# Phase 1 Summary: Simulation Core & Logging (SpaceForgeOS-xAI)

In **Phase 1**, we successfully designed and implemented the foundational simulation framework for the **SpaceForgeOS-xAI** project. This phase focused on building and validating the core power system, subsystem interaction, and telemetry logging infrastructure.

## Core Deliverables

- **Subsystem Base Class** (`Subsystem.hpp`): Defines the interface for all modules with `initialize()`, `tick()`, and `shutdown()` lifecycle methods.
- **SolarArray** (`SolarArray.cpp/.hpp`): Generates power each tick based on an orbital cosine model.
- **Battery** (`Battery.cpp/.hpp`): Stores power with a capped draw rate; tracks state of charge and issues low-charge warnings.
- **PowerBus** (`PowerBus.cpp/.hpp`): Distributes generated power; tracks available energy per tick.
- **Simulation Engine** (`SimulationEngine.cpp/.hpp`): Centralized tick manager that runs subsystem updates and advances time.
- **Telemetry Logger** (`TelemetryLogger.cpp/.hpp`): Records simulation state (battery, solar, bus) to CSV for ML.
- **TickContext Struct**: Unified tick metadata passed to each subsystem (`tick index`, `time`, `dt`).
- **Main Driver** (`main.cpp`): Connects all components and runs the simulation loop.

## Logging Output

- **CSV Format**: `tick, time, battery_charge, solar_output, powerbus_available`
- Solar output decays with time (cosine orbital model).

- Battery charges up by a fixed rate (5 Wh/tick).
- PowerBus flushes unused power at the end of each tick.
- Data is now ML-ready for future GNN pipeline stages.

# Introducing TickPhaseEngine (Phase 1.5 Planning)

As simulation complexity increases, tick **ordering** becomes critical. We are introducing a lightweight `TickPhaseEngine` architecture in preparation for Phase 2 and beyond.

### Why Use TickPhaseEngine?

- Prevent errors due to incorrect tick order.
- Add new subsystems without modifying tick loops.
- Cleanly group modules into simulation **phases** (generation, thermal, logging, etc.).

### Phase-Based Ticking Structure

```
enum class TickPhase {
    Generation,
    Consumption,
    Balancing,
    Thermal,
    Logging
};
```

Each subsystem declares its tick phase. The engine sorts and updates them accordingly.

---

# Future Phases Roadmap

| Phase | Focus | Key Deliverables |
|---|---|---|
| **2** | Thermal Modeling | `HeaterBank`, `WakeChamber`, coupling with power system |
| **3** | Vacuum Physics | `SPARTA`, `Molflow+`, `WakeVacLib`, Monte Carlo runs |

| | | |
|---|---|---|
| **4** | Graph Output | Tick-based GraphML/JSON logs for GNN input |
| **5** | ST-GNN + XAI | GNN training with PyTorch Geometric, Integrated Gradients (Captum) |
| **6** | ML Inference | LibTorch integration, prediction-driven tick loop logic |
| **7** | Scheduling | ML-based decision making, Optuna tuning, feedback control |
| **8** | Evaluation | Annotated GNN behavior, final write-up and submission |

## Immediate Next Steps

1. Finalize Phase 1 testing under various tick steps and conditions.
2. Refactor codebase to support `TickPhaseEngine` scaffolding.
3. Begin Phase 2 thermal system design and integration.
4. Prepare test runs and telemetry validation for heat + power interaction.

This sets the stage for scaling our simulation framework into a multi-physics, ML-augmented system suitable for orbital manufacturing modeling and intelligent scheduling.