Development Plan
Cow-puter Vision

Team # 19, Cow-puter Vision
Changhao Wu
Zongcheng Li
Baoning Zhang
Ji Zhang

September 22, 2025

Table 1: Revision History

Date	**Developer(s)**	**Change**
2025-09-18	Zongcheng Li	Initial draft of Development Plan
2025-09-18	All Members	Check-in with TA and updates based on feedback
2025-09-20	Zongcheng Li	Finish remaining sections of the Development Plan
2025-09-21	Ji Zhang	Updated Appendix, adjusted formatting, and added a reference.
2025-09-21	Changhao Wu	Revised Team Meeting Plan section and Team Member Roles section
2025-09-22	Changhao Wu	Revised Main Risks and Mitigation Strategies section with clearer risks and more detailed mitigation steps
2025-09-22	Changhao Wu	Revised Project Decomposition and Scheduling section
2025-09-22	Changhao Wu	Revised Team Communication Plan section and Project Decomposition and Scheduling
2025-09-22	Ji Zhang	Update Coding Standards section
...	...	...

Development Plan Overview

The development plan for Cow-puter Vision outlines how our team will organize, develop, and deliver an AI-based system to identify cows by their black-white patterns on their back and detect potential health issues. This document includes our Confidential information, IP to protect, Copyright license, Team meeting plan, Team communication plan, Team member roles, Workflow plan, Project decomposition and scheduling, Proof of concept demonstration plan, Expected technology, and Coding standard.

1 Confidential Information?

Our project does not involve confidential information from industry. All data and models will be based on publicly available datasets or collected in compliance with open-source or academic research guidelines.

2 IP to Protect

There is no proprietary intellectual property (IP) to protect for this project. The outputs will be published under an open-source license to encourage collaboration and reproducibility.

NEED UPDATE LATER, since we haven't discuss the details with the industry partner yet.

3 Copyright License

The team has chosen the MIT License, which will be included in the GitHub repository. This license provides flexibility for others to use, modify, and distribute the project with proper attribution.

4 Team Meeting Plan

The team will meet every Tuesday and Sunday between 6:00–10:00 PM, with each meeting lasting approximately 30 minutes. Meetings will be held online using Microsoft Teams or Discord. If an in-person session is required for testing, integration, or major decision-making, a separate time and location will be arranged in advance. These twice-weekly meetings do not include the mandatory weekly meeting with the TA during the Thursday tutorial time, nor the separate weekly meeting with the supervisor.

Each meeting will be structured as follows:

1. The agenda for each meeting will be posted as a GitHub issue ahead of time.
2. Each team member will provide updates on their assigned tasks (progress, challenges).
3. The team will discuss ongoing issues, distribute new tasks, and set short-term goals.
4. Each meeting will have a designated chair and a note-taker. All meeting outcomes will be recorded and shared with the team.

In addition:

- All members must attend the weekly TA meeting during the Thursday tutorial time.
- The supervisor will hold a weekly virtual meeting with the team for progress updates, feedback, and Q&A.

5 Team Communication Plan

- **Discord / WeChat**: Primary platforms for internal team communication, informal discussions, quick updates, and project-related meetings.
- **Discord / MS Teams**: Used for meetings with supervisor and TA.
- **GitHub**: Code-related discussions, issue tracking, project management, and documentation.
- **Email**: Formal communication and sharing of important documents.
- **MS Teams (Shared Folder)**: Collaborative document editing and storage.

6 Team Member Roles

Every team member is responsible for attending meetings, responding to messages, creating issues, coding, testing, reviewing, and documentation. In addition, members will take on the following roles:

- **Project Manager**: Organizes the workflow, monitors progress toward deadlines, balances workload among members, and communicates project status to the supervisor.
- **Meeting Chair**: Prepares agendas, leads meetings, facilitates discussions, ensures decisions are made efficiently, and keeps meetings on track.
- **Notetaker**: Team members will take turns serving as notetaker, responsible for recording meeting minutes, capturing action items, and summarizing key discussions to be shared with the group.
- **Quality Assurance**: All team members will participate in reviewing deliverables to ensure completeness, accuracy, and adherence to project requirements. The review process will be shared collectively rather than assigned to a single individual.

Roles are expected to be flexible, and members may rotate or assume different responsibilities as project needs evolve, ensuring both accountability and adaptability.

7 Workflow Plan

7.1 Normal Workflow

If a team member is working on a new feature, bug fix, testing, or documentation, the following workflow will be adopted:

- Planning and Task Assignment
    1. Discuss any issues or tasks in team meetings or in messages.
    2. Create or update GitHub issues to track tasks, bugs, and feature requests. Move the issue to "Backlog".
    3. Use appropriate labels for classification (e.g., 'feature', 'bug', 'documentation').
    4. Prioritize tasks based on project milestones and deadlines.
    5. Assign tasks to team members based on their expertise and workload and then move the issue to "Ready".
    6. Set in-group deadlines for tasks to ensure timely completion.
- Development
    1. Create a new branch for each feature or bug fix, following the naming convention 'feature/feature-name', 'test/test-name', or 'bugfix/bug-name'.

2. Pull any changes from the main branch before starting new work.
3. Move the related issues to "In Progress".
4. Create a draft of the structure of the code/documentation.
5. Implement the functions without dependencies first, then the functions with dependencies.
6. Commit changes frequently with clear, descriptive messages.
7. Push the branch to the remote repository on GitHub.
8. Open a pull request (PR) to merge the feature/bugfix branch into the main branch, with appropriate link to the related issue and labels.
9. Assign at least one team member as a reviewer for the PR.
10. Address any feedback from the reviewer(s) and make necessary changes.
11. Once approved, the PR will be merged into the main branch.
12. Delete the feature/bugfix branch after merging to keep the repository clean.
13. Move the issue to "Done" once the task is completed and merged.

- Review

1. Check changes of the PR to ensure completeness, accuracy, and adherence to project requirements.
2. If applicable, provide constructive feedback and suggestions for improvement.
3. Approve the PR if it meets the quality standards.
4. Request changes if the PR does not meet the quality standards.
5. Merge the PR into the main branch once approved.
6. Delete the feature/bugfix branch after merging to keep the repository clean.

7.2 Small Fixes

For small fixes (e.g., typos, minor formatting changes), team members can commit directly to the main branch after pulling the latest changes and ensuring no conflicts exist. However, if the fix is part of a larger feature or bug fix, it should follow the normal workflow.

1. Pull the latest changes from the main branch.
2. Make the small fix directly in the main branch.
3. Commit the change with a clear message.
4. Push the changes to the remote repository on GitHub.
5. Send a message to the team notifying them of the change.

8 Project Decomposition and Scheduling

- Link to GitHub Project: https://github.com/InfantMob/Cow-puter-Vision
- The project will be decomposed into the following major components:
  - Requirement Analysis and Specification
  - System Design and Architecture
  - Data Collection, Preprocessing, and Augmentation
  - External Component Integration, and Adaptive Modification
  - Neural Network Model Construction and Training

  - Model Evaluation, Validation, and Improvement
  - Pipeline Integration and Deployment
  - Preparation of Final Deliverables
- Each component is further broken down into smaller tasks, which are tracked using GitHub Issues and linked to the Kanban board.
- The project is managed using GitHub Projects with a Kanban board setup. The board contains the following columns:
  - No Status / Backlog
  - Ready
  - In Progress
  - In Review
  - Done
- Each task will be assigned to a team member, with due dates set based on the overall project timeline.
- Regular updates will be made during weekly meetings to ensure progress is on track.

Deliverable	**Date**
Problem Statement, POC Plan, Development Plan	Sep. 22, 2025
Req. Doc. and Hazard Analysis Revision 0	Oct. 6, 2025
V&V Plan Revision 0	Oct. 27, 2025
Design Document Revision -1	Nov. 10, 2025
Proof of Concept Demonstration	Nov. 17-28, 2025
Term Break	—
Design Document Revision 0	Jan. 19, 2026
Revision 0 Demonstration	Feb. 2-13, 2026
V&V Report and Extras Revision 0	Mar. 9, 2026
Final Demonstration (Revision 1)	Mar. 23-29, 2026
EXPO Demonstration	TBD
Final Documentation (Revision 1)	Apr. 6, 2026

9 Proof-of-Concept Demonstration Plan

The proof-of-concept (POC) is designed to directly address the key risks to project success by demonstrating that the core functionality can be achieved under realistic constraints.

Main Risks and Mitigation Strategies

1. **Feasibility of the computer vision pipeline**: The pipeline for this task consists of multiple components (e.g., detection, tracking, and individual identification). If all components were to be implemented from scratch, the project would likely exceed the available time and result in significantly lower accuracy compared to state-of-the-art methods. Since the overall pipeline accuracy is bounded by the weakest component, unreliable submodules would drag down the entire system's performance.

   - *Mitigation*:

     Use established models as the foundation for certain components, such as YOLO for cow detection and a tracking library (e.g., ByteTrack or DeepSORT) for maintaining identities across frames. This allows the team to focus its efforts on the novel and most critical task—*individual cow identification*. This division of effort makes the project feasible while ensuring the final pipeline maintains competitive accuracy.

2. **Pipeline integration complexity**: Even if individual components (detection, tracking, and identification) work well independently, integrating them into a seamless pipeline may introduce unexpected errors, latency, or data mismatches between modules.

   - *Mitigation*:

     Use standardized data formats (e.g., consistent bounding box representations, unified frame rates) to minimize compatibility issues between modules.

     Employ logging and modular testing to monitor intermediate outputs at each stage, so problems can be traced without debugging the entire pipeline.

3. **Availability and quality of training data**: Datasets may be insufficient in size or diversity, leading to poor generalization.

   - *Mitigation*:

     Combine multiple public datasets (e.g., OpenCows2020, MultiCamCows2024). Perform data augmentation and, if necessary, collect a small supplemental dataset.

4. **Data domain mismatch (training vs. real-world deployment)**: Public datasets may differ significantly from real farm environments in terms of lighting, camera angles, and resolution. As a result, a model trained only on public datasets may underperform when deployed in the field.

   - *Mitigation*:

     Incorporate "non-ideal" data samples in the POC, such as videos captured under different lighting or viewing angles, to test robustness.

5. **Tracking robustness in crowded or occluded scenes**: When multiple cows overlap or partially occlude each other in the video feed, off-the-shelf tracking components may lose track of identities.

   - *Mitigation*:

     Evaluate multiple tracking libraries (e.g., ByteTrack, DeepSORT) during the POC to identify which performs best in crowded scenarios.

Demonstration Plan

The POC will demonstrate a minimal but complete pipeline that directly addresses the identified risks:

1. **Cow detection and tracking**: Run YOLO for cow detection and integrate a tracking library (e.g., ByteTrack or DeepSORT) to maintain identities across frames. This step verifies that off-the-shelf components can provide a reliable foundation (*addresses pipeline feasibility and tracking robustness*).

2. **Individual identification prototype**: Apply the team's custom model to assign stable IDs to detected cows. This tests whether individual recognition can function reliably given outputs from detection and tracking (*addresses feasibility and data quality*).

3. **Domain robustness check**: Use a small set of "non-ideal" video samples with varied lighting and camera angles to observe performance differences. (*addresses domain mismatch*).

4. **Integration validation**: Combine detection, tracking, and identification into a single pipeline, while logging intermediate results at each stage. This confirms that the modules can be integrated without major latency or data mismatch (*addresses integration complexity*).

5. **User-facing output**: Display results in a simple UI showing bounding boxes and correct labels across frames stably. This demonstrates the pipeline's functionality and makes evaluation of performance straightforward.

If successful, the POC will provide evidence that each major risk can be managed and that the overall project is feasible within the available timeframe.

10 Expected Technology

The team plans to use the following technologies for the development of Cow-puter Vision:

- Programming language: Python
- External Libraries: TensorFlow, OpenCV, NumPy, Flask
- Pre-trained models: YOLOv5 for object detection
- Linter tool: flake8
- Unit testing framework: pytest
- Investigation of code coverage measuring tools: coverage.py
- Plans for Continuous Integration (CI), or an explanation that CI is not being done: GitHub Actions will be used for CI/CD.
- Performance measuring tools: TensorBoard for monitoring model training.
- Tools: Git, GitHub, and GitHub Projects.

11 Coding Standard

Because we want to ensure code readability and maintainability, we will adopt the PEP 8 coding standard for Python, which emphasizes readability and consistency. Additionally, we encourage team members to write comments and provide constructive feedback during code reviews to continuously improve our coding practices.

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. Why is it important to create a development plan prior to starting the project?

   It is essential because it gives the team a clear road map. If we do not have such a development plan, there might be scope creep, unclear responsibilities, misaligned expectations occurring. With a clear plan, we are capable to break down the project into manageable tasks, set realistic deadlines, and allocate resources effectively. It not only increase efficiency, but also decreases possibility of potential conflict.

2. In your opinion, what are the advantages and disadvantages of using CI/CD?

   Advantage: Using CI/CD ensures that code is integrated and tested frequently, which reduces the risk of major bugs piling up. Automated pipelines also make deployment faster and more reliable, improving overall team productivity. Otherwise, technical debt would be heavier and heavier.

   Disadvantage: Setting up CI/CD requires a lot of initial effort, and sometimes significant infrastructure costs. If they are not properly configured, pipelines may break frequently, slowing down development instead of speeding it up. As projects grow, pipelines become more complex, thus, build times increase, this would require maintainence and optimization.

3. What disagreements did your group have in this deliverable, if any, and how did you resolve them?

   One of the main disagreements in our group was about how detailed our development plan should be. Some members wanted a high-level overview to maintain flexibility, while others preferred a very detailed, task-by-task breakdown. We resolved this by combining both approaches: we created a high-level road map but also added more details for the first sprint, leaving later stages more flexible.

Zongcheng's answer:

1. It is important to create a development plan prior to starting the project because it shows clear direction of development, helps identify potential risks and challenges early on and avoid potential conflicts.
   We had a discussion on the development plan and get some important consensus, including how the tasks will be assigned, how the communication will be, and how the workflow will be. So that everyone is on the same page and work towards the same goal. It also helps us to understand the deadlines so that nobody will miss any deliverables.

2. From my perspective, using CI/CD has several advantages/disadvantages:

   - Advantages:
     - Faster development: it is a rapid way to deliver code changes, allowing for quicker feedback and iteration.
     - Better collaboration: provides a shared platform, good for team collaboration.
     - Minimize error: since it is automated, it reduces the likelihood of manual error in repeated tasks.
   - Disadvantages:
     - Difficult to start: it is complex and time-consuming to set up initially.
     - Maintenance overhead: it needs continuous maintenance to ensure it works properly.

– Make me lazier: since it is automated, I may rely too much on it and not pay attention to the details.

Ji's answer:

1. I think the development plan is like a "roadmap" for the team. Without a clear plan, the team is prone to problems such as ambiguous goals and unclear responsibilities. By formulating a Development Plan, we can unify our understanding from the very beginning, break down the project into executable stage tasks, and clearly define each person's role and responsibilities. This not only improves team efficiency but also avoids conflicts due to misunderstandings or differences in the later stages.

2. From my perspective, using CI/CD has several advantages/disadvantages:

   - Advantages:
     CI/CD enables continuous integration of code and automatic testing, allowing issues to be detected at an early stage, giving sufficient time for remediation. At the same time, it also makes the deployment process more reliable, reduces risks brought about by human operations, and increases the iteration speed.
   - Disadvantages:
     The initial setup of CI/CD requires additional time and effort. Moreover, if the configuration is not perfect, the pipeline may frequently encounter errors, which will instead slow down the progress. Additionally, as the project scale increases, the maintenance cost of CI/CD will also rise.

Appendix — Team Charter

External Goals

Our team's external goals for this project include:

- Make something useful for the industry partner.
- Gain practical experience in AI/ML development and write on resume.
- Achieve a high grade in the course, we aim for an A.

Attendance

Expectations

We expect all team members to attend all scheduled meetings on time and stay for the entire duration.
We understand that emergencies may arise, and in such cases, team members should notify the team as soon as possible.
Missing meetings without a acceptable excuse may result in consequences as outlined in the "Stay on Track" section.

Acceptable Excuse

We accept excuses such as:

- Illness or medical emergencies
- Family emergencies
- Academic commitments (e.g., exams, presentations, course conflicts)
- Pre-approved absences (e.g., prior commitments, work obligations)

In Case of Emergency

Actions to take in case of an emergency:

- Notify the team as soon as possible via MS Teams or email or other communication channels.
- If it is a meeting with TA or industry partner, notify them as well.
- Provide a brief explanation of the emergency and expected duration of absence.
- If possible, delegate tasks to other team members to ensure continuity.
- Upon return, catch up on missed work and communicate with the team about any challenges faced during the absence.

Accountability and Teamwork

Quality

Expectations regarding quality:

- Prepare questions, updates, and relevant materials before each meeting.
- Deliverables should be completed on time, and be able to run.
- Team members should actively participate in discussions, and replyig to messages in a timely manner.

Attitude

Our expectations regarding attitude:

- Be respectful of each other's ideas and opinions, even if they differ from our own.
- Be open to constructive feedback and willing to provide the same to others.
- Notify the team in advance (at least 24 hours) if unable to meet deadlines or attend meetings, this is important.
- Keep up to date with project progress and contribute actively to discussions and tasks.
- At least check the messages on MS Teams or Discord once a day.

Stay on Track

To keep the team on track, we will set clear milestones and deadlines for each phase of the project. Announcements and reminders will be sent on MS Teams and Discord channels to ensure everyone is aware of upcoming tasks and deadlines.

Attendance, code commits, and task completion will be monitored. If a team member consistently fails to meet expectations, the team will first discuss the issue privately with the individual to understand any challenges they may be facing. If the issue persists, the team may consider meeting with the TA or instructor to seek guidance on how to best support the individual.

Team Building

Our team will build cohesion through regular check-ins, celebrating milestones, and organizing occasional social activities outside of meetings to foster camaraderie.

We encourage open communication and provide constructive feedback so that each member gains experience in teamwork.

Decision Making

When there is a disagreement, the team will first discuss each member's opinion, and then vote if necessary. The majority opinion will be adopted. If we cannot reach a consensus, we will seek advice from our supervisor or TA.

References

1. Marinache, A. *Proof of Concept and Development Plan (Lecture Slides)*. McMaster University, CAS 741. Available at: https://gitlab.cas.mcmaster.ca/courses/capstone/-/blob/main/Lectures/L02b_POCAndDevPlan/POCAndDevPlan.pdf

2. University of Portland. *Team Charter Guidelines*. Available at: https://engineering.up.edu/industry_partnerships/files/team-charter.pdf

3. Python Software Foundation. *PEP 8 - Style Guide for Python Code*. Available at: https://peps.python.org/pep-0008/

4. GitHub Docs. *About GitHub Actions CI/CD*. Available at: https://github.com/solutions/use-case/ci-cd