

Project Planning Phase

Project Planning Template (Product Backlog, Sprint Planning, Stories, Story points)

Date	19 February 2026
Team ID	LTVIP2026TMIDS77295
Project Name	Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables.

Project Planning

The Smart Sorting project was planned and executed in a **structured, phase-wise manner** to ensure timely completion, clear roles, and measurable outcomes.

1. Project Objectives and Scope

- Build an AI-based web system that classifies fruits and vegetables as **Fresh** or **Rotten** from images.
- Use **transfer learning (MobileNetV2)** with a Flask web interface to support farmers, vendors, and warehouses.
- Deliverables:
 - Cleaned and labeled dataset (Fresh/Rotten).
 - Trained and evaluated model with documented performance.
 - Flask web application with Home, About, Predict, and Contact pages.
 - Project documentation including solution canvas, architecture, test results, and burndown chart.

2. Phases of the Project

The project can be described in 5–6 main phases (similar to CRISP-DM / standard ML project lifecycles).

1. Requirement Analysis & Problem Understanding

- Study current manual inspection process, pain points, and user segments (farmers, retailers, inspectors).

- Prepare the **Solution Canvas** (customer segments, constraints, triggers, root causes, proposed solution).
- Finalize project title, objectives, and success criteria (target accuracy, response time).

2. **Data Collection & Preparation**

- Collect fresh and rotten fruit/vegetable images from public datasets and manual sources.
- Label and organize images into folders (fresh, rotten), remove invalid and noisy samples.
- Split into training and test sets and apply preprocessing (resize, normalization) and augmentation.

3. **Model Design & Training**

- Select MobileNetV2 as the base model and design the custom classification head.
- Implement training script, configure hyperparameters, and run experiments.
- Evaluate using accuracy, confusion matrix, and classification report; fine-tune if needed.

4. **Web Application Development (Flask)**

- Design frontend pages (Home, About, Predict, Contact) and navigation.
- Integrate the trained model with Flask routes for prediction.
- Implement file upload, input validation, result display, and error handling.

5. **Testing & Performance Validation**

- Functional testing of all routes and UI elements.
- Performance testing of the model on unseen test data and sample user images.
- Fix bugs, improve messages, and refine UI.

6. **Documentation & Deployment Preparation**

- Prepare project documentation (introduction, literature survey, methodology, testing, future work).
- Create diagrams (architecture, project flow, burndown chart).

- Package code and model for local deployment or future cloud deployment.
-

3. Work Breakdown and Responsibilities

The work was divided among team members to parallelize tasks and ensure ownership:

- **Sumanth** – UI design, HTML/CSS/JS, Flask routing, navigation, and UX.
- **Venkatadurgaparvathi** – Model selection (MobileNetV2), training code, prediction integration with Flask.
- **Divija** – Dataset collection, cleaning, splitting, and documentation drafting.
- **Harika** – Test case design, execution, bug reporting, deployment environment setup.

This clear division allowed multiple phases (e.g., data preparation and UI design) to progress simultaneously.

4. Timeline and Milestones

You can present your planning using weeks/sprints:

- **Week 1 – Project Initiation & Requirements**
 - Finalize topic, objectives, and tools (Python, TensorFlow, Flask).
 - Prepare initial Solution Canvas (customer problems, constraints, and proposed AI-based solution).
- **Week 2 – Data Collection & Cleaning**
 - Download/publicly collect fruit images, manually capture additional samples if needed.
 - Remove duplicates, corrupt images, and mislabelled samples.
 - Organize into dataset/all/fresh and dataset/all/rotten.
- **Week 3 – Data Preparation & Baseline Model**
 - Implement train/test split script and basic preprocessing.

- Load MobileNetV2 and build a baseline classifier head.
 - Train first model, observe initial accuracy.
 - **Week 4 – Model Improvement & Evaluation**
 - Tune hyperparameters, adjust augmentation, and optionally fine-tune MobileNetV2 layers.
 - Evaluate final model on test data and generate confusion matrix and metrics.
 - **Week 5 – Flask App Development**
 - Create templates (index.html, about.html, predict.html, result.html, contact.html).
 - Implement Flask routes and connect them with the trained model.
 - Test end-to-end prediction flow (upload → prediction → result page).
 - **Week 6 – Testing, Documentation & Presentation**
 - Perform full testing (functional, usability, performance).
 - Fix issues, polish UI, and finalize About content with metrics.
 - Complete report, slides, burndown chart, and demonstration video.
-

5. Use of Agile Ideas and Burndown Chart

The team can explain that it followed **lightweight Agile practices**:

- Work divided into **sprints** (e.g., weekly).
- A **task list** was created for each sprint (data tasks, model tasks, UI tasks, testing tasks).
- A **burndown chart** was used to visualize remaining tasks vs days, helping track whether the team was on schedule and identify if any phase was lagging.

Typical tasks included:

- Data tasks: collect images, clean labels, run split scripts.

- Model tasks: implement training, monitor loss/accuracy, tune hyperparameters.
- Web tasks: design pages, integrate model, setup routes.
- Testing tasks: test predictions with multiple images, check error handling, verify UI responsiveness.

The burndown chart (which you already have as an image) can be referenced as evidence of planning and tracking effort over the sprint.

6. Risk Management and Mitigation

During planning, the team identified possible risks and mitigation steps:

- **Risk: Insufficient or imbalanced data (too few rotten samples)**
 - Mitigation: Collect more images, use data augmentation, balance training batches.
 - **Risk: Low accuracy or overfitting**
 - Mitigation: Use transfer learning instead of training from scratch, apply regularization and early stopping, fine-tune carefully.
 - **Risk: Time constraints for web integration**
 - Mitigation: Start UI development in parallel with model training; keep architecture simple with Flask and pre-saved model.
 - **Risk: Environment/installation issues**
 - Mitigation: Use virtual environment, maintain requirements.txt, document setup clearly.
-

7. Expected Outcomes

Based on this planning, by the end of the project the team aims to deliver:

- A trained MobileNetV2-based classifier with good accuracy for Fresh vs Rotten.
- A user-friendly Flask web app where users can upload an image and immediately see classification results.
- Complete documentation and artifacts (architecture, solution canvas, charts, test reports) demonstrating a well-planned, systematically executed project.

Product Backlog, Sprint Schedule, and Estimation

1) Product Backlog & Sprint Allocation

Sprint	Functional Requirement (Epic)	User Story No	User Story / Task	Story Points	Priority	Team Members
Sprint-1	View home page & navigation	USN-1	As a visitor, I can open the home page to know what Smart Sorting does.	2	High	Eluri Sumanth
Sprint-1	View home page & navigation	USN-2	As a visitor, I can click navbar links (Home, About, Predict, Contact).	1	High	Eluri Sumanth
Sprint-1	About page – overview	USN-3	As a visitor, I can read how Smart Sorting uses AI to detect rotten produce.	2	High	Eluri Sumanth, V.D. Parvathi
Sprint-1	About page – statistics	USN-4	As a visitor, I can see model accuracy, number of classes, and dataset size.	2	High	Eluri Sumanth, V.D. Parvathi
Sprint-1	Predict page – open form	USN-5	As a user, I can open the Predict page to upload an image.	2	High	Eluri Sumanth

Total Story Points – Sprint-1 = 2 + 1 + 2 + 2 + 2 = 9

Sprint	Functional Requirement (Epic)	User Story No	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Image upload	USN-6	As a user, I can upload a fruit/vegetable image for analysis.	3	High	Eluri Sumanth, Eella Harika
Sprint-2	Show classification result	USN-7	As a user, I can view whether the item is Fresh or Rotten with a confidence score.	3	High	V.D. Parvathi, Eluri Sumanth
Sprint-2	Handle missing/invalid file	USN-8	As a user, I get a clear message if I submit without selecting an image.	2	Medium	Eella Harika
Sprint-2	Serve uploaded images	USN-9	As the system, I must serve uploaded images so they can be shown on the result page.	2	High	V.D. Parvathi
Sprint-2	Contact / team details page	USN-10	As a visitor, I can open the Contact page to see how to reach the project team.	2	Medium	Divija Durga

Sprint	Functional Requirement (Epic)	User Story No	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Load model & update metrics	USN-11, USN-12	As a developer, I can load/retrain the model and update metrics in About.	4	High	V.D. Parvathi, Divija Durga

Total Story Points – Sprint-2 = 3 + 3 + 2 + 2 + 2 + 4 = 16

2) Project Tracker & Velocity:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed	Sprint Release Date
Sprint-1	9	6 days	08 Feb 2026	13 Feb 2026	9	13 Feb 2026

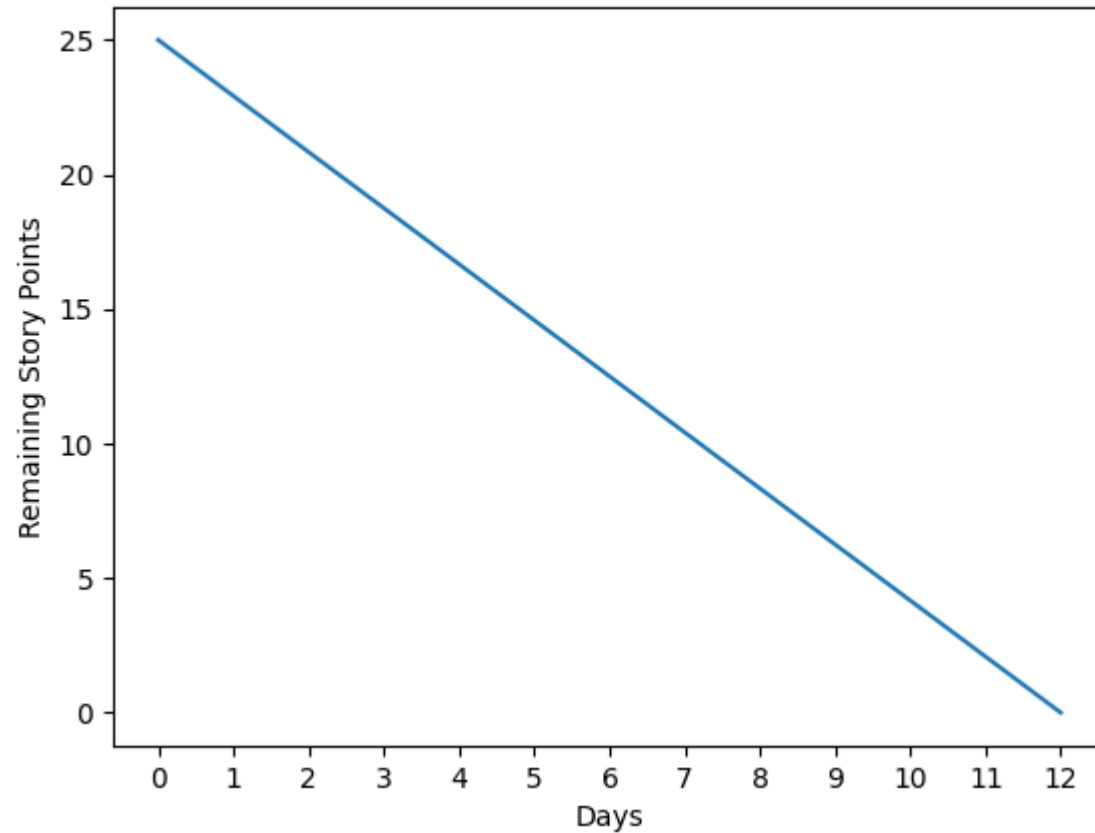
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed	Sprint Release Date
Sprint-2	16	6 days	14 Feb 2026	19 Feb 2026	16	19 Feb 2026

- Total Story Points = 9 + 16 = 25
- Number of Sprints = 2
- Velocity = $25 \div 2 = 12.5 \approx 13$ story points per sprint

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

Sprint Burndown Chart for Smart Sorting (12-Day Project, 25 Story Points)



The burndown chart represents the progress of the Smart Sorting project across two sprints (total 12 days) with a total of 25 story points.

- **X-axis:** Days (0–12)
- **Y-axis:** Remaining Story Points (0–25)
- **Initial backlog:** 25 story points

- **Final remaining points:** 0 story points
- **Velocity:** ~13 story points per sprint

The straight line shows the **ideal burndown**, assuming consistent completion of work across both sprints. Since Sprint 1 completed 9 story points and Sprint 2 completed 16 story points, the total 25 story points were successfully delivered within the planned duration.