Development Plan
Tandem

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

Table 1: Revision History

Date	**Developer(s)**	**Change**
Sept 22, 2026	William Pol	Initial draft for TA meeting
Sept 27, 2026	William Pol	Revised against the Development Plan checklist and rubric; added PoC risks, workflow details, role rotation plan, and course schedule
Sept 28, 2026	William Pol	Addressed pull request review; completed the team charter and reflection

About This Document

This document records how Team 11 will build Tandem: how we meet and communicate, who is responsible for what, how code and documents move through git, when major pieces of work happen, how the Proof of Concept (PoC) demonstration will test our biggest risk, and which tools we expect to use. The problem and goals themselves are described in the Problem Statement and Goals document. This plan is written for two audiences: the current team, as the reference for how we agreed to work, and viewers of the document, such as the teaching assistant (TA), the instructor, and peer reviewers, who should be able to read it and understand how the project is run without asking the team directly. Decisions recorded here can change; changes are made through a pull request that updates this document.

Table 2 defines the terms and acronyms used in this document. Each acronym is also spelled out where it is first used.

Table 2: Terms and acronyms

Term	**Meaning**
LLM	Large language model.
Prefill	The first phase of LLM inference, in which the model processes the whole prompt at once.
Decode	The phase after prefill, in which the model generates output one token at a time.
KV cache	The keys and values computed for each prompt token during prefill, stored so they do not have to be recomputed during decode.
Trunk	The part of a prompt shared by several agents (for example, a document all agents read).
Branch	The part of a prompt specific to one agent (its instructions).
Tree attention mask	An attention mask that lets several branches share one trunk in a single forward pass while never seeing each other.
vLLM	An open-source LLM serving engine that Tandem will extend.

1 Confidential Information?

This project does not involve any confidential information from industry. The team does not have an industry supervisor, so there is no non-disclosure agreement (NDA) or other confidentiality agreement in place.

2 IP to Protect

There is no intellectual property (IP) to protect. Tandem is an open-source systems project that extends existing open-source large language model (LLM) inference engines, with no industry sponsor, so no IP agreement or Intellectual Property Guide Acknowledgement is required.

3 Copyright License

The team is adopting the MIT License. The license text, with the team members as copyright holders, is in the `LICENSE` file at the root of the project repository.

4 Team Meeting Plan

When and where. The team meets once a week for up to one hour, on Mondays from 6:00 to 7:00 pm, separate from the Wednesday TA-team meeting slot. Meetings alternate between in person on campus and online through a Microsoft Teams call, so that at least every second meeting is in person. Physical meeting locations are shared in the team chat, not in this public repository. When a lecture or tutorial slot has nothing scheduled, the team uses it as an extra working session. In the week before a deliverable is due, the team adds a second short check-in.

Agenda and minutes. Every team meeting has a GitHub issue, created from the `Team Meeting` issue template at least 24 hours beforehand. The chair writes the agenda in the issue. The standing agenda is:

1. Review action items from the last meeting.
2. Blockers: anything stopping someone's work.
3. Status of the next deliverable and the checklist items it still needs.
4. Decisions needed this week.
5. New action items, each with one owner and a due date.

During the meeting Granola records notes and a summary automatically; the chair ticks off attendance in the issue and turns that summary into decisions and action items as comments. Every action item becomes its own issue, assigned to its owner, and the meeting issue is closed once those issues exist.

Chair. The chair rotates weekly in alphabetical order by last name. This gives every member the chair role several times each term.

Supervisor meetings. We do not yet have a supervisor; the team is contacting potential advisors, who do not need to be at McMaster. Once one is confirmed, we plan to meet every two weeks on Microsoft Teams, tracked with the `Supervisor Meeting` issue template.

5 Team Communication Plan

- **Microsoft Teams** is for day-to-day discussion, quick questions, and online meetings. Members are expected to reply to messages that mention them within 24 hours on weekdays. Anything urgent (for example, a broken build close to a deadline) is posted with an @-mention of the people needed.
- **GitHub issues and pull requests** are the record of work. Any decision made in chat or in a meeting that affects the code or the documents is written into the relevant issue or PR, so the reason behind it can be found later. If it is not in GitHub, it was not decided.
- **The TA and instructor** are contacted by the team liaison, Justin Kwinecki, with the rest of the team copied. Course discussions on the capstone GitLab repository are handled the same way.
- **Files** (documents, code, data, results) live in the repository, not in chat, so there is one current copy.

6 Team Member Roles

Roles are split into project roles, which keep the team running, and technical areas, which cover the expertise the project needs. No one owns a single narrow role for the whole year.

6.1 Project Roles

Table 3 lists the roles that keep the team running.

Table 3: Project roles

Role	Responsibility and assignment
Liaison	Point of contact with the instructor and TA. Justin Kwinecki, for the whole year.
Meeting chair	Writes the agenda, runs the meeting, ticks off attendance, and turns Granola's notes into decisions and action items. Rotates weekly.
Reviewer	Reviews a pull request before it is merged. Anyone except the author; the author requests a reviewer from outside their technical area at least once a week, to spread knowledge.
Quality assurance (QA)	For each deliverable, one member who did not write most of it checks it against the matching checklist in `docs/Checklists/` and the course rubric, and files issues for anything missing. Rotates per deliverable.
Deliverable lead	For each document or demo, one member coordinates who writes which part and makes sure it is submitted on time. Rotates per deliverable, so every member leads at least one each term.

6.2 Technical Areas

Table 4 lists the three broad areas of technical work. Documentation is shared by everyone and coordinated by the deliverable lead.

Table 4: Technical areas

Area	What it covers
Core mechanism	The tree attention mask, its correctness tests, and making it run fast on graphics processing units (GPUs).
Serving engine	Changes to vLLM: shared memory for trunks, scheduling, and the KV-Promise mechanism, which lets a late request wait for a shared key-value (KV) cache that is still being computed.
Measurement and tooling	Benchmarks and experiments, access to the department GPU clusters, the Python environment, and continuous integration (CI).

How technical areas are assigned. Until the PoC demonstration in November, technical areas are not assigned. No one on the team has prior GPU-serving experience, so every member first works on the PoC (the vLLM measurement

and the tree attention mask tests). This gives everyone a shared understanding of the core mechanism and shows who is drawn to which area. At the first team meeting after the PoC demo, each area is given a primary owner and one or two backups, based on interest and what members learned during the PoC. Every member works in at least one area, and every area has at least two people who understand it.

Changing roles. Roles are reviewed at the first team meeting after each major demo (PoC, Rev 0). Any member can also ask to change areas at a weekly meeting. When an area changes hands, the outgoing owner writes a short handover issue (current state, open problems, where the code is) and pairs with the new owner for one week.

7 Workflow Plan

This section is written to become the project's `CONTRIBUTING.md`.

7.1 Branches and Commits

- `main` is protected: nothing is pushed to it directly, and every change arrives through a pull request (PR).
- Work happens on a short-lived branch named `type/short-description`. The type is one of `feat`, `fix`, `docs`, `poc`, `bench`, or `chore`.
- Commits are small and focused, with a message in the imperative mood that explains why the change was made, not just what changed.

7.2 Pull Requests

- Every PR links the issue it resolves ("Closes #12"), describes what changed, and says how it was tested.
- A PR needs one approving review from another team member and passing CI before it is merged. Work in progress is opened as a draft PR so others can see it early.
- Reviewers check correctness and readability, and for documents, the matching checklist in `docs/Checklists/`.
- PRs are merged with "squash and merge" so each PR becomes one commit on `main`, and the branch is deleted afterwards.

7.3 Tags

When a deliverable is submitted, the commit on `main` that was submitted is marked with an annotated tag (for example, `PS-DevPlan`, `SRS-HA`, `PoC`, `Rev0`, `Final`). This lets the TA, the instructor, and peer reviewers see exactly what was graded, and lets the team compare later work against it.

7.4 Issues

- Every piece of work, bug, research question, and meeting is an issue. The repository already has issue templates for team meetings, TA meetings, supervisor meetings, lectures, and peer reviews.
- Each issue gets one *type* label and, for code, one *area* label:
  - Type: `bug`, `enhancement`, `documentation`, `research`, `meeting`, `review`, `question`.
  - Area: `area:core`, `area:engine`, `area:measurement`.
- Each issue has one assignee and belongs to a milestone. There is one milestone per course deliverable, with the deliverable's due date.
- Peer review issues from other teams (label `review`) are answered within a week and closed with a link to the commit that addresses them, or a short explanation if no change is made.

7.5 Continuous Integration

- **Documents (already running):** a GitHub Actions workflow (`.github/workflows/latex-pages.yml`) rebuilds every changed document, commits the PDFs to `pdfs/`, and publishes them to the project's GitHub Pages site.
- **Code (added with the first code PR):** a second workflow runs the linter and the unit tests on every PR. This workflow includes the tree attention correctness test with a small model on the central processing unit (CPU), so a change that breaks the mask fails CI before it is merged.
- **GPU work:** GitHub's CI machines have no GPU, so GPU tests and benchmarks are run by hand on the department clusters, and their results are committed to the repository with the command used to produce them.

8 Project Decomposition and Scheduling

The team uses a GitHub Project board, Tandem Kanban, with the columns Backlog, Ready, In Progress, In Review, and Done. Every issue is on the board, and the board is reviewed during each weekly meeting.

Table 5 combines the course deadlines with our technical milestones. Dates for Term 1 are from the course's weekly announcements; later dates follow the course schedule and will be updated as they are announced.

Table 5: Schedule

When	Course deliverable (and our technical work)
Sept 28	Problem Statement, PoC Plan, Development Plan, Team Charter
Oct 1	Peer review of another team's Problem Statement and Plans
Oct 19	Software Requirements Specification (SRS) and Hazard Analysis
Oct 22	Peer review of another team's SRS and Hazard Analysis
Week 7	Verification and Validation (V&V) Plan; PoC schedule requests
Week 8	Design Document Rev −1 and Fall custom document (machine learning (ML) model report); PoC productivity report
Mid–late Nov	**PoC demonstration**
Sept–Nov	*PoC work: measure the gap in unmodified vLLM on a department cluster; tree attention mask in PyTorch and its correctness tests; read related work [**?**, **?**, **?**] to sharpen the novelty claim*
Dec–Jan	*Fast tree attention (FlexAttention); vLLM integration of shared trunk memory and per-branch position IDs*
Jan (Week 16)	Design Document Rev 0; module implementation
Feb (Weeks 18–19)	**Rev 0 demonstration**
Feb–Mar	*KV-Promise for staggered requests; benchmarks against vLLM on the department GPU clusters*
Mar	V&V Report and Winter custom document (Performance Report)
Late Mar	**Final demonstration**
Apr	Final documentation, Capstone EXPO poster and video, EXPO

9 Proof of Concept Demonstration Plan

The PoC demonstration tests the two things Tandem's thesis depends on: that the problem is real in today's serving engines, and that our core mechanism is correct. It does not build the fix inside vLLM; that is the work planned after the PoC.

9.1 Risks

The PoC is planned around the risks most likely to stop the project:

1. **The problem may already be solved (thesis risk).** Tandem assumes that when several agents with the same long shared context (trunk) arrive at a serving engine at the same time, the engine processes the trunk once per agent, because its prefix cache is still empty. If current versions of vLLM already avoid this, most of the project's benefit disappears.

2. **Correctness of the core mechanism.** Tandem depends on evaluating a shared trunk and several branches in one forward pass with a tree attention mask, and getting *exactly* the same result as running each branch on its own. If a branch can see another branch, or tokens after itself, or gets the wrong positions, the results are silently wrong and nothing built on top can be trusted.

3. **Learning curve and hardware access.** No team member has worked on an LLM serving engine before, vLLM is a large codebase, and realistic measurements need GPUs on the department clusters, which the team has not yet used.

9.2 What the Demonstration Will Show

- **Risk 1: measuring the gap in unmodified vLLM.** We run the current release of vLLM, with prefix caching turned on, on a department GPU cluster. A script sends K requests that share one trunk and differ only in a short agent-specific branch, in three scenarios:

    1. *Simultaneous arrival:* all K requests are sent at once, with nothing cached.
    2. *Warm cache:* the trunk is processed once first, then the K requests are sent.
    3. *Staggered arrival:* the requests arrive a few milliseconds apart.

    For each scenario, and for several values of K and trunk length, we report the time to first token (TTFT) and the number of prompt tokens vLLM actually processed (taken from its metrics, not estimated). Since vLLM's paged KV cache uses a fixed number of bytes per token, prompt tokens processed is also a direct stand-in for KV cache memory: fewer tokens computed means less KV cache memory allocated. If simultaneous arrival processes close to K copies of the trunk while the warm cache processes one, the gap Tandem targets is real, and the difference between the two scenarios is the most Tandem can save. Reporting actual GPU memory usage and KV cache read/write throughput directly, rather than deriving them from token counts, is a backlog item for later benchmarking once the measurement moves past this PoC scale.

- **Risk 2: tree attention correctness.** A PyTorch program packs a shared trunk and several branches into one sequence, builds the tree attention mask and per-branch position IDs, runs one forward pass, and compares every branch's output scores (logits) against running that branch alone. The comparison covers the next token and several generated tokens, and must match to within floating-point rounding. The mask is printed so the structure can be checked by eye.
- **Risk 3: learning curve and hardware.** Running and measuring vLLM for Risk 1 requires installing it on a department cluster and reading enough of its scheduler and cache code to explain the result. The team will also present a short plan for the vLLM changes (which parts of the engine change, and why), showing that we understand where the work lies.

9.3 If the PoC Fails

If the Risk 1 measurement shows that vLLM already avoids most of the repeated work, the team will review the thesis with the TA before building anything, for example by focusing on the single-pass tree prefill alone. If the tree attention mask cannot be made correct, or its overhead removes any benefit, our fallback is to narrow the scope to an *observability* tool: instrumenting an existing serving engine to detect and report repeated prefill work across concurrent multi-agent requests, without necessarily removing it. This stays in the same problem area, so work done before the PoC is not wasted. If that also proves too ambitious, we would narrow further, for example to offline analysis of recorded multi-agent prompts. This fallback was discussed with the instructor during project approval, who agreed it is a sufficient exit strategy; the exact rescoping would be decided with the TA after the PoC demonstration.

10 Expected Technology

Table 6 lists the tools we expect to use and why each was chosen.

Table 6: Expected technology

Need	**Choice and reason**
Language	Python for almost everything; C++ and CUDA (NVIDIA's GPU programming platform) only if a custom kernel turns out to be necessary.
Model framework	PyTorch and Hugging Face Transformers: the standard tools for loading open models and inspecting their internals, which the PoC needs.
Serving engine	vLLM [**?**], extended rather than rewritten. vLLM has the largest community and documentation, and its paged memory with copy-on-write is the base we build on. The cost is a large codebase to learn; SGLang [**?**] is the alternative if vLLM's design gets in the way.
Fast attention	PyTorch FlexAttention first (tree mask as a small function, compiled into a fast kernel), then splitting attention into a shared-trunk part and a per-branch part using existing kernels. Hand-written CUDA only as a last resort.
Models	Existing open-weight models; no training. Qwen2.5-0.5B for tests (fast on a laptop CPU), larger Llama/Qwen models for benchmarks.
Linter / formatter	`ruff` (lint and format), run in CI.
Unit tests	`pytest`. The core test compares tree-pass outputs against separate runs.
Code coverage	`coverage.py` through `pytest-cov`, reported in CI; no target until the codebase exists.
CI	GitHub Actions for documents (running) and code (planned).
Performance tools	PyTorch Profiler and NVIDIA Nsight Systems, on the department clusters (`ada`, 12× L40S, for development; `grace`, 4× H100, for final benchmarks).
Editor	VS Code and Cursor, both locally and on the department clusters through Remote-SSH, so the same editor is used on a laptop and on a GPU machine.
Notebooks	Jupyter notebooks for quick demos and exploration. They run locally, on Google Colab, or on a cluster through VS Code. Code that the team keeps is moved into regular Python files with tests.
Google Colab	Quick GPU checks with its free GPU, for example while a cluster is busy or not yet set up. Colab is not used for reported measurements, because its GPU type, memory, and session length vary.
Project tools	git, GitHub, GitHub Projects, Microsoft Teams.

11 Use of AI and LLMs

- **Tools:** Claude Code and OpenAI Codex for code, debugging, and document drafting; GitHub Copilot code review on pull requests; Granola for meeting notes and summaries.
- **Code:** AI-written code is held to the same standard as any other code: it is reviewed in a PR and must pass the tests. For correctness-critical code, such as the attention mask, the author must be able to explain every line in review, since a plausible-looking result is not the same as a correct one.
- **Documents:** AI may draft or restructure text, but decisions (roles, schedule, scope) are made by the team, and every claim about other systems or papers is checked against the original source before it is submitted.
- **Disclosure:** each course document has an appendix describing how AI was used for it.

12 Coding Standard

Python code follows the PEP 8 style guide, enforced by `ruff` in CI. Public functions have type hints and a docstring. Tests live in `test/` and are named after the module they test. If C++/CUDA code becomes necessary, it will follow the Google C++ Style Guide, formatted with `clang-format`.

13 Appendix — Generative AI Use Disclosure

This appendix describes how AI tools were used to write this document. Planned use of AI during the project itself is in the Use of AI and LLMs section.

13.1 AI Tools Used

- Claude, through the Claude Code command-line tool (version 2.1.283), using the Claude Sonnet 5 and Claude Opus 5.5 models for the draft and its revisions.
- GitHub Copilot pull request review, on PR #3 of the project repository.
- Gemini 8 Flash, through Google AI Studio.

13.2 How and Where Used

Table 7 lists each part of the document that AI tools contributed to, and how.

Table 7: AI use by part of the document

Part of document	Use
Background research	Gemini was used to research the problem area and find related papers on LLM serving and shared-prefix inference, which informed the project framing and the related work named in this plan.
All sections	Claude produced the first draft of every section from information and decisions given by the team, and reorganized the document to follow the course template.
Checklist and rubric	Claude compared the draft against `docs/Checklists/DevPlan-Checklist.tex` and the course rubric and listed the gaps (missing PoC fallback, technical and QA roles, schedule timing, tool justification); the team decided how to fill each one.
PoC plan	Claude helped design the correctness test and the vLLM measurement scenarios described in Section 9.
Schedule	Claude collected the course dates from the capstone repository's weekly announcements and course schedule.
License	Copilot review pointed out that `LICENSE` was still the template; the team replaced it.
Technology	Claude suggested FlexAttention and summarized related work [**?**, **?**, **?**] as candidates for the team to read, and drafted the bibliography entries.
Reflection	The team supplied its own points and answers; Claude wrote them up as prose, and the team checked that the result says what it meant.

13.3 Verification of AI-Generated Content

William Pol drafted this document with Claude Code from decisions made by the team, and Zayed Umer reviewed the pull request, requesting the PoC memory/throughput note and the tooling additions reflected in this version. The whole team reviews the document again before submission.

- **Decisions:** meeting format, communication tool, role rotation, and fallback scope were chosen by the team; AI-proposed defaults were replaced or confirmed by the team before submission.
- **Results:** any correctness or performance numbers reported in future deliverables will come from running the code in the repository, not from the AI's description of it.

- **Dates:** course dates were checked against the capstone repository's announcements.
- **Bibliography:** the entries for the cited papers were drafted by the AI from memory, so each is checked against the paper itself before submission.
- **Related work:** the papers suggested by the AI are treated as leads to read, not as facts; none of their claims are relied on in this document until a team member has read the paper.
- **Review:** the full document was reviewed by team members against the checklist before submission.

The team is responsible for the accuracy, completeness, consistency, and appropriateness of this document, whether material was written by a person or an AI system.

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. It is good for the team to review and agree on the terms of how we want to work together and develop this project, so that problems surface early instead of showing up in later deliverables. Making explicit decisions about roles, workflow, and technology, and committing to them as a team, is what prevents those problems in the first place: a decision that is only assumed rather than written down and agreed to is a decision the team can quietly disagree about later, right when there is no time left to resolve it. The PoC plan pushed this further: it required us to assess risk and think honestly about the weaknesses in our own proposal before we had written any code. That exercise directly shaped how we are developing the project, since it pushed us to write and run code early, so that if our core idea was going to fail, it would fail early, not right before a deadline.

2. CI/CD is good for making the development pipeline concrete, giving the team an adequate, repeatable framework to develop against. It catches build and documentation errors, and later test failures, before they reach `main`, and it automates tedious tasks like PDF generation. However, we found CI/CD can be both flaky and too strict at the same time. In the main capstone repository, the pipeline for our project description submission would not let us merge because we had marked "supervisor agreed" as false, even though the instructor, Spencer Smith, had told us directly it was fine to proceed without a supervisor for now. Separately, GitHub Pages deployment for our own documentation has worked inconsistently, some runs succeeding and others not, for no clear reason. So while CI/CD is a useful tool, it can become a bottleneck rather than a help if it is not kept reliable, which ties back to the setup and maintenance cost of running it at all.

3. Two disagreements came up while writing this deliverable. The first was whether to assign specific names to the newly proposed team roles right now. Some teammates argued that naming people would give the team firmer direction moving forward and make responsibilities clear. Others

disagreed, arguing that attaching labels before we had actually worked together was misguided without a proper gauge of each teammate's strengths and weaknesses. We resolved this by deferring the decision rather than forcing an answer either way. The second was about the premise of the project itself: preliminary work for the PoC revealed that one of our core assumptions was weaker than we had first imagined. We agreed the overall direction of the project was still intact, because we have strong passion for this area, but the disagreement made clear that we need to keep testing our assumptions rather than assume the original framing still holds.

Appendix — Team Charter

This charter follows the structure of the University of Portland Team Charter template.

External Goals

Build something substantial enough to be a genuine portfolio and interview talking point in systems engineering, and aim for a strong final grade; present at the Capstone EXPO if the results are compelling.

Attendance

These rules cover team meetings, TA meetings, and demonstrations.

Expectations

Team members are expected to attend all scheduled team meetings on time and stay for the full meeting; if unable to attend, notify the team as far in advance as possible.

Acceptable Excuse

Illness, other course or exam conflicts, and emergencies are acceptable excuses. Simply forgetting, or deprioritizing without advance notice, is not.

In Case of Emergency

Notify the team chat and the meeting chair as soon as possible; the chair will share Granola's meeting summary with the absent member to bring them up to speed on decisions and action items.

Accountability and Teamwork

These expectations apply to every member, whatever their current role.

Quality

Members are expected to come to meetings having completed their agreed-upon work, or to flag blockers before the meeting rather than at it. Pull requests should be self-reviewed (tests passing, documents updated) before requesting review.

Attitude

We adopt the Contributor Covenant as our code of conduct. Disagreements are expected to be resolved through discussion and, if unresolved, a team vote (see Decision Making below).

Stay on Track

We use the course's project management metrics (commits, issue activity) as a lightweight check-in signal, with a quantified trigger: if a member has no commits or issue activity and misses two consecutive weekly meetings without an acceptable excuse (see Attendance above), the team will raise it with them directly in the next meeting. If it persists for a further two weeks, the team will escalate to the TA.

Team Building

The team eats together after milestone deliverables (Problem Statement, PoC demo, Rev 0 demo, Final demo) to mark the milestone and decompress before the next one starts. Weekly meetings open with a few minutes of non-project conversation before the agenda starts, so meetings stay a place the team wants to show up to, not just a status check.

Decision Making

Default to consensus; if consensus is not reached after discussion, decide by majority vote among team members.