026, proof of concept in November, initial demonstration in February and

final work in March–April; confirm course dates and reconcile scope under OI5 (see Section 21.2).

3.7 Budget Constraints

Use existing university GPU resources; available compute time and storage remain unconfirmed (OI7), so reschedule or reduce tests if resources are insufficient.

3.8 Enterprise Constraints

Review repository changes through issues and pull requests, record release versions, and keep credentials and private conversations out of public files; use only public, synthetic or permitted test data.

4 Naming Conventions and Terminology

4.1 Glossary of All Terms, Including Acronyms, Used by Stakeholders involved in the Project

Term	Meaning
LLM	Large language model; generates text.
Inference	Running a trained model to produce output.
Semantic routing	Selecting a target model from request needs and policy; the planned router is vLLM Semantic Router.
Serving engine	Software that loads models, schedules requests and returns responses.
Source/target model	The model supplying existing KV state / the router-selected model receiving predicted KV state.
Token/tokenizer	A unit of text / the converter between text and token IDs.
Prefill	Processing prompt tokens to compute attention state before generation.
Decoding	Generating response tokens while extending the KV cache.

Term	Meaning
Attention	Combining contextual information using query, key and value tensors.
Tensor	A multidimensional numeric array.
KV cache	Stored attention-layer key/value tensors for processed tokens.
Source KV cache	The earlier model's KV tensors for a recorded prefix.
Cache handle	A reference to engine-managed KV tensors.
Uncached prompt suffix	All tokens after the cached prefix; still requires target prefill.
KV cache predictor	A versioned mapping from source KV to estimated target KV, possibly fitted/trained.
Predicted target KV cache	Estimated target attention state; accuracy requires evaluation.
Native reference KV cache	Target KV computed by normal prefill for offline ground truth.
Cache provider / loading component	Trusted engine-side code retrieving source KV and loading accepted predictions.
Cross-model KV reuse	Replacing target-prefix prefill with KV predicted from source KV.
Prefix caching	Reusing KV for an exact shared prompt prefix.
Candidate cache	Predicted target KV awaiting runtime validation.
Target-ready cache	KV in the target layout; this alone does not establish accuracy.
Context prefix	The exact shared start of a conversation in target-model tokens.
Native prefill	Standard prompt processing, including available native prefix caching.
Native processing / baseline	The engine without Tandem, retaining its existing cache features.
Fallback	Returning to native processing when prediction/reuse cannot be used.
Compatibility manifest	Versioned list of qualified model pairs, predictors and settings.
Profile / qualification	A model/cache setup / its required approval tests.

Term	Meaning
Reuse domain	Host-assigned group permitted to share specified history.
Cold/warm cache	No useful saved prefix / a relevant cached prefix.
TTFT	Time from request arrival to first output token.
Latency / throughput	Response delay / successful completions per second.
p50 / p95	Median / 95th-percentile measurement.
Logit / greedy generation	Next-token score / choosing the highest-scoring token.
Confidence interval	Range expressing uncertainty across repeated trials.
GPU / CPU	Graphics processing unit / central processing unit.
API	Application programming interface.
V&V / PoC / Rev 0	Verification and validation / proof of concept / initial demonstration.
FR / NFR / SR / DR	Functional / quality / safety-security / documentation requirement IDs.

Data fields use lower-case names with underscores. A request ID identifies one request; a model ID includes the model's exact version, not just its display name. Times use milliseconds unless stated otherwise. Throughput uses requests per second, memory uses bytes and token counts are whole numbers.

5 Relevant Facts And Assumptions

5.1 Relevant Facts

- **Fact 1:** Tandem addresses repeated prefill after model switches by predicting target-model KV state from source-model KV state. The earlier problem statement describes the model-switching problem; its integration-only scope needs updating to include predictor development.

- **Fact 2:** vLLM already supports reusing previously processed shared prompt prefixes; this reduces prefill work rather than the work of generating new tokens.

- **Fact 3:** A cache must match the intended model, conversation and calculation settings. Having the same size is not enough to show it can be reused correctly.
- **Fact 4:** Repeating model results requires controlled software and hardware settings.

The supplied project files do not yet contain measured Tandem performance or evidence that a KV predictor meets its accuracy and performance criteria.

5.2 Business Rules

BR1: Preserve the router-selected target model, including during replay. BR2: Valid cacheless requests retain native processing. BR3: Share cached information only with permitted requests. BR4: Preserve request IDs across responses, errors and measurements. BR5: Support performance claims with requests, settings and output checks. BR6: Qualify each versioned predictor/model pair for KV/output accuracy, privacy and failure handling before enabling reuse.

5.3 Assumptions

A1: Operators control model files/settings and a trusted cache provider; users cannot upload arbitrary caches. A2: Applications supply intended history/new turns, and the router selects the target; Tandem still validates inputs/limits. A3: The server can run the models normally; allocation failures remain tested. A4: At least one predictor can meet output criteria more cheaply than target prefill; this is an unproven proof-of-concept question. A5: Permitted public/synthetic data supports paired KV development and held-out tests. A6: Competing GPU work can be controlled or recorded. Review affected requirements/test plans if assumptions fail.

6 The Scope of the Work

6.1 The Current Situation

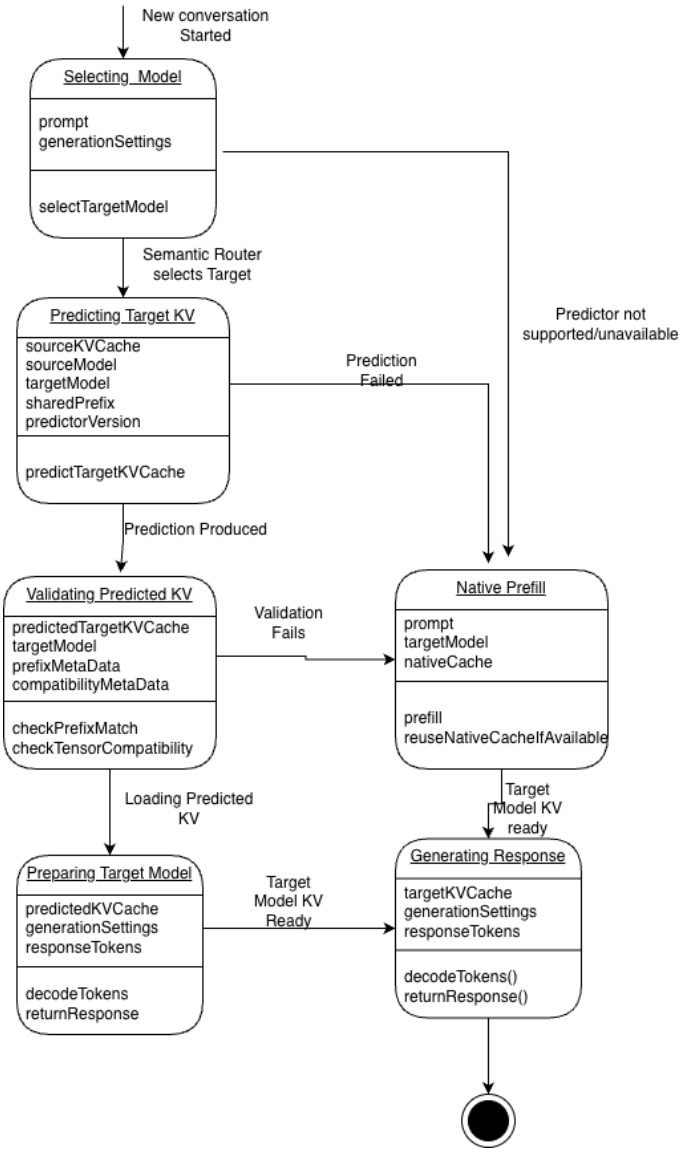


Figure 1: Tandem state diagram for predicting target-model KV caches from source-model KV caches.

6.2 The Context of the Work

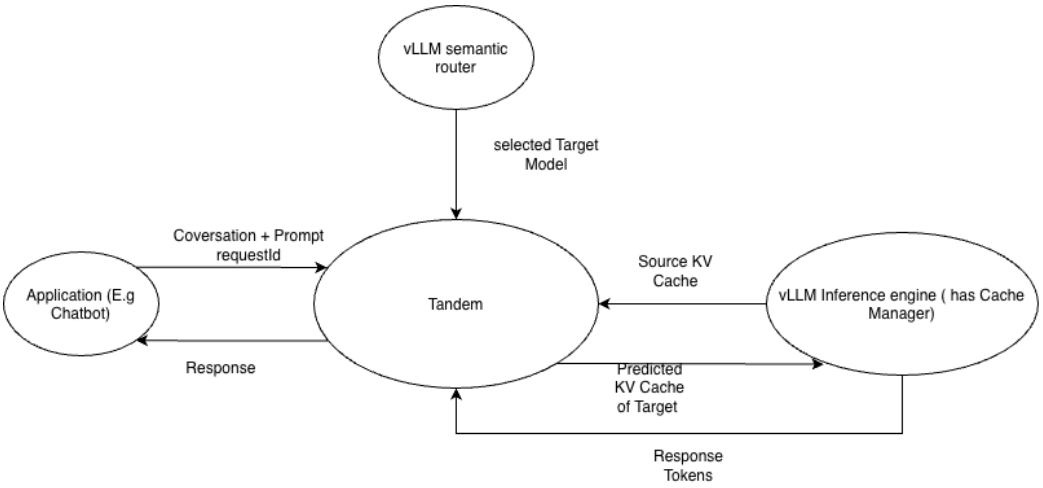


Figure 2: Tandem context diagram showing model selection, source and predicted target KV caches, and application responses.

6.3 Work Partitioning

Part I lists external events and exchanged data; Part II summarizes the corresponding business use cases (BUCs).

Table 2: Work Partitioning Part I

Event Number	**Event Name**	**Input**	**Output**
1	Routed inference request received	Request ID, full prompt, target model selected by vLLM Semantic Router, and generation parameters.	Accepted request with validated inputs, or a rejection with a reason.

Table 2: Work Partitioning Part I

Event Number	**Event Name**	**Input**	**Output**
2	Target KV prediction needed after a model switch	Source-model KV tensors or handle, target model, shared-prefix metadata and supported predictor version.	Predicted target-model KV cache and runtime validation result, or native-prefill fallback with a reason.
3	Response token generated	Generated token, output position and request ID.	Streamed response token associated with the correct request.
4	Request completed or stopped	Completion, failure, cancellation or timeout event.	Final request status and release of resources no active request still needs.
5	Benchmark run requested	Held-out conversations, recorded routing decisions, native target KV references, predictor version and serving settings.	Report of KV prediction error, output checks, prediction time, TTFT, throughput, prompt work and GPU time.
6	Serving configuration changed	Updated model versions, predictor parameters or checkpoints, alignment rules, resource limits or reuse control.	Validated configuration and invalidated incompatible cache references, or a configuration error.

Table 3: Work Partitioning Part II

Event Number	Summary of BUC
1	Validate routed inputs and preserve the target selection; reject invalid requests before generation.
2	Predict target KV from source KV; validate/load, prefill the suffix and decode, or use native fallback.
3	Stream ordered tokens under the correct request ID.
4	Record one final status and release resources without invalidating active shared caches.
5	Measure held-out KV/output accuracy and matched serving performance, including prediction overhead and cost assumptions.
6	Apply valid settings, invalidate incompatible caches and disable unqualified setups.

6.4 Specifying a Business Use Case (BUC)

BUC1: Model-switch inference (events 1–4). Accept the routed request, predict and validate target KV from source KV, prefill the suffix and decode; use native fallback when needed. Return ordered output and one final status, or reject invalid inputs before generation.

BUC2: Evaluate a setup (event 5). Compare held-out KV predictions, outputs, overhead and serving performance against matched native runs.

BUC3: Update serving settings (event 6). Validate the configuration, invalidate incompatible caches and keep unqualified setups disabled.

7 Business Data Model and Data Dictionary

7.1 Business Data Model

A run contains configuration and requests. Each request links its model, sharing group, source KV, versioned prediction/prefix, output and measurements. Comparisons pair native/Tandem requests; accepted requests have one final status, rejections stay separate and applications retain conversation storage.

7.2 Data Dictionary

These are logical inputs and engine-side state, not an HTTP tensor payload; retain the full prompt and derive the suffix after prefix validation.

Field	Allowed value	Purpose
`run_id`	Nonempty text	Identifies a test run.
`request_id`	Nonempty text, unique per run	Links output, final status and measurements.
`history`	Ordered messages	Includes prior conversation and system instructions.
`latest_message`	Text or role/message	New turn in the target conversation format.
`model_id`	Known name and exact version	Router-selected target; fixed during replay.
`settings`	Supported generation controls	Output cap, sampling, stops, optional seed; record defaults.
`reuse_domain`	Host-assigned group ID	Sharing permission; requests cannot self-authorize.
`source_kv`	Optional engine cache reference	Source keys/values for the covered prefix.
`predictor`	Versioned supported mapping	Model pair, method, parameters/checkpoint and alignment.
`candidate`	Optional predicted KV reference	Target KV awaiting validation.
`suffix`	Derived target-token sequence	All tokens after the accepted cached prefix.
`cache_metadata`	Versioned cache description	Source/target/predictor versions, token/layer alignment, prefix/positions, tokenizer/format, precision, sharing group, readiness and configuration version.
`configuration`	Versioned setup	Hardware/software, frozen LLMs, qualified predictors/pairs, limits and reuse switch.
`output_event`	Request ID, order, token/text	An ordered response fragment.

Field	Allowed value	Purpose
`status`	Rejected/completed/failed/cancelled	Failures/cancellations may follow partial output.
`reason`	Defined code	Rejection, no reuse, cancellation, timeout or server/resource failure.
`measurement`	Value with units, or missing/reason	Timings, computed/reused tokens and observed memory.
`comparison`	Matched-run results	Speed, throughput, uncertainty, output differences and failures.

Input plus requested output must fit model/server limits; the output cap is positive and unsupported controls are rejected. Unusable optional caches trigger native processing, while invalid inputs/server failures may prevent completion. Normal exports omit raw conversations and KV.

8 The Scope of the Product

8.1 Product Boundary

Tandem includes source-to-target KV predictor development and fitting, reference-data preparation, engine integration, cache validation/loading, request isolation, fallback and evaluation on one server. The application supplies the prompt and settings; vLLM Semantic Router selects the target. Only supported predictors and exact shared prefixes are reused, and existing native prefix caching remains enabled in the baseline.

Base-LLM training, router development, document retrieval, agent coordination, a graphical application and multi-server execution are outside scope.

8.2 Product Use Case Table

ID	What happens	Required result
PUC1	A model switch has source KV state and a supported predictor	Predict target KV for the shared prefix, load it after runtime checks, prefill the suffix and decode.
PUC2	A cache is missing or fails a check	Process the prompt normally and record why reuse was not used.
PUC3	Reuse fails during processing	Restart safely before output where possible; otherwise report the failure.
PUC4	An operator changes settings	Accept supported settings and reject inconsistent or unsupported ones.
PUC5	An operator or researcher starts a comparison	Run matching requests with and without Tandem and produce a repeatable report.
PUC6	A request is cancelled or times out	Report its final status and release memory no active request needs.

8.3 Individual Product Use Cases (PUC's)

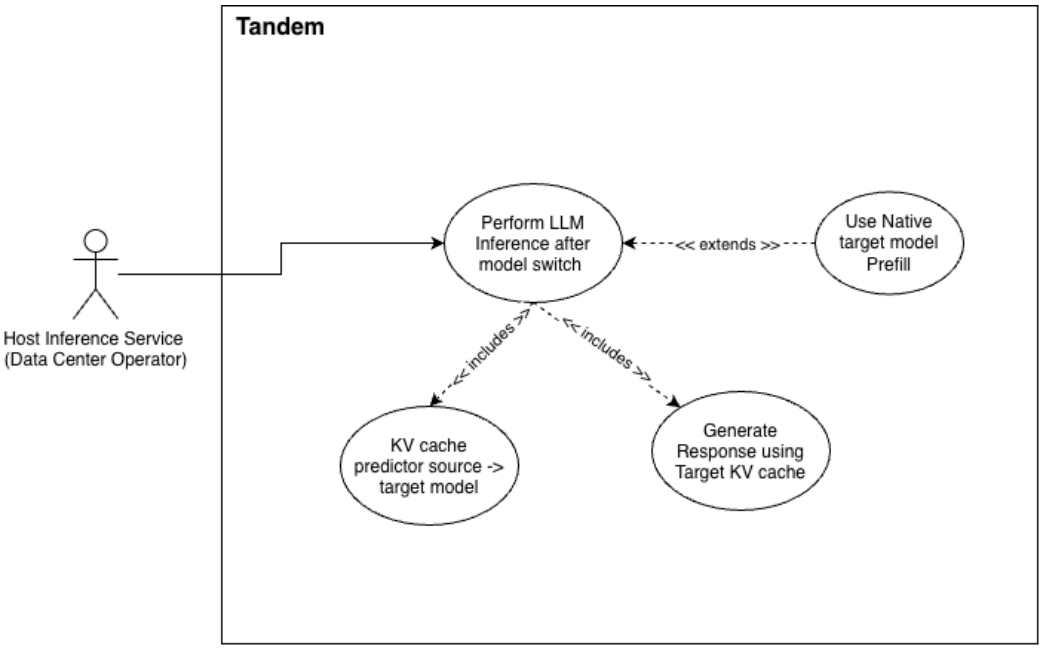


Figure 3: Use case for model-switch inference with source KV available, prediction enabled and a supported predictor.

9 Functional Requirements

9.1 Functional Requirements

Functional requirements describe what Tandem must do; non-functional requirements describe its quality and performance. Each bullet states a requirement, how to check it, and why it matters. "Shall" means required. FR and NFR are requirement IDs. Detailed test procedures belong in the Verification and Validation (V&V) Plan. Draft accuracy and performance limits need team approval before final tests; review them on development data (OI3), without changing thresholds to rescue failed final tests.

- **FR-01 — Check requests before processing**

  Tandem shall validate requests against the Data Dictionary and check

request/cache sizes, references and formats against supported limits before allocating associated resources.

Check. Verify that Tandem rejects missing fields, unknown models, invalid settings, duplicate IDs and oversized prompts before generation, with reasons; valid cacheless requests pass. Check inconsistent lengths, invalid cache handles and unsupported formats; report rejection, native processing or failure without unlimited retries.

Why. Invalid inputs differ from cache misses.

- **FR-02 — Use the requested model and inputs**

  Tandem shall preserve the router-selected model version, complete ordered prompt and accepted generation settings.

  Check. Compare these with native processing; no instruction or history is omitted.

  Why. Preserve application intent.

- **FR-03 — Match the shared history exactly**

  Tandem shall reuse only an exact history prefix in the selected model's formatting and tokenization.

  Check. Vary messages/order, system instructions, tokenizer/format versions and positions; relevant mismatches block reuse. The latest turn remains new input.

  Why. Semantic similarity or equal sizes are insufficient.

- **FR-04 — Validate a KV cache before reuse**

  Tandem shall load a predicted KV cache only after the reuse checks below pass.

  Check. Make each reuse check fail independently, then confirm that a valid cache is accepted. Missing evidence, unsupported versions, invalid shapes and nonfinite tensors block reuse without requiring runtime native-KV comparison.

  Why. Ensure cache eligibility.

- **FR-05 — Load predicted KV state and prefill the remaining prompt**

Tandem shall load accepted predicted KV, prefill the uncached suffix once in order and decode with the selected target model.

Check. Verify covered history, suffix positions and actual prompt work; outputs pass NFR-05.

Why. Replace target-prefix prefill in actual serving.

- **FR-06 — Keep normal processing available**

  Tandem shall retain native processing, including existing prefix caching, when reuse is disabled, unavailable or unsuitable.

  Check. Check same-model turns, unshared history, missing caches and unsupported pairs against native output rules.

  Why. Cache misses must not invalidate requests.

- **FR-07 — Recover from or report a prediction/reuse failure**

  Tandem shall use the recovery rules below when prediction, validation or cache loading fails.

  Check. Inject predictor exceptions, invalid tensors/loading failures before output and execution failures afterwards. At most one safe native restart is allowed before output, retaining the request ID without duplicate output; otherwise report failure. Never combine failed-attempt and restarted output as one successful response.

  Why. Prevent retry loops and mixed answers.

- **FR-08 — Keep output attached to the right request**

  Tandem shall return ordered output under the correct request ID and exactly one final status per accepted request.

  Check. Run overlapping requests, cancellations and failures; accepted requests end completed, failed or cancelled, with separate rejection records.

  Why. Prevent mixed or duplicate responses.

- **FR-09 — Release memory when a request ends**

  Tandem shall release ended requests' resources while preserving cache state needed by active requests.

Check. Repeat completion, failure and cancellation for 100 cycles; no ended request retains its allocation and active outputs remain correct. Policy-retained reusable caches are allowed.

Why. Prevent leaks without discarding useful caches.

- **FR-10 — Record what happened for each request**

  Tandem shall record run and request IDs, source and target models, predictor version, timings, actual prompt work, final status, and the reuse decision and reason in structured results.

  Check. Check fields and counts against a known prompt; unavailable measurements have a missing-value reason, not an estimate.

  Why. Expose saved work and failures.

- **FR-11 — Use matching requests for comparisons**

  Test tools shall compare the same requests, routing decisions, arrival times and generation settings with and without Tandem.

  Check. Check configuration/request records; unexplained differences invalidate comparisons. Baseline prefix caching remains equally enabled.

  Why. Ensure a fair baseline.

- **FR-12 — Make comparison results repeatable**

  Test tools shall report KV error, prediction and response times, throughput, prompt work, total GPU time, GPU time per completed request, output differences and fallback outcomes for individual requests and the complete run.

  Check. Check versions, hardware, models, predictor/checkpoint, splits, seeds, requests, repetitions, cache preparation and failures. Include prediction/validation/loading overhead; separate offline costs and calculate priced estimates under Costs.

  Why. Make results reproducible.

- **FR-13 — Control supported cache-reuse setups**

  Tandem shall validate settings, provide a prediction on/off switch, and record approved model pairs, predictor versions, checkpoints and limits.

Check. Verify that Tandem rejects inconsistent settings, uses native processing for untested combinations and invalidates incompatible caches after changes.

Why. Model loading alone does not qualify reuse.

- **FR-14 — Test a setup before enabling reuse**

  Tandem shall enable a setup only after recording KV prediction error and passing output, privacy and failure-handling tests.

  Check. Verify that missing or failed evidence prevents setup approval. Record held-out evaluation, predictor and parameter versions, data splits and limits. Fitting and final evaluation use separate conversation groups.

  Why. Qualify accuracy before speed claims.

- **FR-15 — Stop cancelled or timed-out requests**

  Tandem shall support cancellation and enforce configured request timeouts.

  Check. Cancel queued and running requests, and test requests exceeding the configured timeout (120 seconds by default). Each stops safely, records the right reason and releases resources under FR-09.

  Why. Running GPU operations need safe stopping points.

- **FR-16 — Predict target-model KV caches from source-model KV caches**

  Tandem shall predict the target model's key and value tensors from the source KV cache and alignment metadata for the shared history. It shall use an approved model pair and predictor version, without running target-model prefill over that history to construct the prediction.

  Check. Using held-out conversations, check the prefix, cache layout and model metadata, and measure KV error for each layer or cache group against offline native references. Instrumentation confirms skipped target-prefix prefill; loading/output pass FR-05 and NFR-05. Record architecture, parameters and fitting procedure.

  Why. Source-to-target KV prediction is Tandem's core function.

KV prediction and reuse rules. Tandem's core operation is:

$$\widehat{\mathrm{KV}}_{\mathrm{target}} = f_\theta(\mathrm{KV}_{\mathrm{source}}).$$

The left side is the predicted target cache. The function f_θ is the predictor, with versioned parameters θ, for a supported source/target model pair and recorded token/layer/head alignment. Base LLM weights stay frozen.

- **Before reuse:** The request must be valid, prediction enabled, and the predictor/model pair approved under FR-14. The predicted cache must cover an exact, nonempty prefix of the current history, with matching target formatting, tokenizer and positions. Source and target token IDs may differ if the approved alignment maps them to the same history.
- **Cache checks:** Require sharing permission, complete tensors, correct shapes/layout and precision, finite values, current versions and enough resources without changing caches still in use. Missing evidence or any failed check blocks reuse; equal tensor sizes alone are insufficient.
- **Processing:** Reject invalid requests. For valid requests, load an accepted prediction and prefill all remaining tokens once in order, or use native processing when reuse is unavailable. No prompt token is skipped or logically processed twice. Runtime checks do not prove prediction accuracy or require computing native target KV for comparison.
- **Failure handling:** Restart natively only before any output, from valid state, at most once. Otherwise report failure; never append a restarted answer to partial output. Stop cancellations/timeouts safely and record exactly one final status: completed, failed or cancelled.

Accuracy checks.

- Compute native target KV offline as the reference for the same history. For each declared layer/cache group, report absolute prediction error as a Frobenius norm, and relative error as that error divided by the reference norm, separately for keys and values. If the reference norm is zero, report absolute error and mark relative error undefined. Small KV error alone does not establish acceptable outputs.

- Compare corresponding logits (next-token scores) after identical prior tokens. The draft limit on the absolute difference is 0.001 plus 0.001 multiplied by the absolute native score. Apply token masks consistently; unexpected nonfinite values fail.

- Greedy decoding must return identical tokens for 32 tokens, or until both runs stop earlier. First verify native-run repeatability. Sampling settings need separate recorded qualification or native fallback; equal seeds alone do not guarantee matching outputs. These checks apply to the declared test set/settings, not every possible prompt.

Final tests use paired source/native-target KV for held-out conversations; no fitting or threshold tuning on final results. Cover declared limits, same-model/switching requests, shared/unshared histories, cold/warm caches, accepted/missing/rejected predictions, concurrency and forced failures. Match prompts, routing, settings, arrivals and resource limits; record cache preparation/hits and explain differences.

Start with three untimed warmups and ten alternating timed pairs, adding trials if uncertainty is too large; record cache preparation and report skips, failures, uncertainty, slowdowns and negative results. Throughput counts successful completions from first arrival to last final status. TTFT is missing if no token returns; separately report end-to-end time, actual prompt work and directly measured memory.

10 Look and Feel Requirements

10.1 Appearance Requirements

Not Applicable.

10.2 Style Requirements

Use glossary terms, defined status codes and text-readable outcomes without relying on colour; generated responses are not stylistically rewritten.

11 Usability and Humanity Requirements

11.1 Ease of Use Requirements

Ordinary requests use the existing serving interface; only technical users configure KV prediction.

- **NFR-02 — Make setup instructions usable**

  A technical user with the listed prerequisites shall be able to install Tandem and run the basic example by following its instructions.

  Check. Have an independent reviewer follow all documented commands/results in a clean supported environment.

  Why. Operators need repeatable setup.

11.2 Personalization and Internationalization Requirements

Not Applicable.

11.3 Learning Requirements

Provide a supported installation example explaining KV prediction, fallback and results under DR-01 and NFR-02.

11.4 Understandability and Politeness Requirements

Messages identify outcomes, invalid fields and corrective actions without exposing private prompts or cache data.

- **NFR-01 — Make results readable**

  Tandem shall provide readable results with clear field names and units, plus structured data that other programs can read.

  Check. Verify that reports distinguish reuse, native processing, fallback and failure without inspecting engine internals.

  Why. Developers/operators need understandable results.

11.5 Accessibility Requirements

Commands, reports and instructions use readable text, require no particular visual editor, and provide machine-readable results without colour-dependent meanings.

12 Performance Requirements

12.1 Speed and Latency Requirements

Measure TTFT, end-to-end latency and throughput on preselected workloads, including non-reuse cases. Timing includes queueing, source retrieval, prediction, checks/loading, suffix prefill and decoding; handle model loading consistently. Separate predictor/loading/engine timings and offline development costs, and retain failures and cancellations.

Compare median TTFT, p95 TTFT (the 95th-percentile wait) and successful requests per second against the native engine. Estimate confidence intervals at the draft 95% confidence level using 2,000 bootstrap resamples of complete paired trials; record the random seed. Requests within one trial are not independent repetitions. Use the conservative interval bounds to decide whether an improvement or allowed slowdown has been demonstrated.

- **NFR-04 — Show a useful reduction in waiting time**

  On at least one preselected model-switch workload, Tandem shall reduce computed prompt work and median TTFT beyond timing variation.

  Check. Require fewer computed prompt tokens and a confidence interval confirming lower median TTFT, with passing output checks and no additional failures. Report all workloads, including slowdowns.

  Why. Demonstrate measured benefit.

12.2 Safety-Critical Requirements

Not Applicable.

12.3 Precision or Accuracy Requirements

Evaluate KV error and outputs separately. Review strict draft tolerances on development data/native variation before final tests (OI3); runtime validation cannot prove accuracy, and output consistency does not prove factual correctness.

- **NFR-05 — Keep the selected model's output consistent**

  Each approved setup shall pass the logit and generated-token checks in the KV prediction and reuse rules on its declared test set.

  Check. Apply the draft logit tolerances in the accuracy checks above. Require identical greedy output for 32 tokens or an earlier shared stop. Mismatches prevent approval; do not change limits after failed final tests.

  Why. Draft tolerances need OI3 review.

- **NFR-03 — Preserve supported text**

  Tandem shall preserve valid Unicode text through the selected model's normal conversation formatting and tokenization.

  Check. Check accents, non-Latin scripts, whitespace and delimiters; prompt tokens equal native processing.

  Why. Prevent false prefix matches.

12.4 Robustness or Fault-Tolerance Requirements

A failed optional prediction uses native processing when possible; unavailable models or exhausted resources produce an explicit failure.

- **NFR-06 — Report failures and allow later normal requests**

  Tandem shall explain failures and retain native processing after recoverable cache errors.

  Check. Inject incomplete/mismatched caches, allocation failures and timeouts; follow FR-07. Later valid requests complete if the engine remains healthy; unrecoverable engines require operator action.

  Why. Recovery requires a working engine/resources.

12.5 Capacity Requirements

Publish and test the supported prompt lengths, concurrency and memory limits for each model/hardware setup.

- **NFR-07 — Handle more requests without excessive delay**

  On at least one preselected shared-history workload, Tandem shall improve throughput within the permitted p95 TTFT increase.

  Check. Require confidence intervals showing higher throughput and a p95 TTFT increase no greater than 5% of the native value, with no extra failures. Record requests and arrival times.

  Why. Capacity gains must retain acceptable delay.

- **NFR-08 — Limit slowdowns for normal requests**

  For requests that do not use prediction, Tandem shall keep latency increases and throughput losses within the allowed limit.

  Check. Use the conservative confidence-interval bounds: median and p95 TTFT may increase by at most 5% of their native values, and throughput may fall by at most the same fraction. Excess uncertainty is inconclusive.

  Why. Bound integration overhead.

12.6 Scalability or Extensibility Requirements

The initial scope is one server and tested model pairs; extra GPUs, model families, conversation branches and multi-server scaling require further qualification or expanded scope.

- **NFR-09 — Publish supported models and limits**

  Each release shall list tested setups, limits on simultaneous requests and tokens, and unsupported settings.

  Check. Test boundaries and just-beyond values; support claims require evidence. Untested setups remain experimental or disabled.

  Why. Distinguish tested support.

12.7 Longevity Requirements

Maintain the prototype through the final capstone release, retain versioned code/test records, and qualify model or dependency updates before claiming support.

13 Operational and Environmental Requirements

13.1 Expected Physical Environment

Not Applicable.

13.2 Wider Environment Requirements

GPU results require the recorded Linux/model/software setup; local CPU checks do not establish serving performance.

- **NFR-10 — Record how to recreate the test setup**

  Each tested release shall record the hardware, software and commands needed to recreate the setup on compatible hardware.

  Check. Rebuild the setup and run basic and output tests; identify model/dependency/build versions and report hardware/competing-load differences for performance repeats.

  Why. Setup affects timing and output.

13.3 Requirements for Interfacing with Adjacent Systems

Use the existing serving interface, router-selected target model and engine-managed source KV reference; load validated predicted target KV through the engine integration. The Data Dictionary defines these fields, and the application assigns user identity and sharing permissions.

13.4 Productization Requirements

Not Applicable.

13.5 Release Requirements

Publish versioned code, dependencies, supported setups, limits and test results; failed or untested combinations remain experimental or disabled.

14 Maintainability and Support Requirements

14.1 Maintenance Requirements

Not Applicable.

14.2 Supportability Requirements

Use repository issues with version details, error codes and non-private reproduction steps; no support response time is promised.

- **NFR-12 — Explain status and error codes**

  The release shall explain rejection, fallback and failure codes, including what the operator should do or whether no action is needed.

  Check. Compare codes with triggered outcomes; diagnostics identify the setup for reproduction while meeting SR-06.

  Why. Distinguish unused caches from failed requests.

14.3 Adaptability Requirements

Not Applicable.

15 Security Requirements

15.1 Access Requirements

Not Applicable.

15.2 Integrity Requirements

Use only complete, current caches and protect state needed by active requests.

SR-02 — Do not use incomplete or incompatible caches

Tandem shall reject incomplete, inconsistent, outdated, unsupported or unqualified caches.

Fit criterion. Vary each condition; rejected caches never enter generation. Use native processing or report server/resource failure; missing evidence means unsupported.

Source and rationale. C4, BR6; prevent invalid cache use.

SR-03 — Protect caches still in use

Tandem shall protect active caches from invalidating removal, modification or reassignment.

Fit criterion. Overlap generation, cancellation and cache replacement/eviction; outputs pass NFR-05 and cleanup passes FR-09.

Source and rationale. G2, G4; preserve active state.

15.3 Privacy Requirements

KV state can contain private information; restrict sharing and keep private content out of normal logs. Published tests use public or synthetic data.

SR-01 — Allow only permitted history sharing

Tandem shall reuse history only with host-granted sharing permission.

Fit criterion. Use matching histories across groups and request-controlled permission changes; unauthorized caches are neither used nor exposed.

Source and rationale. BR3; protect private history.

SR-04 — Keep private instructions and answers separate

Tandem shall exclude other requests' private instructions/answers from reused history.

Fit criterion. Vary another request's private instructions while fixing the current request; scores pass NFR-05, greedy tokens match and native controls pass.

Source and rationale. BR3; shared history grants no access to later private text.

SR-05 — Invalidate old caches after restart or setup changes

The first release shall restrict caches to the serving lifetime and invalidate references after restart/incompatible configuration changes.

Fit criterion. Verify that Tandem rejects old references; default exports contain no prompts/caches.

Source and rationale. BR3; host handles physical GPU-memory erasure.

SR-06 — Keep private contents out of normal logs

Normal logs/measurements shall retain IDs/reasons but omit raw prompts, answers, KV, credentials and request-controlled secrets.

Fit criterion. Check exports for injected secrets. Detailed output traces require explicit enablement, public/synthetic data and a declared retention period.

Source and rationale. BR3; minimize sensitive records.

15.4 Audit Requirements

Not Applicable.

15.5 Immunity Requirements

Not Applicable.

16 Cultural Requirements

16.1 Cultural Requirements

Not Applicable.

17 Compliance Requirements

17.1 Legal Requirements

Tandem uses the repository's MIT licence; retain model/data/dependency licence conditions, required notices and permissions, and leave deployment-specific privacy obligations to the adopting organization.

17.2 Standards Compliance Requirements

Follow the Volere outline and course checklist; document tested interfaces under NFR-09 without claiming external certification.

18 Open Issues

The following decisions still need to be made or confirmed:

ID	What needs deciding	When it is needed
OI1	Choose the source/target LLM pair, predictor architecture, token/layer alignment, paired KV dataset and any fitting/training procedure.	During the proof of concept; before prediction is enabled under FR-14.
OI2	Choose exact software/model versions, calculation settings and GPU hardware.	Before correctness and timing tests.
OI3	Approve cache-error reporting, downstream output tolerances and acceptable slowdown for the chosen predictor. Review the strict draft output checks using development data and normal-run variation.	Before final held-out evaluation; do not adjust limits to rescue failed final tests.
OI4	Ask potential users or representatives for feedback and record which requirements it affects.	Before approving the SRS for submission.
OI5	Align the Problem Statement, Development Plan and Hazard Analysis with target-KV prediction from source KV, predictor development/training and engine integration. Keep base-LLM training and router development outside scope.	Before final technical task assignments and the proof-of-concept review.
OI6	Review cache-related risks in the Hazard Analysis and keep assumptions and mitigations consistent with the SRS.	During the SRS/Hazard Analysis review.

ID	What needs deciding	When it is needed
OI7	Confirm GPU time, storage and permissions to use the chosen models and test data.	Before booking and running experiments.
OI8	Complete personal reflections, course details and the required disclosure from the team's actual records.	Before final submission.

Recording these decisions does not mean they have already been resolved.

19 Off-the-Shelf Solutions

19.1 Ready-Made Products

Use the serving engine's existing inference and prefix-caching features as the baseline, then evaluate Tandem's added benefit after model switches.

19.2 Reusable Components

Reuse model loaders, tokenizers, tensor/cache operations and profiling tools, then validate the combined system for accuracy, privacy and failure handling.

19.3 Products That Can Be Copied

Not Applicable.

20 New Problems

20.1 Effects on the Current Environment

Prediction, validation, data movement and bookkeeping can add latency and memory use, so comparisons must include requests that do not benefit from reuse.

20.2 Effects on the Installed Systems

Engine updates may break cache integration; publish supported versions, retain native processing and verify unchanged model selection and request settings.

20.3 Potential User Problems

Document unsupported model pairs, cache misses and fair baseline conditions; matching native model output does not establish factual correctness.

20.4 Limitations in the Anticipated Implementation Environment That May Inhibit the New Product

Insufficient GPU memory, limited cache-insertion/profiling interfaces, prediction error, numerical variation or excessive predictor/loading overhead may prevent a setup from meeting accuracy or performance goals.

20.5 Follow-Up Problems

Disable and requalify affected setups after model/engine changes or output mismatches; review infeasible prediction approaches with course staff rather than claiming the original goals were met.

21 Tasks

21.1 Project Planning

Track scope review, model-pair selection, paired KV data, predictor development, engine integration and evaluation through repository issues and reviews; the team assigns task owners.

21.2 Planning of the Development Phases

Confirm these Development Plan windows against course dates:

Phase / window	Work and completion evidence
SRS/HA; Oct 2026	Review scope, user feedback, risks and open decisions (OI3–OI6).
Proof of concept; Nov 2026	Choose models; prepare paired KV data; develop/fit predictor. Establish native repeatability and held-out KV/output accuracy before claiming serving gains.
Integration; Dec 2026–Jan 2027	Demonstrate prediction/loading, suffix prefill, native fallback, cancellation and privacy/failure tests on one qualified pair.
Initial demonstration; Feb 2027	Deliver a repeatable prototype, core requirement tests, setup records and instructions.
Final evaluation; Feb–Mar 2027	Complete held-out accuracy, paired speed/capacity/cost tests and uncertainty reports; rerun affected core checks after changes.
Final release / EXPO; Apr 2027	Review supported limits, reports, traceability and team-approved reflections.
Future work	Review Waiting Room scope and tests before implementation.

22 Migration to the New Product

22.1 Requirements for Migration to the New Product

Verify native processing, install Tandem and enable only qualified predictors; rollback by disabling prediction or restoring the recorded engine version.

22.2 Data That Has to be Modified or Translated for the New System

Not Applicable.

23 Costs

Main costs are development effort, paired KV data, predictor fitting/training, GPU experiments, storage and maintenance; confirmed estimates remain

open under OI7. Report offline development costs separately from serving costs.

For each matched workload, include source-cache handling, prediction, validation/loading and failed-request resource use in total GPU time; divide by successful completions. If rates are known, multiply allocated GPU-hours by those rates, add stated charges and report cost per completion; otherwise report GPU time without claiming money savings. State any amortization, request-volume or retraining assumptions. Keep routing decisions fixed for Tandem comparisons and report cheaper-model selection savings separately with prices and task-quality checks.

24 User Documentation and Training

24.1 User Documentation Requirements

Provide repository instructions for installation, operation, testing and troubleshooting.

DR-01 — Explain installation and operation

The release shall document prerequisites, supported setups, fields, reuse controls, native processing, errors, rollback and a repeatable example.

Fit criterion. Check coverage, limits/problems and independent example execution under NFR-02.

Source and rationale. Technical users need operational guidance.

DR-02 — Explain test results

The release shall explain reproducing/interpreting native–Tandem tests, including cache preparation, uncertainty, failures and output tolerances.

Fit criterion. Trace a sample report to its settings and individual results, and distinguish pass, failure and inconclusive outcomes.

Source and rationale. G1–G4; make evidence assessable.

24.2 Training Requirements

Not Applicable.

25 Waiting Room

Future work may include adding model/predictor setups without changing application request fields, more model families, conversation branches, GPUs, lower memory use and another serving engine; multi-server execution, semantic-prefix reuse, router development and a public application require new scope.

26 Ideas for Solution

Develop a source-to-target KV predictor using paired caches from frozen LLMs, accounting for token/layer/head alignment and cache layout. Choose the architecture and any fitting method at proof of concept, keeping training and held-out conversations separate. Integrate prediction, runtime checks and loading so only the uncached suffix requires target prefill; retain native fallback and evaluate KV error, output effects and total serving cost. Tree-attention and KV-Promise features from the older plan are not required.

Likely and unlikely changes. LC1: New model pairs need editable configuration/tests. LC2: Engine cache/interface changes need an identifiable integration boundary. LC3: New workloads/hardware need comparable test records. LC4: Preapproval threshold revisions need versioned values/reasons. UC1: Base-LLM training and routing remain external. UC2: Initial execution remains on one server. UC3: Reuse retains exact-prefix and permission checks, not semantic similarity.

27 Appendix — Generative AI Use Disclosure

27.1 AI Tools Used

Generative writing assistance supported this draft; the team must complete tool-identification details in its course disclosure record (OI8).

27.2 How and Where Used

Assistance covered drafting/organizing requirements, fit criteria, formalization, traceability and evaluation defaults; stakeholder consultations and experiments are not claimed as completed.

27.3 Verification of AI-Generated Content

Scope/template/checklist, source references and compilation were checked; these do not validate prediction or performance. Before approval, the team must verify technical claims/thresholds, reconcile the Hazard Analysis and complete actual verification records and reflections.

28 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:

These draft observations and learning plans need team review; they do not assert individual experiences or course histories.

1. What went well while writing this deliverable?

   The scope records helped identify inputs, outputs and boundaries. Pairing requirements with fit criteria exposed missing failure cases; Volere prompted attention to installation and maintenance alongside performance.

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

   Older planning documents differ from the current KV-prediction scope; OI5 tracks reconciliation. Model selection, numerical tolerances and user feedback remain open rather than being presented as settled decisions.

3. How many of your requirements were inspired by speaking to your client(s) or their proxies (e.g. your peers, stakeholders, potential users)?

   No confirmed count is available. The draft uses project records and potential-user needs; record consultation-derived requirements after actual feedback.

4. Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project.

   Requirements engineering, testing, operating systems, algorithms, computer architecture and machine learning support this work. Team members must add their actual course histories.

5. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

   Collectively, the team needs transformer/KV alignment, predictor design and fitting, dataset splitting, serving-engine integration, GPU memory and failure handling, controlled benchmarking, requirements/hazard analysis and clear technical communication. Share critical knowledge across members.

6. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   For each area, combine study with practice: engine documentation/code with request tracing; prediction papers/implementations with paired-KV mapping experiments and supervised fitting; numerical/dataset methods with native-reference accuracy tests; profiling tutorials with forced allocation/cancellation cases; published benchmark methods with synthetic repeated workloads; and course requirements/hazard examples with success/failure walkthroughs. This proposed combination connects theory to observed behaviour. Keep final evaluation separate from development. The team must confirm individual choices and reasons from actual experience before submission.