Project Aura: 6-Day Sprint Execution Plan

November 3-8, 2025 (Mon-Sat)

Team Structure (4 People)

- **P1 - Backend & Data Lead**: Data pipeline, Pandas ingestion, Great Expectations validation, analytical functions
- **P2 - AI & Agent Lead**: Scikit-learn learning loop, LangChain agent, tool orchestration
- **P3 - Frontend & UI Lead**: Streamlit application, visualizations (Plotly), dashboard, chat interface
- **P4 - Project Manager & Integrator**: Git management, integration, testing, demo prep, pitch deck

Technology Stack (Validated & Unified)

Layer	Tool	Justification
Data Ingestion	Pandas	Industry standard for data manipulation in Python
Validation	Great Expectations (GX)	"Unit tests for data" - ensures trial balance = nil
Learning Loop	Scikit-learn	Simple classification for GL account learning
Agentic Framework	LangChain	Orchestrates AI agent with tools for reporting assistant
UI/Dashboard	Streamlit	Fastest Python-based interactive dashboard
Visualization	Plotly	Interactive charts for drill-down capability

State-of-the-Art Enhancements & Quality Bar

To win decisively, we'll layer a high-intensity track on top of the base plan. These enhancements focus on performance, reliability, explainability, and a truly autonomous agent experience.

- CI/CD: GitHub Actions (lint, type-check, tests, coverage gate ≥ 80%)
- Tooling: pre-commit (ruff, black, isort), mypy (strict), bandit (security), license check
- Packaging: pyproject.toml, modular src layout, typed APIs, docstrings
- Docs: Architecture diagram (Mermaid), ADRs, API contracts, runbooks
- Great Expectations: custom expectations, data docs site, validation store, checkpoints
- Golden datasets and snapshot tests; property-based tests for edge cases
- Vectorized Pandas; caching; lazy loading; profiling (perf + memory)
- Large-sample test (≥ 250k rows) and latency budget (p95 < 800ms per core op)
- Feature engineering; model selection (Tree + Gradient Boosting)
- MLflow experiments; reproducible seeds; metrics dashboard
- Continual learning with safety rails and rollback on regressions

- LangChain tools with Pydantic structured outputs; function calling
- RAG over project data (FAQ, accounting rules) with Chroma/FAISS
- Guardrails for tool invocation and self-healing retries; proactive insights
- Role-based demo personas; audit log; activity timeline
- Cross-filtered visuals, drill-downs, keyboard shortcuts, dark theme
- Error boundaries, user-friendly failures, offline fallback (no-network demo)

Daily Intensification Track (Adds on top of base plan)

These are additive to the existing Day 1–6 plan. Each block assumes ~12–14 hours/person/day across 4 people.

Day 1 SOTA add-on (Phase 1 + Phase 0 Env & CI)

- Initialize pyproject.toml; enable ruff, black, isort, mypy (strict), bandit in pre-commit
- Create GitHub Actions workflow: lint/type/test with coverage ≥ 80% gate
- Contribute CODEOWNERS, PR template, issue templates, conventional commits guide
- Add docs: ADR-001 (Unified Python Stack), ADR-002 (Agent w/ tools), Architecture.md stub

Day 2 SOTA add-on (Phase 2)

- Great Expectations advanced: custom expectations, checkpoints, data docs site
- Validation store (filesystem) + data docs hosted locally; snapshot tests for golden data
- SLA deviation calculations with parameterized rules and badges
- Build review status model with user/department aggregations and exportable CSVs

Day 3 SOTA add-on (Phase 3 + 4)

- Feature engineering; baseline model + Gradient Boosting alternative; MLflow experiments
- Continual learning pipeline with rollback on regression (tracked by AUC/F1)
- Visualizations: cross-filtering, drill-down pages, cached aggregations; performance profiling

Day 4 SOTA add-on (Phase 4)

- LangChain: Pydantic structured tools and function-calling; deterministic tool routing
- RAG over accounting FAQ and project docs (Chroma/FAISS vector store)
- Guardrails: safe tool parameters, retries with backoff, circuit breaker for failing tools

Day 5 SOTA add-on (Phase 4 Polish + Phase 5)

- Observability: request tracing, structured logs, timing, error taxonomy
- Security checks (bandit), dependency audit, license check; threat-model readme
- Load test with medium/large CSVs; measure latency and memory; tune caching & IO

Day 6 SOTA add-on (Phase 5)

- Demo resilience: offline runbook, backup datasets, pre-rendered charts fallback
- Executive-ready pitch deck: before/after metrics, benchmarking vs manual baseline
- Full rehearsal with risk table, Q&A bank, and live failure recovery scenario

Input Files (Critical)

- `trial_balance.csv`: Main financial data (GL Account, Debit, Credit, Description, Entity, Period)
- `responsibility_matrix.csv`: User assignments (GL Account, Responsible Person, Department, Email)

Day-by-Day Execution Plan

Day 1: Monday, November 3 - Foundation & Data Pipeline (Phase 1)

Morning (9 AM - 1 PM): Setup & Schema Design

P4 - Project Manager (Lead)

- ☐ Create Git repository structure:

```
/data/raw/          # Sample CSV files
/data/processed/    # Cleaned data
/src/               # Source code
/notebooks/         # Jupyter notebooks for exploration
/tests/             # Test files
/docs/              # Documentation
```

- ☐ Setup Python virtual environment
- ☐ Create `requirements.txt` with initial dependencies:

```
pandas==2.1.0
great-expectations==0.18.0
scikit-learn==1.3.0
langchain==0.1.0
streamlit==1.28.0
plotly==5.17.0
openai==1.0.0
python-dotenv==1.0.0
```

- ☐ Initialize Git with `.gitignore` (Python standard)

P1 - Backend & Data Lead (Lead)

- ☐ Define and document data schemas:

  trial_balance.csv:

```
GL_Account (string): GL account number
GL_Description (string): Account description
Entity (string): Company/entity name
Period (string): YYYY-MM format
Debit (float): Debit amount
Credit (float): Credit amount
```

responsibility_matrix.csv:

```
GL_Account (string): GL account number
Responsible_Person (string): User name
Department (string): Department name
Email (string): User email
```

P2 - AI & Agent Lead (Support)

- ☐ Research LangChain documentation
- ☐ Design tool interface specifications for agent:
  - `load_data_tool`
  - `validate_data_tool`
  - `generate_report_tool`
  - `variance_analysis_tool`

P3 - Frontend & UI Lead (Support)

- ☐ Streamlit project structure planning
- ☐ UI/UX wireframe for main dashboard
- ☐ Design navigation flow (file upload → validation → reports → chat)

Afternoon (2 PM - 6 PM): Core Data Pipeline

P1 - Backend & Data Lead (Lead)

- ☐ **Create `src/data_ingestion.py`**:

```python
def load_trial_balance(file_path):
    """Load and validate trial balance CSV"""
    # Read CSV with Pandas
    # Basic data type validation
    # Return DataFrame

def load_responsibility_matrix(file_path):
    """Load responsibility matrix CSV"""
    # Read CSV
    # Return DataFrame
```

```
def consolidate_data(trial_df, resp_df):
    """Join trial balance with responsibility matrix on GL_Account"""
    # Merge dataframes
    # Handle missing values
    # Return consolidated DataFrame
```

- [ ] Create sample data files in `/data/raw/`:
  - `trial_balance_sample.csv` (20-30 rows)
  - `responsibility_matrix_sample.csv` (15-20 rows)
- [ ] Test ingestion functions with sample data

P2 - AI & Agent Lead

- [ ] Setup LangChain project structure
- [ ] Create `src/agent_tools.py` (skeleton with function signatures)
- [ ] Document tool input/output specifications

P3 - Frontend & UI Lead

- [ ] **Create basic Streamlit app** `app.py`:

```
import streamlit as st

st.title("Project Aura - Automated Balance Sheet Assurance")

# File upload section
st.header("1. Data Upload")
trial_file = st.file_uploader("Upload Trial Balance", type=['csv'])
resp_file = st.file_uploader("Upload Responsibility Matrix", type=
['csv'])
```

- [ ] Test app runs locally: `streamlit run app.py`

P4 - Project Manager

- [ ] Setup project documentation in `/docs/README.md`
- [ ] Create daily standup template
- [ ] Track Day 1 progress
- [ ] Ensure all team members can push to Git

End of Day 1 Deliverable

☑ Git repository with proper structure
☑ Data schemas defined and documented

- [x] Sample CSV files created
- [x] Basic data ingestion script functional
- [x] Streamlit app runs with file upload UI

Day 2: Tuesday, November 4 - Validation & Initial UI (Phase 2)

Morning (9 AM - 1 PM): Great Expectations Integration

P1 - Backend & Data Lead (Lead)

- [ ] **Initialize Great Expectations**:

```
great_expectations init
```

- [ ] **Create Expectation Suite** in `src/data_validation.py`:

```python
def create_trial_balance_expectations(context):
    """Create GX expectation suite for trial balance"""
    suite = context.add_expectation_suite("trial_balance_suite")

    # Critical: Total debits must equal total credits
    suite.expect_column_sum_to_equal("Debit", "Credit")

    # No null values in critical columns
    suite.expect_column_values_to_not_be_null("GL_Account")
    suite.expect_column_values_to_not_be_null("Entity")

    # Valid data types
    suite.expect_column_values_to_be_of_type("Debit", "float")
    suite.expect_column_values_to_be_of_type("Credit", "float")

def run_validation(df):
    """Run GX validation checkpoint"""
    # Execute checkpoint
    # Return Pass/Fail with details
```

- [ ] Test validation with sample data
- [ ] Create validation result parser

P2 - AI & Agent Lead

- [ ] Design classification model for GL account categorization
- [ ] Create `src/ml_model.py`:

```python
from sklearn.tree import DecisionTreeClassifier

def train_gl_classifier(X, y):
    """Train initial model on GL account patterns"""
    model = DecisionTreeClassifier()
    model.fit(X, y)
    return model

def save_model(model, path):
    """Save model using joblib"""

def load_model(path):
    """Load trained model"""
```

- [ ] Create sample training data for initial model

P3 - Frontend & UI Lead (Lead)

- [ ] **Enhance Streamlit app**:
  - [ ] Display uploaded data in tables (st.dataframe)
  - [ ] Add consolidated data view
  - [ ] Create placeholder for validation results
- [ ] Design layout with st.columns for better UX

P4 - Project Manager

- [ ] Code review of Day 1 deliverables
- [ ] Setup testing framework (pytest)
- [ ] Create integration checklist

Afternoon (2 PM - 6 PM): Validation Integration & Reporting Functions

P1 - Backend & Data Lead (Lead)

- [ ] **Create** `src/analytics.py`:

```python
def calculate_variance(current_period, previous_period):
    """Calculate period-over-period variance"""
    # Group by GL Account
    # Calculate % change
    # Return variance DataFrame

def get_review_status(consolidated_df):
    """Get status of GL accounts review"""
    # Count reviewed vs pending
    # Group by department/person
    # Return status summary
```

```
def identify_anomalies(df, threshold=0.5):
    """Identify GL accounts with major variances"""
    # Flag accounts with >50% variance
    # Return list of anomalies
```

- [ ] Test each function independently
- [ ] Create unit tests

P2 - AI & Agent Lead

- [ ] Create initial model training pipeline
- [ ] Test model save/load functionality
- [ ] Document model retraining process

P3 - Frontend & UI Lead (Lead)

- [ ] **Integrate validation display**:

```
if st.button("Validate Data"):
    validation_result = run_validation(consolidated_df)

    if validation_result['success']:
        st.success("✅ Validation Passed")
    else:
        st.error("❌ Validation Failed")
        st.write(validation_result['details'])
```

- [ ] Add data quality metrics display
- [ ] Create status indicators (Pass/Fail badges)

P4 - Project Manager

- [ ] Integration testing: Data pipeline → Validation
- [ ] Document API contracts between modules
- [ ] Update progress tracker

End of Day 2 Deliverable

✅ Great Expectations integrated and working
✅ Trial balance validation (Debit = Credit) functional
✅ Streamlit app displays data and validation results
✅ Basic analytical functions (variance, status) created
✅ Initial ML model structure in place

Day 3: Wednesday, November 5 - Visualizations & Learning Loop (Phase 3 + Phase 4)

Morning (9 AM - 1 PM): Interactive Dashboards

P3 - Frontend & UI Lead (Lead)

- [ ] **Create visualization module** `src/visualizations.py`:

```
import plotly.express as px
import plotly.graph_objects as go

def create_variance_chart(variance_df):
    """Bar chart showing GL variances"""
    fig = px.bar(variance_df, x='GL_Account', y='Variance_Pct',
                 title='Major GL Variances vs Previous Period')
    return fig

def create_review_status_pie(status_df):
    """Pie chart of review status"""
    fig = px.pie(status_df, values='Count', names='Status',
                 title='GL Account Review Status')
    return fig

def create_hygiene_indicator(metrics):
    """Gauge chart for overall hygiene score"""
    fig = go.Figure(go.Indicator(
        mode = "gauge+number",
        value = metrics['hygiene_score'],
        title = {'text': "GL Hygiene Score"}))
    return fig
```

- [ ] Test each chart with sample data
- [ ] Add interactivity (hover, click)

P1 - Backend & Data Lead

- [ ] Create helper functions for chart data preparation
- [ ] Optimize data aggregations for performance
- [ ] Add caching for frequently accessed data

P2 - AI & Agent Lead (Lead)

- [ ] **Implement user feedback mechanism** in `src/feedback_handler.py`:

```
def collect_user_correction(gl_account, old_category, new_category):
    """Store user correction for retraining"""
    # Append to feedback log
    # Return confirmation

def prepare_retraining_data(feedback_log, original_data):
```

```
    """Merge feedback with original data"""
    # Combine datasets
    # Return augmented training data
```

P4 - Project Manager

- ☐ Review visualization designs
- ☐ Test data flow: backend → visualizations
- ☐ Document chart specifications

Afternoon (2 PM - 6 PM): Learning Loop & Dashboard Integration

P2 - AI & Agent Lead (Lead)

- ☐ **Complete retraining pipeline**:

```
def retrain_model(feedback_data):
    """Retrain model with user corrections"""
    # Load existing model
    # Prepare augmented dataset
    # Retrain model
    # Save updated model
    # Return metrics (accuracy improvement)
```

- ☐ Create UI for user correction:
  - Display current classification
  - Allow user to correct
  - Trigger retraining
- ☐ Test end-to-end learning loop

P3 - Frontend & UI Lead (Lead)

- ☐ **Integrate charts into Streamlit**:

```
st.header("2. Analytical Dashboard")

col1, col2 = st.columns(2)
with col1:
    st.plotly_chart(variance_chart)
with col2:
    st.plotly_chart(review_status_chart)

st.plotly_chart(hygiene_indicator)
```

- ☐ Add drill-down functionality (click on chart → show details)

- [ ] Implement tabs for different report sections

P1 - Backend & Data Lead

- [ ] Refactor backend functions as LangChain Tools
- [ ] Create `src/langchain_tools.py`:

```
from langchain.tools import Tool

data_loading_tool = Tool(
    name="Load Financial Data",
    func=load_and_consolidate,
    description="Loads trial balance and responsibility matrix"
)

validation_tool = Tool(
    name="Validate Data Quality",
    func=run_validation,
    description="Runs GX validation checks on data"
)

variance_tool = Tool(
    name="Analyze Variances",
    func=calculate_variance,
    description="Calculates period-over-period variances"
)
```

P4 - Project Manager

- [ ] Integration testing: All components together
- [ ] Create test scenarios and data
- [ ] Bug tracking setup

End of Day 3 Deliverable

- [x] Interactive Plotly charts (bar, pie, gauge) working
- [x] Charts integrated into Streamlit dashboard
- [x] User correction UI functional
- [x] Model retraining pipeline complete
- [x] Backend functions wrapped as LangChain Tools
- [x] Demonstrable learning loop

Day 4: Thursday, November 6 - AI Agent & Conversational Interface (Phase 4)

Morning (9 AM - 1 PM): LangChain Agent Setup

P2 - AI & Agent Lead (Lead)

- [ ] **Create LangChain agent** in `src/agent.py`:

```
from langchain.agents import initialize_agent, AgentType
from langchain.llms import OpenAI

def create_aura_agent(tools):
    """Initialize Aura AI Agent"""
    llm = OpenAI(temperature=0)

    agent = initialize_agent(
        tools=tools,
        llm=llm,
        agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
        verbose=True
    )
    return agent

def run_agent_task(agent, task_description):
    """Execute agent task"""
    result = agent.run(task_description)
    return result
```

- [ ] Test agent with simple tasks:
  - "Load the financial data"
  - "Validate data quality"
  - "Calculate variances"

P1 - Backend & Data Lead

- [ ] Create orchestration logic for multi-step analysis
- [ ] Add error handling for tool execution
- [ ] Create logging mechanism for agent actions

P3 - Frontend & UI Lead (Lead)

- [ ] **Design conversational interface**:

```
st.header("3. AI Assistant Chat")

# Chat history
if 'messages' not in st.session_state:
    st.session_state.messages = []

# Display chat history
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
```

```
            st.write(message["content"])

    # Chat input
    if prompt := st.chat_input("Ask Aura anything about your financials..."):
        st.session_state.messages.append({"role": "user", "content": prompt})
        # Will connect to agent in afternoon
```

P4 - Project Manager

- ☐ Review agent architecture
- ☐ Create demo script outline
- ☐ Test user journey flow

Afternoon (2 PM - 6 PM): Agent Integration & Proactive Insights

P2 - AI & Agent Lead (Lead)

- ☐ **Expand agent capabilities** - handle specific queries:

```
def handle_query(agent, query):
    """Route user query to appropriate tool"""
    # Parse query intent
    # Execute agent with context
    # Format response
    return response
```

- ☐ Pre-define common query patterns:
  - "What are the major variances?"
  - "Show me GL accounts pending review"
  - "Which accounts exceed threshold?"

P3 - Frontend & UI Lead (Lead)

- ☐ **Connect chat to agent**:

```
if prompt := st.chat_input("Ask Aura..."):
    # Add user message
    st.session_state.messages.append({"role": "user", "content": prompt})

    # Get agent response
    with st.spinner("Aura is thinking..."):
        response = handle_query(agent, prompt)

    # Add agent response
    st.session_state.messages.append({"role": "assistant", "content":
response})
    st.rerun()
```

- [ ] Add suggested questions (quick actions)
- [ ] Format agent responses nicely (tables, charts in chat)

P1 - Backend & Data Lead (Lead)

- [ ] **Create proactive insights engine** in `src/insights.py`:

```
def generate_proactive_insights(df):
    """Generate insights automatically"""
    insights = []

    # Example: Expense spike detection
    expense_variance = calculate_expense_variance(df)
    if expense_variance > 2.0:
        insights.append(f"⚠️ Expenses for current period are
{expense_variance}x the previous period")

    # Example: Review bottlenecks
    pending_reviews = get_pending_reviews(df)
    if len(pending_reviews) > 10:
        insights.append(f"📋 {len(pending_reviews)} GL accounts pending
review")

    return insights
```

- [ ] Display insights prominently on dashboard

P4 - Project Manager

- [ ] End-to-end testing of conversational flow
- [ ] Document query examples for demo
- [ ] Create troubleshooting guide

End of Day 4 Deliverable

- [x] LangChain agent operational
- [x] Agent can execute all backend tools
- [x] Conversational chat interface working
- [x] Agent responds to natural language queries
- [x] Proactive insights displayed on dashboard
- [x] Full agentic behavior demonstrated

Day 5: Friday, November 7 - Polish, Testing & Integration (Phase 4 Polish + Phase 5)

Morning (9 AM - 1 PM): UI/UX Polish

P3 - Frontend & UI Lead (Lead)

- ☐ **Enhance dashboard aesthetics**:
  - ☐ Add custom CSS for professional look
  - ☐ Implement Streamlit themes/colors
  - ☐ Add Adani Group branding (if applicable)
  - ☐ Loading spinners for all async operations
  - ☐ Progress bars for multi-step processes
- ☐ **Improve navigation**:
  - ☐ Use st.sidebar for navigation
  - ☐ Add page sections with st.expander
  - ☐ Organize workflow: Upload → Validate → Analyze → Chat
- ☐ **Error handling in UI**:
  - ☐ Graceful error messages
  - ☐ Input validation feedback
  - ☐ Help tooltips

P1 - Backend & Data Lead

- ☐ Code optimization and refactoring
- ☐ Add comprehensive error handling
- ☐ Performance testing with larger datasets
- ☐ Add data caching where appropriate

P2 - AI & Agent Lead

- ☐ Fine-tune agent prompts for better responses
- ☐ Add more query examples
- ☐ Optimize model inference speed
- ☐ Test edge cases

P4 - Project Manager (Lead)

- ☐ Create comprehensive test plan
- ☐ Prepare multiple test datasets (small, medium, large)
- ☐ Document known issues and limitations

Afternoon (2 PM - 6 PM): Integration Testing & Bug Fixing

P4 - Project Manager (Lead)

- ☐ **Execute full integration tests**:
  - ☐ Test 1: Upload → Consolidate → Display
  - ☐ Test 2: Validation Pass scenario
  - ☐ Test 3: Validation Fail scenario

    - ☐ Test 4: Variance analysis
    - ☐ Test 5: User correction → Retraining
    - ☐ Test 6: Agent query responses
    - ☐ Test 7: Proactive insights generation
- ☐ Create bug list with priorities (P0, P1, P2)
- ☐ Assign bugs to team members

All Team Members

- ☐ Bug fixing sprint (2-6 PM)
- ☐ Focus on P0 (critical) and P1 (high) bugs
- ☐ Cross-review each other's fixes
- ☐ Re-test after fixes

P3 - Frontend & UI Lead

- ☐ Final UI polish based on feedback
- ☐ Add sample data download button
- ☐ Create onboarding/welcome message

P1 & P2

- ☐ Final code cleanup
- ☐ Add inline comments
- ☐ Update docstrings

End of Day 5 Deliverable

☑ Polished, professional UI
☑ All critical bugs fixed
☑ Stable end-to-end workflow
☑ Comprehensive testing completed
☑ Application ready for demo
☑ Code clean and documented

Day 6: Saturday, November 8 - Demo Prep & Pitch (Phase 5)

Morning (9 AM - 1 PM): Demo Preparation

P4 - Project Manager (Lead)

- ☐ **Create pitch deck** (PowerPoint/Google Slides):

  Slide Structure:

  1. **Title**: Project Aura - Automated Balance Sheet Assurance
  2. **Problem Statement**: Current pain points (manual process, time-consuming)

3. **Solution Overview**: AI-powered autonomous review agent
4. **Technology Stack**: Clean diagram showing architecture
5. **Core Capabilities** (6 capabilities with screenshots):
    - Data Ingestion & Consolidation
    - Automated Report Generation
    - Validation & Compliance Checks
    - Continuous Learning Loop
    - Agentic Behaviour
    - Interactive Visualization
6. **Live Demo Flow**: Walkthrough of key features
7. **Learning Demonstration**: Before/after model accuracy
8. **Impact & Benefits**: Time saved, accuracy improved
9. **Scalability & Future Roadmap**
10. **Team & Thank You**

P1 - Backend & Data Lead

- ☐ Create compelling demo data:
    - ☐ Realistic company names (Adani Ports, Adani Power, etc.)
    - ☐ GL accounts with clear variances
    - ☐ Some intentional data quality issues to show validation
    - ☐ Period-over-period data for variance analysis

P2 - AI & Agent Lead

- ☐ Prepare learning loop demonstration:
    - ☐ Initial model accuracy metrics
    - ☐ Apply user corrections
    - ☐ Show improved accuracy
    - ☐ Document improvement percentage

P3 - Frontend & UI Lead

- ☐ Create demo walkthrough script
- ☐ Prepare sample queries for chatbot
- ☐ Test demo flow multiple times
- ☐ Ensure smooth transitions

Afternoon (2 PM - 6 PM): Rehearsals & Final Polish

Full Team - Demo Rehearsals (3-4 run-throughs)

Demo Script (8-10 minutes):

1. **Introduction (1 min)** - P4
    - Problem context
    - Solution overview

2. **Data Upload & Consolidation (1.5 min)** - P3
    - Upload trial_balance.csv
    - Upload responsibility_matrix.csv
    - Show consolidated view with responsibilities assigned
3. **Validation & Compliance (1.5 min)** - P1
    - Click "Validate Data"
    - Show PASS scenario (debits = credits)
    - Briefly show FAIL scenario (pre-recorded or second dataset)
4. **Automated Reports & Visualizations (2 min)** - P3
    - Show variance bar chart (major GL accounts)
    - Show review status pie chart
    - Show GL hygiene indicator
    - Demonstrate drill-down (click on chart element)
5. **Continuous Learning Loop (1.5 min)** - P2
    - Show current GL classification
    - User corrects a misclassified item
    - Click "Retrain Model"
    - Show improved accuracy metrics
    - "Evidence of learning from user corrections" ✓
6. **Agentic Behaviour & Proactive Insights (1.5 min)** - P2/P3
    - Show proactive insight: "Expenses are 2.5x previous period"
    - Ask agent via chat: "What are the major variances?"
    - Agent responds with analysis
    - Ask: "Show me GL accounts pending review"
    - Agent provides list
7. **Closing & Impact (1 min)** - P4
    - Recap key capabilities delivered
    - Impact metrics (time saved, accuracy)
    - Future roadmap
    - Thank you

Tasks for Afternoon

- ☐ **P4**: Finalize and practice narration for slides
- ☐ **P3**: Ensure demo app runs flawlessly, backup plan if internet/system issues
- ☐ **P1**: Prepare technical Q&A responses
- ☐ **P2**: Prepare ML/AI specific Q&A responses
- ☐ **All**: Practice handoffs between team members
- ☐ **All**: Rehearse timing (must fit in time limit)

- ☐ **All**: Prepare for judges' questions:
  - How does this scale to 1000+ entities?
  - What about data security?
  - Integration with SAP systems?
  - Model accuracy and improvement metrics?
  - Technology choices rationale?

Evening (6 PM - 8 PM): Final Checks

- ☐ Final code commit to Git
- ☐ Create README.md with setup instructions
- ☐ Backup demo data and application
- ☐ Charge all laptops
- ☐ Test on presentation laptop/system
- ☐ Prepare backup USB drive with code + demo
- ☐ Print pitch deck (if required)
- ☐ Team pep talk and rest!

End of Day 6 Deliverable

☑ Polished pitch deck ready
☑ Demo rehearsed and timed perfectly
☑ Compelling demo data prepared
☑ Learning loop demonstration ready
☑ Team confident and prepared
☑ All Q&A scenarios covered
☑ **Ready to win!** 🏆

Daily Standup Template

Time: 9:00 AM Daily
Duration: 15 minutes

Each person answers:

1. What did I complete yesterday?
2. What will I do today?
3. Any blockers or dependencies?

P4 tracks progress and removes blockers.

Success Criteria Checklist

Problem Statement Requirements ✓

- ☑ **Data Ingestion & Consolidation**: CSV upload, merge on GL_Account
- ☑ **Automated Report Generation**: Variance charts, status tracking, visual summaries

- ☑ **Validation & Compliance Checks**: Trial balance = nil (GX)
- ☑ **Continuous Learning Loop**: User correction → retrain model
- ☑ **Agentic Behaviour**: Proactive insights, autonomous analysis
- ☑ **Interactive Visualization**: Dashboard with drill-down, conversational interface

Expected Hackathon Output ✓

- ☑ Upload sample datasets (CSV)
- ☑ AI Agent ingests, consolidates, generates reports
- ☑ Dashboard presents key numbers, charts, anomalies
- ☑ Evidence of learning from user corrections

Risk Mitigation

Risk	Mitigation
LangChain API issues	Have fallback: hardcoded responses for demo
Model training slow	Pre-train model, show cached results
Streamlit crashes	Regular commits, multiple backups
Internet connectivity	Run everything locally, offline mode
Time overruns	Daily time-boxing, cut scope if needed
Integration bugs	Daily integration testing, not just final day

---

Tech Setup Checklist (Before Day 1)

- ☐ Python 3.9+ installed
- ☐ Git installed and configured
- ☐ VS Code or PyCharm ready
- ☐ OpenAI API key (if using GPT models) or use local alternatives
- ☐ All team members have GitHub access
- ☐ Virtual environment setup knowledge
- ☐ Basic familiarity with each tool in tech stack

Notes

- **Strategic Focus**: We're building a **prototype** not a production system
- **Demo-Driven Development**: Everything must be demonstrable
- **Prioritize Core Features**: All 6 capabilities must be shown, even if simplified
- **Code Quality**: Clean, commented code for judges' review
- **Story > Tech**: Explain the business impact, not just technical implementation

---

Let's build something amazing! 🚀