Long-Running Agent

Implementation Documentation

B2B Lead Discovery Agent

with Memory, Checkpointing & Context Engineering

Generated: February 04, 2026

1. Overview

This document describes the implementation of long-running agent capabilities for the B2B Lead Discovery Agent. The implementation transforms a single-run sequential agent into an autonomous system with persistent memory, state management, and intelligent context engineering.

Key Capabilities Added

* Three-tier memory system (short-term, long-term, working)
* Checkpoint-based state management with resume capability
* Context engineering for intelligent prompt injection
* Auto-recovery from failures with retry logic
* Learning from past successes and failures

Problem Solved

Traditional AI agents suffer from context window limitations. When processing multiple items sequentially, the agent's limited active memory becomes overloaded. This leads to degraded performance where later items are processed less accurately. The long-running agent architecture solves this by:

1. Externalizing memory to persistent storage
2. Checkpointing state for recovery
3. Selective context retrieval instead of loading everything

2. Architecture

Execution Flow

The agent follows a Plan -> Act -> Observe -> Remember -> Re-Plan cycle:

1. Goal is stored as primary objective
2. Pipeline stages are executed with checkpointing
3. Each stage result is observed and saved to memory
4. On failure, agent retries or pauses for manual intervention
5. On completion, learnings are extracted and stored

Memory Architecture

Three types of memory work together:

Short-term Memory (Session)

Stores recent actions and events during the current session. Limited to 20 items. Cleared when session ends. Used for immediate context and recent decisions.

Long-term Memory (Persistent)

Persisted to disk (data/memory.json). Stores:
- Company analyses and outcomes
- Successful patterns with success rates
- Past failures to avoid
- General insights and learnings

Working Memory (Scratchpad)

Temporary storage for current task reasoning. Holds intermediate values like current lead ID, company being processed. Cleared after task completion.

3. Implementation Details

New Files Created

memory/memory_manager.py

Core memory management class with:
- add_to_short_term(): Log events to session memory
- remember_company(): Store company analysis outcome
- recall_company(): Retrieve past company data
- remember_pattern(): Store successful patterns
- remember_failure(): Log failures to avoid
- get_stats(): Memory statistics

memory/state_manager.py

Checkpoint and state management with:
- start_execution(): Begin new pipeline execution
- start_stage() / complete_stage(): Track stage progress
- fail_stage(): Handle failures with retry logic
- resume_execution(): Resume from checkpoint
- get_resumable_executions(): List paused executions

memory/context_builder.py

Context engineering for intelligent prompts:
- build_discovery_context(): Context for company analysis
- build_enrichment_context(): Context for contact enrichment
- build_verification_context(): Context for lead scoring
- extract_learnings(): Extract patterns from results

Files Modified

workflow.py

Added LongRunningWorkflow class that:
- Initializes memory and state managers
- Calls checkpoint after each pipeline stage
- Injects memory context into each agent
- Handles interrupts gracefully (Ctrl+C)
- Extracts learnings after completion
- Supports resume from checkpoint

agent.py

Added new CLI commands:
- resume: List and resume paused executions
- status: Display memory statistics
- learn: Show learned patterns and insights
- forget <company>: Clear memory for specific company

Data Storage

New JSON files created in data/ directory:

* memory.json - Long-term memory (companies, patterns, failures, insights)
* checkpoints.json - Execution state checkpoints
* execution_history.json - Event log for debugging

4. How It Works

Example: Analyzing a Company

When you run 'analyze Microsoft':

1. LongRunningWorkflow creates new execution
2. StateManager saves initial checkpoint
3. MemoryManager checks for past Microsoft analysis
4. ContextBuilder injects past context into discovery prompt
5. Discovery agent runs with enriched context
6. StateManager checkpoints after discovery completes
7. Process repeats for structure, roles, enrichment, verification
8. On completion, learnings extracted and stored
9. Company outcome saved to long-term memory

Example: Resuming Interrupted Analysis

If you interrupt with Ctrl+C during enrichment:

1. StateManager saves current state to checkpoint
2. Execution marked as 'paused'
3. Later, run 'resume' command
4. StateManager loads checkpoint
5. Completed stages (discovery, structure, roles) are restored
6. Pipeline continues from enrichment stage

Example: Learning from Past

When analyzing Alphabet after analyzing Google:

1. ContextBuilder checks long-term memory
2. Finds Google analysis from previous session
3. Injects context: 'Similar company analyzed before...'
4. Agent uses this context for better decisions
5. Patterns that worked for Google inform Alphabet analysis

5. Usage Guide

New Commands

```
> status
  Shows memory statistics:
  - Total analyses performed
  - Companies remembered
  - Patterns learned
  - Failures recorded
```

```
> learn
  Shows what agent has learned:
  - Successful patterns with success rates
  - Recent insights
  - Failures (learning opportunities)
```

```
> resume
  Lists paused/failed executions
  Automatically resumes if only one exists
```

```
> forget Microsoft
  Clears all memory of Microsoft
  Use to force fresh analysis
```

6. Summary

The long-running agent implementation transforms the B2B Lead Discovery Agent from a stateless single-run system into an autonomous agent that:

1. Remembers past analyses across sessions
2. Can be interrupted and resumed safely
3. Learns from successes and failures
4. Uses past knowledge to improve accuracy
5. Handles errors gracefully with retry logic

This architecture follows the pattern used by advanced AI agents like Manus AI and AutoGPT, implementing the core Plan-Act-Observe-Remember cycle with externalized memory and state management.

Files Summary

* memory/__init__.py - Module exports
* memory/memory_manager.py - 3-tier memory (300+ lines)
* memory/state_manager.py - Checkpointing (400+ lines)
* memory/context_builder.py - Context engineering (250+ lines)
* workflow.py - Updated with LongRunningWorkflow class
* agent.py - Added 4 new commands (150+ lines added)
* README.md - Updated documentation