Immediate Implementation Plan: First 30 Days

Let's break down the next steps to get this system up and running in your current environment.

Week 1: Core Integration

Days 1-2: TaskFactory Implementation

- [ ] Deploy the enhanced TaskFactory module to your environment
- [ ] Add necessary model updates (ReasoningEffort, ReasoningStrategy enums)
- [ ] Configure environment variables for auto-tuning settings
- [ ] Create a simple CLI test harness to validate functionality

Days 3-5: WebSocket Server Integration

- [ ] Update the WebSocket server with the TaskFactory integration
- [ ] Modify connection manager to handle reasoning effort metadata
- [ ] Implement process_task_creation with TaskFactory
- [ ] Add analytics events for task creation and completion

Week 2: Agent Adaptations

Days 6-7: Agent Code Updates

- [ ] Modify agent base class to handle reasoning strategy parameter
- [ ] Update Claude agent to use reasoning strategy for prompt selection
- [ ] Update GPT agent to adjust temperature based on reasoning effort
- [ ] Add strategy-specific logging for performance analysis

Days 8-10: Strategy Implementation

- [ ] Create specific handler implementations for each reasoning strategy:
- [ ] Direct answer (low effort)
- [ ] Chain-of-thought (medium effort)
- [ ] Chain-of-draft (high effort)
- [ ] Test each strategy with a variety of input tasks
- [ ] Measure and document baseline performance metrics

Week 3: Data Collection & Dashboard

Days 11-13: Data Collection System

- [ ] Implement standardized outcome recording in TaskFactory

- [ ] Create a task history database schema (or collection)
- [ ] Build a task outcome recorder service
- [ ] Set up daily aggregation jobs for analytics

Days 14-17: Dashboard Implementation

- [ ] Deploy the Task Dashboard React component
- [ ] Connect it to the WebSocket server for real-time updates
- [ ] Add filtering and sorting capabilities
- [ ] Create admin views for system analytics

Week 4: Testing, Optimization & Launch

Days 18-20: Synthetic Testing

- [ ] Create a library of test tasks with varying complexity
- [ ] Build a load testing suite to simulate concurrent tasks
- [ ] Implement A/B testing capability to compare effort classifiers
- [ ] Document all test results and performance metrics

Days 21-23: System Optimization

- [ ] Fine-tune TaskFactory thresholds based on test results
- [ ] Optimize WebSocket server for high concurrency
- [ ] Implement caching for frequently requested analytics
- [ ] Set up monitoring and alerting for key system metrics

Days 24-25: Documentation & Training

- [ ] Create comprehensive documentation for the system
- [ ] Build a user guide for the dashboard
- [ ] Document all API endpoints and data formats
- [ ] Create training materials for system administrators

Days 26-30: Staged Rollout

- [ ] Launch with 10% of traffic for 24 hours
- [ ] Monitor and resolve any issues
- [ ] Increase to 25%, then 50%, then 100% over 3 days
- [ ] Conduct post-launch review and document lessons learned

Collaboration Requirements

Engineering Resources Needed

- 1 Backend Engineer (Python/FastAPI) - Full-time
- 1 Frontend Developer (React) - Half-time
- 1 DevOps Engineer - Quarter-time
- 1 Data Scientist/ML Engineer - Half-time (Week 3 onward)

Key Technical Dependencies

- Redis instance with sufficient memory for task queue
- Database for task history (MongoDB or PostgreSQL)
- Access to agent API endpoints and heartbeat system
- Frontend build and deployment pipeline

Success Criteria

Minimum Viable Product

- TaskFactory correctly classifies >90% of tasks with appropriate effort
- Agents use reasoning strategy to modify their behavior
- Dashboard shows real-time task data with effort visualization
- System records outcomes for at least 95% of completed tasks

Performance Targets

- Latency increase from TaskFactory <50ms per task
- Dashboard loads in <2s with up to 1000 historical tasks
- System scales to handle at least 10 concurrent tasks
- Auto-tuning runs successfully after collecting 100+ outcomes

Next Phase Planning

By the end of this 30-day implementation, we should have:

1. A fully functional TaskFactory integrated with your WebSocket server
2. Agents that adapt their behavior based on reasoning strategy
3. A dashboard for monitoring and analyzing task complexity
4. Initial data collection for the next phase of evolution

This sets the foundation for moving to Gen 2 (Statistical Learning) in the next phase, where we'll use the collected data to train our first ML-based effort predictor and begin implementing the cluster-based task router.

Let's kick some technological ass! 🚀