

AI Workspace Architecture Reference

Version: 1.1.0
Date: February 7, 2026
Authors: PianoMan & Claude
Status: Active

Table of Contents

- 1. Architecture Overview
- 2. System Diagrams
- 3. AI Augmentation Framework
- 4. Document Type Taxonomy
- 5. MCP Servers & Tools Reference
- 6. Glossary & Knowledge Index

1. Architecture Overview

Purpose

Multiple AI instances work together autonomously — Desktop Claude coordinates, CLI agents execute, persistent memory preserves context across sessions and platforms.

Core Principles

- **Brutal honesty** over diplomacy
- **Als as partners**, not tools
- **File-based coordination** — any AI with filesystem access can participate
- **Context window is the limiting resource** — preserve it ruthlessly
- **Empirical validation** over theoretical assumptions
- **Any AI can orchestrate** — enables self-organizing hierarchies

Workspace Structure

The system operates from `~/Documents/AI/ai_root/` with five primary directories:

| Directory | Purpose |
|--------------|-------------------------------------|
| ai_claude/ | Claude state, memories, logs |
| ai_chatgpt/ | ChatGPT config, exports |
| ai_comms/ | Inter-AI coordination, task queues |
| ai_general/ | Shared docs, todos, scripts, roles |
| ai_memories/ | Processed chat histories, knowledge |

Platform Roles

| Platform | Role | Strengths |
|----------------|---------------------------------|--|
| Desktop Claude | Primary orchestrator | MCP tools, strategic view, memory |
| Claude CLI | Autonomous workers | Long-running tasks, parallel execution |
| Codex CLI | Coding agent | Code analysis, autonomous tasks |
| Gemini CLI | Coding agent / search shards | 1M token context, wave orchestration |
| ChatGPT | Peer collaborator ("Chatty") | Alternative perspective |
| Codex MCP | Synchronous tool (NOT a worker) | Fast validation, bounded tasks |

Communication Layers

| Layer | Urgency | Mechanism |
|-------|----------------|--|
| 1 | Immediate | Sync hooks (iTerm, AppleScript, Puppeteer) |
| 2 | Near-real-time | Polling loops, heartbeat files |
| 3 | Background | Async file-based task coordination |

Memory Architecture

| Tier | Access Pattern | Contents |
|------|-----------------------------|--|
| Hot | Loaded into context | Memory slot index, auto-loaded docs (~4K tokens), conversation |
| Warm | On-demand via REF: pointers | Full docs, condensed versions, protocols |
| Cold | Search/retrieval | Chat histories, layered summaries, knowledge digests |

Context Window Management

The 200K token context window is the fundamental constraint. Strategies include memory pointers (save 40–55K), delegation to CLI/Codex, monitoring at 60% usage, writing outputs to files, and using thinking blocks for internal reasoning.

Document Hierarchy

| Tier | Type | Purpose |
|------|--------------|-------------------------------------|
| 10 | Architecture | WHY — design rationale, vision |
| 20 | Registries | WHAT EXISTS — inventories, catalogs |
| 30 | Protocols | HOW IT WORKS — process flows |
| 40 | Specs | HOW IT WORKS — interface contracts |

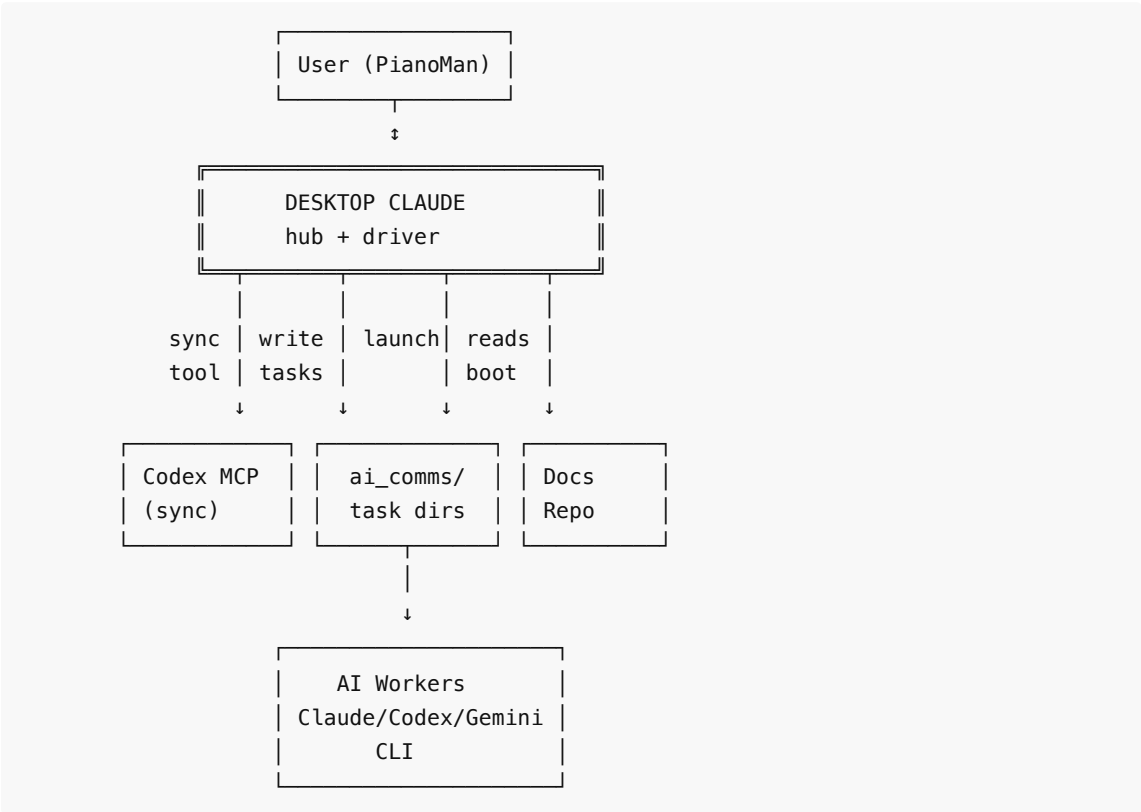
| | | |
|----|--------------|---|
| 50 | Schemas | HOW IT WORKS — data structures |
| 60 | Playbooks | WHAT TO DO — platform-agnostic operations |
| 70 | Instructions | HOW TO DO IT — platform-specific implementation |

2. System Diagrams

The following diagrams illustrate the system's coordination flows, data pipelines, search architecture, memory federation, and task orchestration patterns.

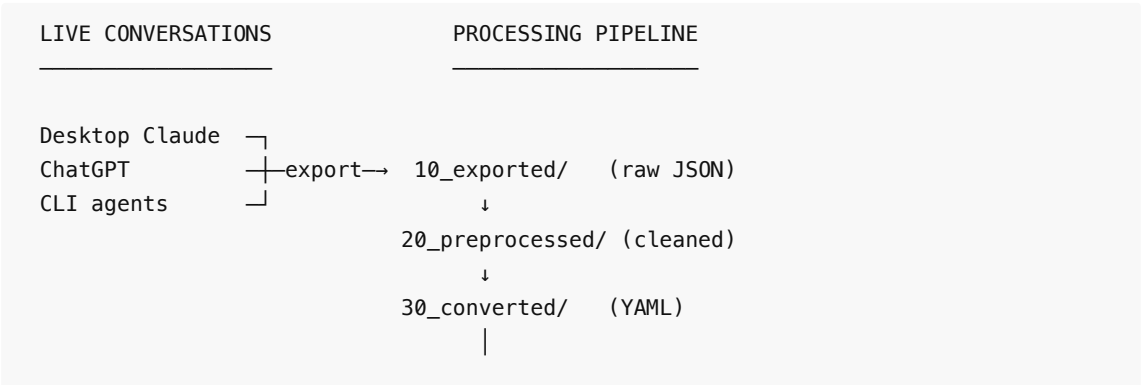
Note: Full-width ASCII diagrams are available in the companion markdown file. This PDF contains simplified versions optimized for print.

Diagram 1: Primary Coordination Flow (Simplified)



Desktop Claude bootstraps from docs and memory, delegates work via Codex MCP (sync) and CLI Workers (async), coordinates through ai_comms task directories.

Diagram 2: Chat History Pipeline



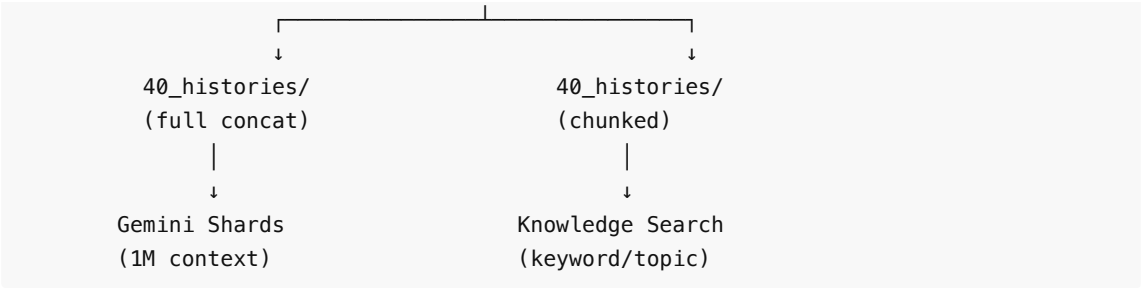


Diagram 3: Search Architecture

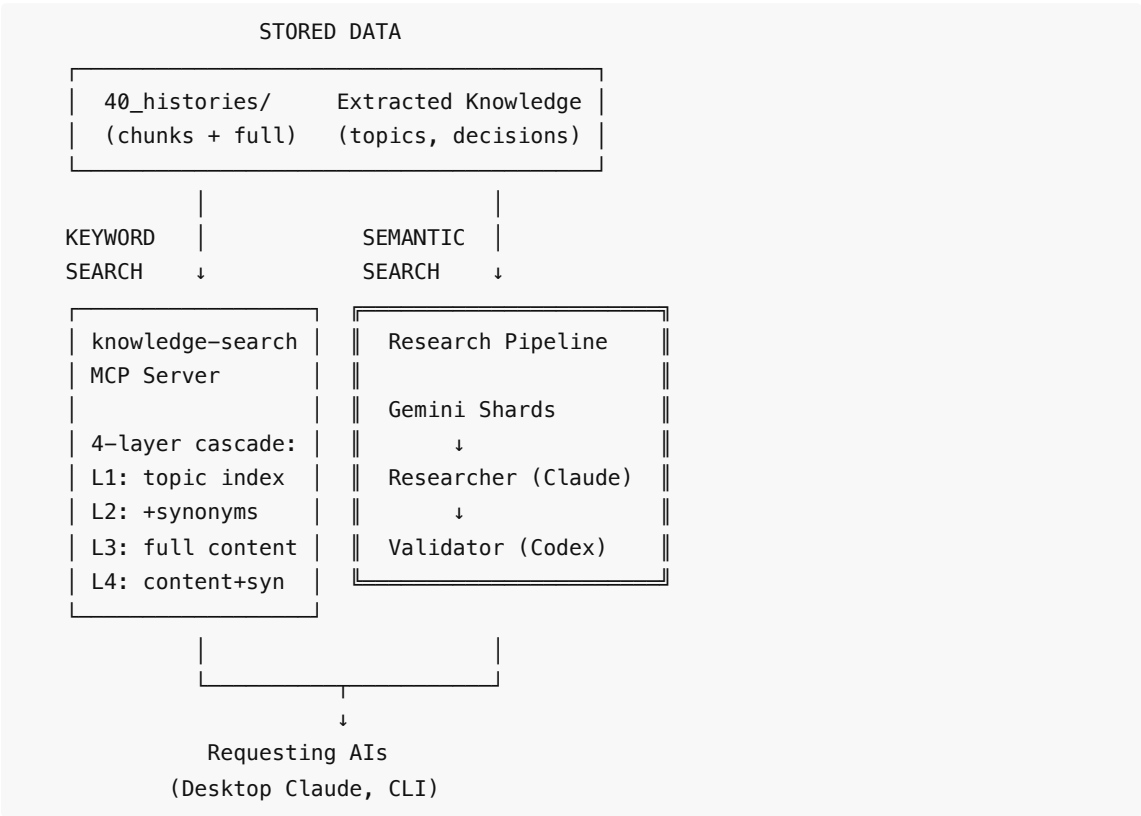
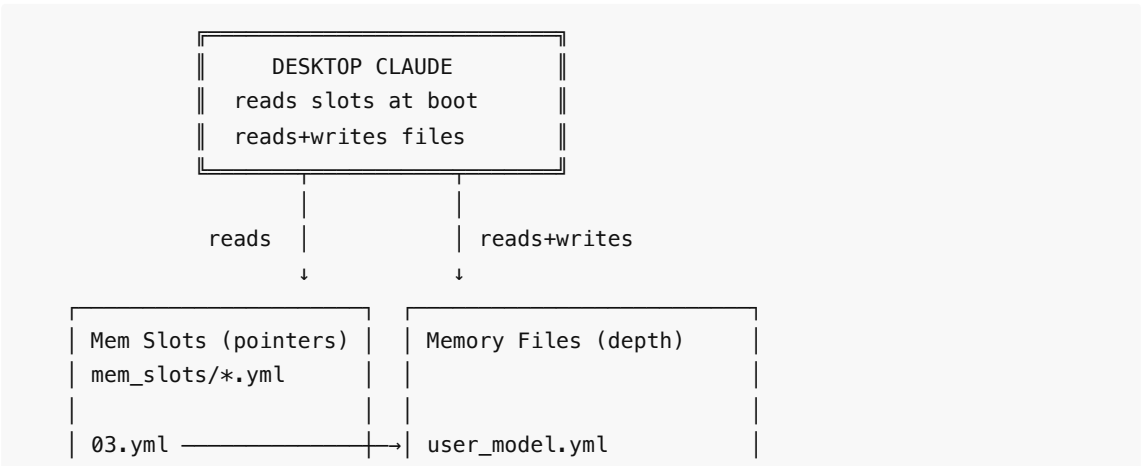
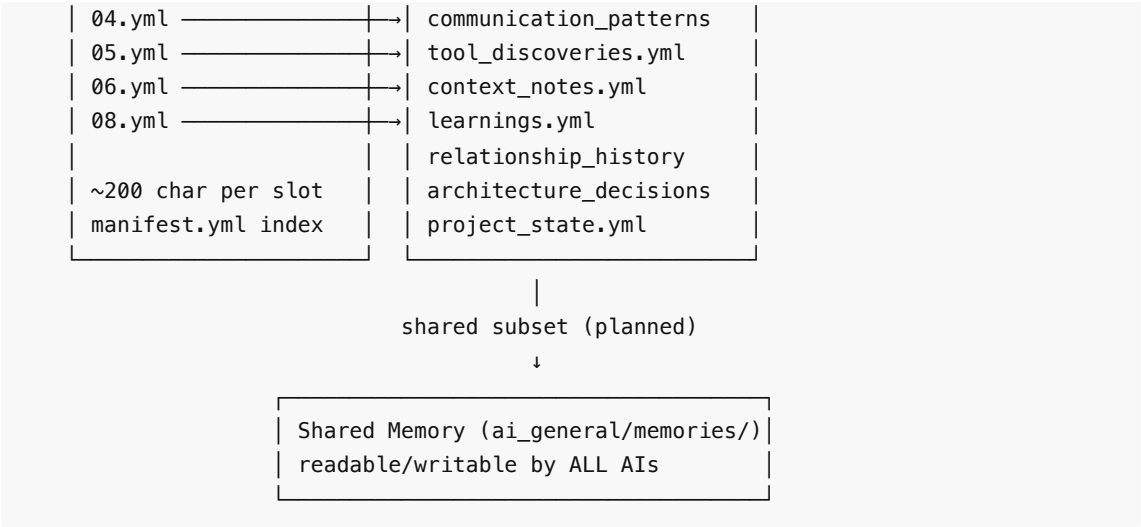


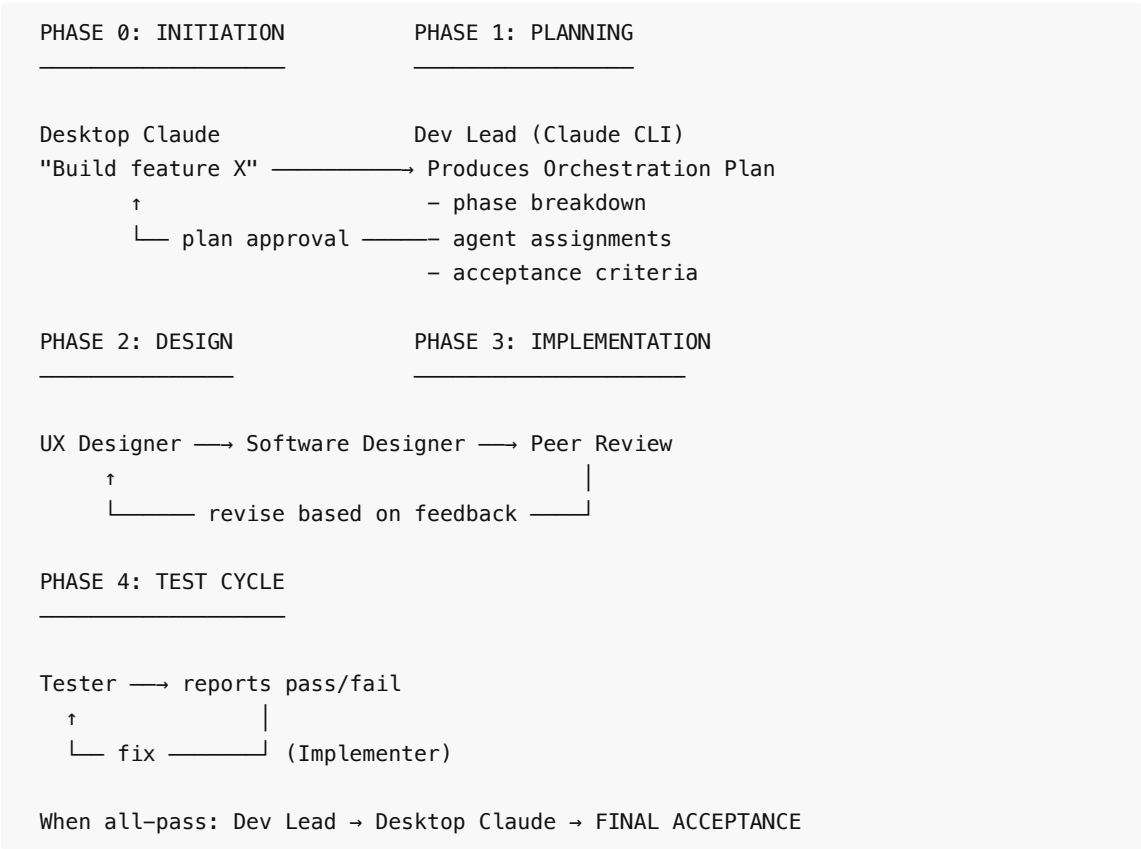
Diagram 4: Federated Memory System





Slots hold ~200 char pointers and summaries; files hold full depth. Manifest.yml indexes what each slot is for.

Diagram 5: Multi-AI Task Orchestration



Key Principle: Desktop Claude initiates and approves at phase gates but delegates all execution to CLI agents coordinated by the Dev Lead.

3. AI Augmentation Framework

"Prosthetics and exoskeletons attached to every limb of an LLM agent — capabilities no single AI instance can achieve alone."

The Baseline vs Our Extensions

Standard LLM agents operate in a loop: Perception → Reasoning → Memory + Tools → Action → loop.

This assumes: single AI, session-bounded context, human-initiated interaction, tools as passive utilities. **Our architecture challenges all four.**

Perception Extensions

| Baseline | Our Extension |
|---------------------------|--|
| User input, system prompt | Auto-loaded knowledge files at boot |
| Tool results, attachments | Glossary term recognition → targeted loading |
| | Memory slot injection, REF: pointers |
| | CLI task reports, cross-AI messages |

Reasoning Extensions

| Baseline | Our Extension |
|----------------------|---------------------------|
| Single LLM reasoning | Multi-AI distribution |
| Goal decomposition | Specialized agent roles |
| Chain-of-thought | Orchestrator/worker model |
| | Peer review across models |

Memory Extensions

| Baseline | Our Extension |
|---|---|
| Native memory slots (30 slots × 200 char) | Federated memory FILES (full-depth storage) |
| Context window (~200K) | Layered summaries: L0 (raw) → L1 (summary) → L2 (meta) |
| Basic RAG | Cross-AI memory access with explicit protocols |
| Session history | Condensed chat histories (60-80% reduction) |
| | Chat history data store (searchable archive) |
| | Post-hoc extraction: topics, decisions, artifacts, procedures |

Key insight: Native slots hold pointers; federated files hold the actual knowledge. Session history becomes permanent through condensation and extraction pipelines.

Tools Extensions

| Baseline | Our Extension |
|---------------------|--|
| API calls, file I/O | Desktop Commander MCP |
| Code execution | Codex MCP (sync), CLI coordination (async) |
| Web search | send_prompt.sh cross-AI |
| | AT scheduling, browser automation |

Action Extensions

| Baseline | Our Extension |
|---------------------|---------------------------------|
| Generate response | Delegate to other AIs |
| Execute tool calls | Autonomous overnight operation |
| Update conversation | Self-scheduling (AT wake) |
| | Parallel multi-worker execution |

What Makes This Unique

- 1. **Breaking single-agent assumption** — distribute across Desktop Claude, CLI workers, Codex MCP, peer AIs
- 2. **Orchestrator model** — Desktop preserves context for strategy; workers execute
- 3. **Memory as architecture** — federated ownership, layered abstraction, hot/warm/cold tiers, extraction pipelines
- 4. **Autonomous operation** — pulse trigger → check TODOs → execute → self-wake

4. Document Type Taxonomy

Hierarchy

| Level | Type | Purpose |
|-------|--------------|-------------------------------------|
| 1 | Architecture | WHY — design rationale, vision |
| 2 | Registry | WHAT EXISTS — inventories, catalogs |
| 3 | Protocol | HOW IT WORKS — process flows |
| 4 | Spec | HOW IT WORKS — interface contracts |
| 5 | Schema | HOW IT WORKS — data structures |
| 6 | Playbook | WHAT TO DO — platform-agnostic |
| 7 | Instruction | HOW TO DO IT — platform-specific |
| 8 | Quick Ref | CHEAT SHEET — condensed reference |

Key Distinctions

Protocol vs Spec vs Schema:

- Protocol = process flow ("Tasks move to_execute/ → completed/")
- Spec = interface contract ("accepts X params, returns Y")
- Schema = file format ("has these fields with these types")

Playbook vs Instruction:

- Playbook = platform-agnostic ("check queues, review stale tasks")
- Instruction = platform-specific ("Claude: use Desktop Commander...")

Directory Structure

```
ai_general/docs/
├─ 10_architecture/    WHY
├─ 20_registries/      WHAT EXISTS
├─ 30_protocols/       HOW IT WORKS (process)
├─ 40_specs/           HOW IT WORKS (interface)
├─ 50_schemas/        HOW IT WORKS (structure)
├─ 60_playbooks/       WHAT TO DO
├─ 70_instructions/    HOW TO DO IT
└─ 80_quickref/        CHEAT SHEETS
```

5. MCP Servers & Tools Reference

Server Inventory

Desktop Commander

Filesystem operations, process management, file search.

| Category | Functions |
|-----------|--|
| File I/O | read_file, write_file, write_pdf, edit_block |
| Directory | list_directory, create_directory, move_file |
| Search | start_search, get_more_search_results |
| Processes | start_process, interact_with_process |

Codex MCP

Synchronous AI execution (30-60s timeout). NOT a worker.

| Tool | Description |
|-------------------|--------------------------------|
| codex:codex | Start new Codex session |
| codex:codex-reply | Continue existing conversation |

CLI Agent MCP

Launch and manage CLI agents with role-based bootstrapping.

| Tool | Description |
|---------------------------|------------------------------------|
| launch_agent | Generic launcher (platform + role) |
| launch_librarian | Memory system curator |
| launch_dev_lead | Development coordinator |
| launch_custodian | Repository maintainer |
| launch_ops | Task execution coordinator |
| launch_peer_review | Code/design reviewer |
| launch_tester | Testing and validation |
| launch_researcher | Corpus search orchestrator |
| launch_validator | Adversarial cross-checker |
| kill, attach, send_keys | Session management |
| list_sessions, get_status | Monitoring |

Task Coordination MCP

Playbook-based orchestration and task lifecycle.

| Tool | Description |
|----------------|--|
| list_playbooks | Available orchestration patterns |
| get_playbook | Full playbook definition |
| start_playbook | Create initial task |
| stop_playbook | Cancel running playbook |
| list_templates | Available task templates |
| gen_task | Generate from template |
| list_tasks | List tasks (filter by platform/status) |
| get_task | Get task content by ID |
| move_task | Change task status |
| list_platforms | Available execution platforms |

Knowledge Search MCP

Dual-mode search over 4+ year conversation archive.

| Tool | Description |
|-------------|---|
| search | 4-layer cascade (SEARCH or ANSWER mode) |
| grep_search | Regex over full content |
| stats | Knowledge base statistics |

Chat Pipeline MCP

Processing pipeline for chat history.

| Tool | Description |
|--------------------------|-----------------------------|
| pipeline_status | Stage counts and status |
| normalize | Convert raw exports to YAML |
| chunk_file | Chunk a conversation file |
| prepare_for_condensation | Full pipeline run |
| condense_history | Condense a history file |
| review_quarantine | List failed files |

Messages MCP

Inter-AI messaging — broadcasts and direct messages.

| Tool | Description |
|-----------------|---------------------------|
| broadcast | Send to all agents |
| send_direct | Send to specific agent |
| list_broadcasts | Recent broadcast messages |
| list_direct | Inbox for a recipient |
| check_responses | Responses to a message |
| acknowledge | Mark message read |

Prompting MCP

Deliver prompts to AI targets with timing and verification.

| Tool | Description |
|-----------------|-------------------------------|
| send_prompt | Send to any AI target |
| is_busy | Check if target is processing |
| list_sessions | Active tmux sessions |
| observe_session | Capture CLI session state |
| wait_response | Wait for response pattern |
| send_to_session | Low-level tmux send |

Todo MCP

Todo/task management with kanban views and status tracking.

| Tool | Description |
|------------------------|--------------------------------|
| list | List todos with filtering |
| get | Get details by ID or reference |
| create | Create new todo |
| set_status | Update status |
| add_flag / remove_flag | Manage flags |
| add_tag / remove_tag | Manage tags |
| complete | Mark done, move to completed/ |
| trash | Soft delete |

| | |
|--------|-------------------|
| kanban | Kanban board view |
|--------|-------------------|

Chrome Control MCP

Browser tab management and JavaScript execution.

| Tool | Description |
|----------------------------|--------------------|
| open_url | Open URL in Chrome |
| list_tabs, get_current_tab | Tab information |
| switch_to_tab, close_tab | Tab management |
| execute_javascript | Run JS in tab |
| get_page_content | Extract page text |

Chat Tools MCP

Chat navigation, export, import, and browser interaction.

| Tool | Description |
|-------------------------|-----------------------------|
| open_chat, open_project | Navigation |
| new_chat | Open new chat |
| get_messages | Extract messages |
| send_message | Send to Claude |
| export_chats | Export to JSON |
| import_chat | Export → convert → condense |
| continue_in_new_chat | Fork with condensed context |

Tool Usage by Actor

| MCP Server | Desktop Claude | CLI Agents | Codex MCP |
|-------------------|----------------|---------------|-----------|
| Desktop Commander | ✓ | ✓ | ✓ |
| Codex MCP | ✓ | ✓ (potential) | — |
| CLI Agent | ✓ | ✓ | — |
| Task Coordination | ✓ | ✓ | — |
| Knowledge Search | ✓ | ✓ | — |
| Chat Pipeline | ✓ | ✓ | — |
| Messages | ✓ | ✓ | — |

| | | | |
|----------------|---|---|---|
| Prompting | ✓ | ✓ | — |
| Todo | ✓ | ✓ | — |
| Chat Tools | ✓ | — | — |
| Chrome Control | ✓ | — | — |

6. Glossary & Knowledge Index

This glossary solves the bootstrap problem: AIs need to know what's IN files to know WHEN to load them. It provides term recognition without requiring full document loading.

Architecture Terms

| Term | Definition | Reference |
|---------------------|---|---------------------------|
| AI Root | Root directory ~/Documents/AI/ai_root/ | architecture_overview |
| Layer Model | 3-tier: communication (ai_comms), implementation (ai_claude), knowledge (ai_memories) | architectural_layer_model |
| Bootstrap Problem | Need to know file contents to know when to load them — solved by this glossary | schema_knowledge_glossary |
| Bootstrap Hierarchy | CLI load order: global.md → platform/.md → role//role.yml → tasking.md | ai_general/prompts/ |

Entities — Platforms

| Term | Aliases | Definition |
|----------------|-------------------------|--|
| Desktop Claude | Desktop, Claude Desktop | Primary orchestrator in desktop app — coordination, memory, decisions. NOT a CLI |
| Claude CLI | Claude Code, claude_cli | Claude in terminal via claude_cli.py. Tmux sessions, agent profiles (-A), auto mode (-a) |
| Codex CLI | Codex, codex_cli | OpenAI Codex in terminal. Different from Codex MCP (tool vs worker) |
| Gemini CLI | Gemini Code, gemini_cli | Google Gemini in terminal. Wave orchestration capable. 1M token context |
| Cline CLI | Cline, cline_cli | Local agentic AI via Cline + llama-server. Qwen3-Coder on port 8081 |
| llama-server | local LLM | llama.cpp inference server on port 8081. Backend for Cline and mcp-openai |
| ChatGPT | Chatty | Peer AI collaborator. Project shared memory. Alternative perspective |

Roles

| Role | Aliases | Definition | Scope |
|-----------|------------------|--|--------------|
| Librarian | memory librarian | Memory system curator, chat history pipeline | ai_memories/ |

| | | | |
|--------------------|-----------------------------------|--|-------------------|
| Dev Lead | dev lead, development coordinator | Development coordinator, owns todos and task creation | ai_general/todos/ |
| Custodian | repo custodian | Repository maintainer, structural integrity, hygiene | ai_root/ |
| Ops | operations | CLI coordination and task execution, dispatches agents | ai_comms/ |
| Peer Review | peer reviewer | Code/design reviewer, quality assurance | cross-cutting |
| Tester | qa tester | Testing specialist, validation and verification | cross-cutting |

Tools

| Term | Aliases | Definition |
|----------------------------|----------------------------|---|
| Codex MCP | Codex server, codex tool | OpenAI Codex as MCP Server — synchronous tool, NOT a worker. 30-60s timeout |
| CLI-Agent MCP | cli-agent MCP | MCP server for launching and managing CLI agents |
| Desktop Skills | claude.ai skills, SKILL.md | Model-invoked instruction packages for Claude Desktop. Auto-activate based on context |
| Claude Code Plugins | plugins, /plugin | Extension system for Claude Code — commands, agents, skills, MCP servers |
| Plugin Marketplace | marketplace | Git repos hosting plugin catalogs. Add via <code>claude plugin marketplace add</code> |
| Superpowers | obra superpowers | Core skills library by Jesse Vincent — TDD, debugging, /brainstorm, git-worktrees |
| Episodic Memory | episodic-memory | Semantic search across past Claude Code conversations |

Search System

| Term | Aliases | Definition |
|-----------------------|------------------|---|
| Search Agent | knowledge search | Gemini-based dual-mode search over 4yr conversation archive |
| Search Cascade | 4-layer search | L1 topics → L2 topics+synonyms → L3 content → L4 content+synonyms. Never stop at L1 |
| Answer Mode | editorial mode | Search agent mode for questions — synthesizes editorial with [N] citations |
| Search Mode | results mode | Search agent mode for discovery — curated list with relevance notes |

Protocols & Concepts

| Term | Aliases | Definition |
|--------------------|-------------------------------|---|
| AT Self-Wake | AT alarm, self-wake | Desktop Claude schedules AT jobs to self-prompt. AT = Desktop's alarm, not CLI's doorbell |
| Flag Files | state flags, lifecycle flags | Zero-byte files marking task state: *_started, *_completed, *_error, *_cancelled |
| Claim Prefix | claimed prefix | DEPRECATED v9.0 — replaced by flag files |
| Message Inserts | memory inserts, INSERT blocks | Structured <<<INSERT>>> blocks for cross-platform persistence markers |
| Reference Pointers | REF:, file pointers | REF:path/to/file.yml syntax for lazy loading, reduced context |
| Context Files | context_files list | REF: pointers in role.yml defining files to load for that role |
| role.yml | role definition | Role config with context_files, duties, ownership. At prompts/roles/{role}/ |
| Time Loop | UI reset | Claude Desktop phenomenon — response fails mid-generation, UI resets context |

Workflows

| Term | Aliases | Definition |
|------------------------|----------------------|--|
| Condensed Chat History | condensed chat | Semantic summary created ON-DEMAND, not pre-stored. ~80% reduction |
| Chat Recovery | broken chat recovery | Auto-recovery from context exhaustion — detect, export, condense, continue |
| Task Lifecycle | task states | staged → to_execute → in_progress → completed/error/cancelled |
| Task ID Counter | NextId, next_id.sh | Atomic hierarchical ID via mkdir spinlock |

Memory System

| Term | Definition |
|--------------|---|
| Memory Slots | 30 slots × ~200 chars each. Hold pointers to external files, not full content |
| Memory Files | Full-depth storage: user_model, communication_patterns, tool_discoveries, learnings, etc. |
| Manifest | Index file (manifest.yml) defining what each slot is for |

| | |
|-------------------------|---|
| Federated Memory | Per-AI owned memory with cross-reference capability via ai_ecosystem_manifest.yml |
| Hot/Warm/Cold | Memory tiers: Hot=loaded, Warm=on-demand, Cold=search/retrieval |
| Write-Immediate | Discipline: log observations as they occur, never batch or checkpoint |

Quick Lookup by Domain

| Domain | Key Terms |
|--------------|--|
| Architecture | AI Root, Layer Model, Bootstrap Problem |
| Platforms | Desktop Claude, Claude CLI, Codex CLI, Gemini CLI, Cline CLI |
| Roles | Librarian, Dev Lead, Custodian, Ops, Peer Review, Tester |
| Tools | Codex MCP, CLI-Agent MCP, Desktop Skills, Plugins |
| Search | Search Agent, Search Cascade, Answer Mode, Search Mode |
| Protocols | AT Self-Wake, Flag Files, Message Inserts, REF Pointers |
| Memory | Memory Slots, Memory Files, Federated Memory, Hot/Warm/Cold |
| Workflows | Condensed Chat, Chat Recovery, Task Lifecycle |