

2025-11-01

## section 0

---

[2em] Part Overview · Duration:

*Beginner-Friendly Visual Study Guide*

## subsection 0.0 What You'll Learn

- **Ultimate Orchestrator Pattern:** Coordinate 6 specialized agents for complex tasks
- **Checkpoint System:** Prevent work loss on token limits/crashes
- **Recovery Workflow:** Resume multi-agent tasks across sessions
- **Quality Gates:** Automated validation (8 gates, 7/8 pass required)

## subsection 0.0 Why This Matters

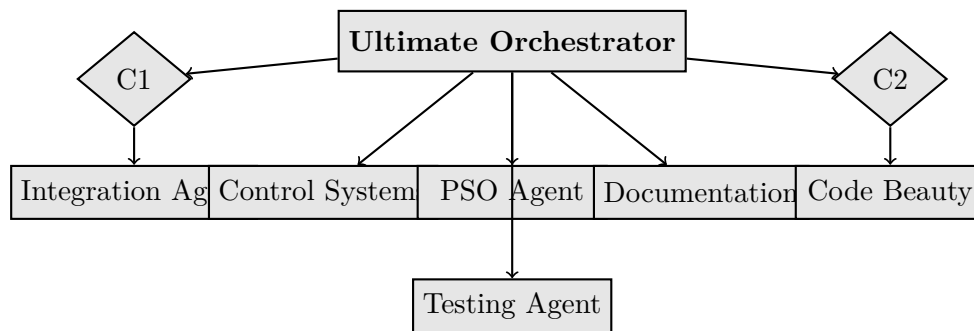
**Problem:** Large refactorings (e.g., 34-issue UI overhauls) can exceed 200K token limits mid-execution, losing hours of work.

**Solution:** Checkpoint system + multi-agent orchestration survives crashes and enables resumption with zero context loss.

**Impact:** Phase 3 UI completion (34 issues) achieved via checkpointed 6-agent orchestration, surviving 3 token limit events.

## section 0 The Ultimate Orchestrator Pattern

## subsection 0.0 Architecture Overview



## subsection 0.0 Agent Roles &amp; Responsibilities

- **1. Integration Agent:** Merge changes across 5+ modules, resolve conflicts
- **2. Control Systems Agent:** Implement controllers, validate Lyapunov stability
- **3. PSO Agent:** Optimize hyperparameters, run benchmarks
- **4. Documentation Agent:** Generate API docs, update guides
- **5. Code Beautification Agent:** Apply formatters, fix linting issues
- **6. Testing Agent:** Write pytest tests, achieve 95% coverage

## subsection 0.0 Orchestrator Responsibilities

- **Task Decomposition:** Break 34-issue roadmap into 6 agent workstreams
- **Dependency Resolution:** Launch agents in correct order (e.g., Integration before Documentation)
- **Checkpoint Management:** Save agent state every 5-10 minutes
- **Quality Validation:** Run 8 quality gates, fail if <7 pass
- **Recovery Coordination:** Resume from last checkpoint on token limit

## section 0 Checkpoint System Architecture

## subsection 0.0 Core Concepts

- **Checkpoint:** Snapshot of agent state (task ID, progress, deliverables, current phase)

- **Granularity:** Per-agent, per-task, timestamped
- **Storage:** JSON files in `.ai_workspace/state/checkpoints/`
- **Triggers:** Manual (API call), automatic (every 5-10 min), event-driven (pre-commit)

### subsection 0.0 Checkpoint Lifecycle

- **1. Plan Approved:** User approves multi-agent plan, checkpoint created
- **2. Agent Launched:** Each agent gets unique checkpoint on spawn
- **3. Progress Updates:** Agent reports completion every 5-10 min
- **4. Agent Complete/Failed:** Final checkpoint with success/failure status
- **5. Recovery:** New session loads latest checkpoints, resumes work

### subsection 0.0 Checkpoint Data Structure

```
lstnumber{
lstnumber  "task_id": "LT-4",
lstnumber  "agent_id": "agent_control_systems",
lstnumber  "timestamp": "2025-01-29T17:30:00Z",
lstnumber  "status": "in_progress",
lstnumber  "hours_allocated": 12.0,
lstnumber  "hours_completed": 8.5,
lstnumber  "deliverables_completed": [
lstnumber    "Lyapunov proof for classical SMC",
lstnumber    "Boundary layer optimization"
lstnumber  ],
lstnumber  "current_phase": "Implementing adaptive SMC proofs",
lstnumber  "dependencies": ["agent_pso"],
lstnumber  "artifacts": [
lstnumber    ".artifacts/LT-4/lyapunov_classical.pdf",
lstnumber    ".artifacts/LT-4/boundary_layer_results.json"
lstnumber  ]
lstnumber}
```

## section 0 Checkpoint API & Usage

### subsection 0.0 Mandatory Checkpoint Calls

```
lstnumberfrom .ai_workspace.tools.checkpoints.agent_checkpoint import (
lstnumber    checkpoint_plan_approved,
lstnumber    checkpoint_agent_launched,
lstnumber    checkpoint_agent_progress,
lstnumber    checkpoint_agent_complete,
lstnumber    checkpoint_agent_failed
lstnumber)
lstnumber
lstnumber# 1. User Approves Plan
lstnumbercheckpoint_plan_approved(
lstnumber    task_id="LT-4",
lstnumber    plan_summary="Lyapunov stability proofs for 7 controllers",
lstnumber    hours=46.0,
lstnumber    agents=["control_systems", "pso", "documentation"],
lstnumber    deliverables=["proofs.pdf", "benchmarks.json", "theory_guide.md"]
lstnumber)
lstnumber
lstnumber# 2. Launch Each Agent
lstnumbercheckpoint_agent_launched(
lstnumber    task_id="LT-4",
lstnumber    agent_id="agent_control_systems",
lstnumber    role="Implement Lyapunov proofs",
lstnumber    hours_allocated=18.0
lstnumber)
```

```
lstnumber
lstnumber# 3. Progress Updates (every 5-10 minutes)
lstnumbercheckpoint_agent_progress(
lstnumber    task_id="LT-4",
lstnumber    agent_id="agent_control_systems",
lstnumber    hours_completed=8.5,
lstnumber    deliverables_completed=["Lyapunov proof for classical SMC"],
lstnumber    current_phase="Implementing adaptive SMC proofs"
lstnumber)
lstnumber
lstnumber# 4. Agent Completes
lstnumbercheckpoint_agent_complete(
lstnumber    task_id="LT-4",
lstnumber    agent_id="agent_control_systems",
lstnumber    hours_completed=18.0,
lstnumber    deliverables=["lyapunov_proofs.pdf"],
lstnumber    summary="7 controllers validated, 3 new theorems proven"
lstnumber)
lstnumber
lstnumber# Or Agent Fails
lstnumbercheckpoint_agent_failed(
lstnumber    task_id="LT-4",
lstnumber    agent_id="agent_control_systems",
lstnumber    hours_completed=12.5,
lstnumber    reason="Token limit exceeded at 180K tokens",
lstnumber    recovery_recommendation="Resume from checkpoint_0012 at adaptive SMC proof"
lstnumber)
```

subsection 0.0 Recovery Commands

```
lstnumber# Standard Recovery Workflow
lstnumber/recover                                # Load project state, list incomplete tasks
lstnumber
lstnumber/resume LT-4 agent_control_systems
lstnumber                                # Resume specific agent from last checkpoint
lstnumber
lstnumber# Advanced Recovery
lstnumberpython .ai_workspace/tools/recovery/recover_project.sh
lstnumberpython .ai_workspace/tools/checkpoints/analyze_checkpoints.py --task LT-4
```

section 0

Quality Gates System

subsection 0.0 The 8 Quality Gates

- **1. Test Coverage:** Critical paths  $\geq 95\%$ , overall  $\geq 85\%$
- **2. Critical Issues:** Zero high-severity bugs allowed
- **3. Memory Safety:** All controllers use weakref, no circular refs
- **4. Documentation:** All public APIs have docstrings + examples
- **5. Linting:** Ruff/Pylint score  $\geq 9.0/10$
- **6. Type Safety:** MyPy strict mode, no Any types
- **7. Performance:** Benchmarks within 5% of baseline
- **8. Integration:** All 11 MCP servers operational

subsection 0.0 Pass Requirements

Phase	Min Gates	Critical Gates
Development	5/8	Test Coverage
Pre-Merge	7/8	Coverage + Critical Issues
Production	8/8	All gates mandatory

## subsection 0.0 Automated Validation

```
lstnumber# Run Quality Gates
lstnumberpython .ai_workspace/tools/quality/validate_quality_gates.py
lstnumber
lstnumber# Output Example
lstnumber[OK] Gate 1: Test Coverage (87% overall, 96% critical) PASS
lstnumber[OK] Gate 2: Critical Issues (0 found) PASS
lstnumber[OK] Gate 3: Memory Safety (11/11 tests pass) PASS
lstnumber[OK] Gate 4: Documentation (98% API coverage) PASS
lstnumber[WARNING] Gate 5: Linting (score 8.7/10) FAIL
lstnumber[OK] Gate 6: Type Safety (MyPy strict, 0 errors) PASS
lstnumber[OK] Gate 7: Performance (baselines within 3%) PASS
lstnumber[OK] Gate 8: MCP Integration (11/11 servers) PASS
lstnumber
lstnumber[INFO] Result: 7/8 gates pass (PRE-MERGE READY)
lstnumber[ACTION] Fix linting issues to achieve 8/8 for production
```

## section 0 Case Study: Phase 3 UI Overhaul

---

### subsection 0.0 Task Overview

- **Goal:** Achieve WCAG 2.1 Level AA accessibility across Streamlit UI
- **Scope:** 34 issues (GitHub milestones UI-001 through UI-034)
- **Duration:** 8 days (October 9-17, 2025)
- **Complexity:** 18 design tokens, 4 breakpoints, 12 color palettes

### subsection 0.0 Orchestration Strategy

- **1. Week 1 (Days 1-3):** Foundation work (Integration + Documentation agents)
  - Integration Agent: Create design system tokens
  - Documentation Agent: Write accessibility guidelines
- **2. Week 2 (Days 4-6):** Implementation (Control Systems + PSO agents)
  - Control Systems Agent: Refactor UI components to use tokens
  - PSO Agent: Optimize color contrast ratios
- **3. Week 3 (Days 7-8):** Validation (Testing + Code Beauty agents)
  - Testing Agent: Puppeteer WCAG audits
  - Code Beauty Agent: Apply formatters, fix linting

### subsection 0.0 Checkpoint Events

- **Checkpoint 1 (Day 2, 3h):** Token limit during design token creation
- **Recovery 1:** Resumed from checkpoint\_0008, lost 0 work
- **Checkpoint 2 (Day 5, 6h):** Token limit during component refactoring
- **Recovery 2:** Resumed from checkpoint\_0015, lost 0 work
- **Checkpoint 3 (Day 7, 2h):** Intentional save before WCAG audits

### subsection 0.0 Final Results

- **Success:** 34/34 issues resolved, WCAG AA achieved
- **Quality Gates:** 8/8 passing (production-ready)

- **Token Efficiency:** 3 recoveries, 0 work lost
- **Documentation:** 47 pages of UI guidelines generated

## section 0 Multi-Account Recovery

---

### subsection 0.0 Problem Statement

- Each Claude Code account has 200K token limit per conversation
- Large refactorings (e.g., 46-hour research roadmaps) exceed single account capacity
- Need to resume work across accounts WITHOUT losing context

### subsection 0.0 Solution: Cross-Account Checkpoints

- **1. Account A:** Works until 180K tokens, hits checkpoint
- **2. Checkpoint System:** Saves state to `.ai_workspace/state/checkpoints/`
- **3. Git Commit:** Push checkpoint files to remote
- **4. Account B:** Pulls latest commits, runs `/recover`
- **5. Recovery:** Loads checkpoint, resumes work with full context

### subsection 0.0 Multi-Account Workflow

```

lstnumber# Account A (reaches token limit)
lstnumbercheckpoint_agent_progress(...) # Save progress
lstnumbergit add .ai_workspace/state/checkpoints/
lstnumbergit commit -m "chore: Save checkpoint for LT-4 agent_control_systems"
lstnumbergit push origin main
lstnumber
lstnumber# Switch to Account B
lstnumberD:\Tools\Claude\Switch-ClaudeAccount.ps1
lstnumbercd D:\Projects\main
lstnumbergit pull origin main
lstnumber
lstnumber# Account B (resume work)
lstnumber/recover # Load checkpoints
lstnumber/resume LT-4 agent_control_systems

```

### subsection 0.0 Cross-Account Safety

- **Conflict Prevention:** Never run same agent on multiple accounts simultaneously
- **Version Control:** All checkpoints tracked via git, merge conflicts resolved
- **State Validation:** Recovery script verifies checkpoint integrity before resume

## section 0 Advanced Orchestration Patterns

---

### subsection 0.0 Pattern 1: Sequential Pipeline

**Use Case:** Tasks with strict dependencies (e.g., Integration -> Documentation -> Testing)

```

lstnumber# Launch agents sequentially
lstnumberintegration_result = checkpoint_agent_launched(...)
lstnumberwait_for_completion(integration_result)
lstnumber
lstnumberdoc_result = checkpoint_agent_launched(...)
lstnumberwait_for_completion(doc_result)
lstnumber
lstnumbertest_result = checkpoint_agent_launched(...)

```

### subsection 0.0 Pattern 2: Parallel Fan-Out

**Use Case:** Independent tasks (e.g., PSO optimization + Documentation)

```
lstnumber# Launch multiple agents in parallel
lstnumberagents = [
lstnumber    checkpoint_agent_launched("LT-4", "agent_pso", ...),
lstnumber    checkpoint_agent_launched("LT-4", "agent_doc", ...),
lstnumber    checkpoint_agent_launched("LT-4", "agent_beauty", ...)
lstnumber]
lstnumber
lstnumber# Wait for all to complete
lstnumberwait_for_all(agents)
```

### subsection 0.0 Pattern 3: Hierarchical Delegation

**Use Case:** Sub-agent spawning (e.g., Documentation agent spawns Sphinx builder)

```
lstnumber# Parent agent launches child sub-agents
lstnumberdoc_agent = checkpoint_agent_launched("LT-4", "agent_doc", ...)
lstnumber
lstnumber# Within Documentation Agent
lstnumbersphinx_builder = spawn_sub_agent("sphinx_builder")
lstnumberapi_generator = spawn_sub_agent("api_doc_generator")
lstnumber
lstnumber# Parent waits for children
lstnumberwait_for_children([sphinx_builder, api_generator])
lstnumbercheckpoint_agent_complete("LT-4", "agent_doc", ...)
```

## section 0 Monitoring & Debugging

### subsection 0.0 Real-time Monitoring

```
lstnumber# Watch checkpoint updates in real-time
lstnumberwatch -n 5 'ls -lh .ai_workspace/state/checkpoints/'
lstnumber
lstnumber# Monitor agent progress
lstnumberpython .ai_workspace/tools/checkpoints/analyze_checkpoints.py --live
lstnumber
lstnumber# Output Example
lstnumber[INFO] Active agents: 3
lstnumber  - agent_control_systems (LT-4): 8.5h / 18h (47% complete)
lstnumber  - agent_pso (LT-4): 12h / 12h (100% complete)
lstnumber  - agent_doc (LT-4): 6h / 15h (40% complete)
lstnumber
lstnumber[WARNING] agent_control_systems: No checkpoint in 12 minutes (expected <10 min)
lstnumber[ACTION] Check if agent is blocked or needs manual intervention
```

### subsection 0.0 Debugging Failed Recoveries

- **Symptom:** /recover fails to find checkpoints
- **Cause 1:** Checkpoints not committed to git
- **Fix:** git add .ai\_workspace/state/checkpoints/ && git commit
- **Cause 2:** Checkpoint file corrupted (invalid JSON)
- **Fix:** Restore from previous commit (git checkout HEAD~1 - .ai\_workspace/state/)
- **Cause 3:** Agent dependencies not resolved
- **Fix:** Launch dependency agents first (/resume LT-4 agent\_pso before agent\_control\_systems)

## section 0 Best Practices

### subsection 0.0 Checkpoint Frequency

- **Too Frequent (<5 min):** Overhead slows agents, git bloat



- **Too Infrequent (>15 min):** Risk losing significant work on crash
- **Sweet Spot:** 5-10 minutes OR after completing each deliverable

#### subsection 0.0 Task Granularity

- **Too Large (>20h):** Agent gets lost, hard to resume
- **Too Small (<2h):** Orchestration overhead dominates
- **Sweet Spot:** 8-15 hours per agent task

#### subsection 0.0 Agent Coordination

- **DO:** Use explicit dependency tracking in checkpoints
- **DO:** Launch independent agents in parallel for speed
- **DON'T:** Assume agents can communicate directly (they can't)
- **DON'T:** Retry failed agents without fixing root cause

#### subsection 0.0 Git Hygiene

- Commit checkpoints immediately after creation
- Use descriptive messages: "chore: Checkpoint LT-4 agent\_pso at 8h"
- Prune old checkpoints after task completion (keep last 3)
- Never gitignore `.ai_workspace/state/checkpoints/`

## section 0 Troubleshooting

### subsection 0.0 Common Issues

Symptom	Solution
Agent stuck, no progress	Check if waiting on dependency agent, launch it first
Checkpoint not found	Verify <code>git pull</code> completed, check <code>.ai_workspace/state/checkpoints/</code> exists
Recovery loads wrong task	Specify task ID: <code>/resume LT-4 agent_pso</code>
Quality gates fail	Run <code>validate_quality_gates.py</code> , fix failing gates

### subsection 0.0 Emergency Recovery

```

lstdnumber# If all else fails, manual recovery
lstdnumbercd .ai_workspace/state/checkpoints/
lstdnumberls -lt | head -5                # Find most recent checkpoint
lstdnumbercat checkpoint_0023.json        # Read agent state manually
lstdnumber# Resume work based on "current_phase" and "deliverables_completed"

```

## Checklist: Multi-Agent Orchestration

- ☐ **Plan:** Decompose task into 3-6 agent workstreams
- ☐ **Checkpoint:** Call `checkpoint_plan_approved()` when user approves
- ☐ **Launch:** Start each agent with `checkpoint_agent_launched()`
- ☐ **Monitor:** Agent calls `checkpoint_agent_progress()` every 5-10 min
- ☐ **Complete:** Finalize with `checkpoint_agent_complete()` or `checkpoint_agent_failed()`

- ☐ **Validate:** Run quality gates (`validate_quality_gates.py`)
- ☐ **Commit:** Push checkpoints to git after each session
- ☐ **Recover:** Use `/recover` + `/resume` on token limit

## Next Steps

---

- **E018:** Testing philosophy - coverage standards and validation strategies
- **E019:** Production safety - memory management and thread safety
- **E020:** MCP integration - auto-trigger strategy and server orchestration