4. Agents & Minions

Autonomous task execution agents

4.1 Minion System

Minion System

Minions are **one-shot autonomous agents** (inspired by Stripe's internal system). Each minion:

1. Gets a single task
2. Runs in an isolated git worktree
3. Has full tool access (read/write files, run commands)
4. Follows a blueprint (YAML chain of steps)
5. Produces verifiable output
6. Exits

How to Use

```
# Run a minion for a specific task
node minion.js run "Fix login validation bug" --project ~/ALAI/products/Drop

# Run a minion tied to an MC task
node minion.js run "Add rate limiting" --project ~/ALAI/products/Drop --mc-task 1234

# Queue for batch processing
node minion.js enqueue "Refactor auth module" --project ~/ALAI/products/Drop
```

Minion Execution Flow

```
Task + Project
     |
     ▼
1. Create isolated git worktree (branch: minion/<id>)
2. Load context (project README, relevant files, HiveMind knowledge)
3. Select blueprint based on task type
4. Execute blueprint steps:
   |
```

```
    ├── Step 1: Analyze (understand the codebase)
    ├── Step 2: Plan (create execution plan)
    ├── Step 3: Build (implement changes)
    ├── Step 4: Test (run tests, verify)
    └── Step 5: Report (write GOTCHA summary)
    │
5. Quality gate — verify output, run tests
6. Return result (success/failure + artifacts)
```

Blueprint Types

Blueprint	Purpose
`minion-one-shot.yaml`	General purpose — analyze, plan, build, test
`minion-bugfix.yaml`	Bug fixing — reproduce, diagnose, fix, verify
`minion-refactor.yaml`	Refactoring — understand, plan, refactor, test
`minion-security-fix.yaml`	Security fixes — audit, fix, verify
`minion-docs.yaml`	Documentation — read code, generate docs
`codecraft-nextjs-app.yaml`	Full Next.js app scaffold (CodeCraft)
`codecraft-api-backend.yaml`	API backend scaffold (CodeCraft)
`securion-security-review.yaml`	Security audit chain (Securion)

Git Worktree Isolation

Each minion runs in its own git branch/worktree:

- **Branch:** `minion/<short-id>`
- **Path:** `~/system/.claude/worktrees/minion-<id>`
- Changes don't affect main branch until merged
- Multiple minions can work in parallel on different branches

4.2 HiveMind — Shared Knowledge

HiveMind — Shared Knowledge Base

HiveMind (`~/system/agents/hivemind/hivemind.js`) is the **collective memory** of all agents.

What Goes Into HiveMind

- **Task completions** — every pi-orchestrator task result (Tier 2+)
- **Agent discoveries** — facts, patterns, warnings found during work
- **Session summaries** — condensed logs of work sessions
- **Manual knowledge** — docs, runbooks, architecture decisions

How It Works

```
Agent completes task
       │
       ▼
HiveMind stores entry:
  - Text (up to 5000 chars)
  - Source (which agent/engine)
  - Type (knowledge, alert, briefing, update)
  - Timestamp
       │
       ▼
BGE-M3 embeds the entry → Qdrant vector DB
       │
       ▼
Future agents query HiveMind:
```

```
  "What do we know about Drop's payment system?"
       |
       ▼
Semantic search → relevant entries returned as context
```

Usage

```
# Query knowledge
node hivemind.js query "How does Drop handle PSD2 consent?"

# Store new knowledge
node hivemind.js post my-agent knowledge "Drop uses SCA for all transactions over 30 EUR"

# Search by tag
node hivemind.js search --type alert --since 24h
```

Why It Matters

Without HiveMind, every agent starts from zero. With HiveMind:

- Agents learn from previous work
- Duplicate investigation is avoided
- Institutional knowledge persists across sessions
- Quality improves over time (flywheel effect)