

# CVC Advancement Roadmap

119 Recommendations to Surpass Hermes Agent

Compiled by Sofia — Master Orchestrator

**Goal:** Make CVC's Python agentic core more advanced than Hermes Agent.

**Approach:** Audit Hermes repo → identify every subsystem → port to CVC.

**Implementation:** Done later with Tina (subagent teams).

## ■ KEY

- [P0] = Must-have (core agentic loop, providers, auth)
- [P1] = High-value (skills, memory, tool guardrails)
- [P2] = Advanced (insights, curator, kanban, MCP)
- [P3] = Nice-to-have (TTS, voice, platform transports)

## CATEGORY 1: MODEL PROVIDERS & ZERO-COST COPILOT (P0)

### 1.1 — Copilot OAuth Device Code Flow [P0]

Hermes implements the full GitHub OAuth device code flow (`client_id: Ov23li8tweQw6odWQebz`) — same as `opencode/Copilot CLI`. This is the backbone of "zero but feels like unlimited" model usage. CVC needs:

- `cvc/providers/copilot_auth.py` with `resolve_copilot_token()`, `copilot_device_code_login()`, `_try_gh_cli_token()`
- Token validation rejecting `ghp_*` classic PATs, accepting `gho_*`, `github_pat_*`, `ghu_*`
- Env var priority: `COPILOT_GITHUB_TOKEN` → `GH_TOKEN` → `GITHUB_TOKEN` → `gh auth token`
- Token exchange hitting `https://api.githubcopilot.com` with proper OAuth polling

### 1.2 — Copilot Provider Profile with Editor Attribution Headers [P0]

The Copilot profile in Hermes adds editor-version headers that make GitHub attribute usage differently (toward your Copilot quota, not API billing):

- `build_api_kwargs_extras()` injecting `X-Editor-Version`, `X-Editor-Plugin-Version` headers
- Per-model `api_mode` routing: `GPT-5+/Codex` → `codex_responses`, `Claude` → `anthropic_messages`, `rest` → `chat_completions`

### 1.3 — Provider Profile Registry (Declarative) [P0]

Hermes moved from 20+ boolean flags to declarative `ProviderProfile` dataclasses. Each provider is one object declaring: `name`, `aliases`, `env_vars`, `base_url`, `auth_type`, `fallback_models`, `fixed_temperature`, `default_max_tokens`, hooks for `build_api_kwargs_extras()`, `prepare_messages()`, `fetch_models()`.

- `cvc/providers/base.py` — `ProviderProfile` dataclass with all fields
- `cvc/providers/__init__.py` — `register_provider()` global registry

- Eliminates giant if/else chains in the agent loop

## 1.4 — Multi-Provider Fallback Chain [P0]

The agent should try providers in order on failure: Copilot (primary, \$0) → NVIDIA NIM (free tier, \$0) → Gemini → fallback models.

- Config-driven provider priority list
- Each entry has its own `api_key`, `base_url`, `model`
- Models defined as `{provider}/{model_name}` e.g. `copilot/claude-sonnet-4.6`

## 1.5 — Credential Pool with Auto-Rotation [P0]

Hermes's `CredentialPool` is the most important subsystem for "feels like unlimited":

- Multiple API keys per provider (e.g. 3 Copilot tokens)
- Persistent pool stored in `~/.cvc/credential_pool.json`
- Auto-rotation on 429/rate-limit: marks entry exhausted, picks next
- Two strategies: `fill_first` (drain one key fully before next) vs `round_robin`
- Thread-safe with locking
- Auto-refresh for OAuth tokens (Codex, Anthropic, Nous)
- `PooledCredential` dataclass: `api_key`, `base_url`, `status`, `source`, `exhausted_until`, `priority`

## 1.6 — Model Normalization Layer [P0]

`normalize_model_for_provider()` — handles vendor prefixes, hyphens vs dots, DeepSeek aliases. CVC needs identical logic so `claude-sonnet-4.6` on Copilot automatically becomes `claude/sonnet-4.6` if needed.

## 1.7 — API Mode Routing [P0]

Support multiple API response formats from a single agent:

- `chat_completions` (OpenAI standard)
- `codex_responses` (GPT-5+/Codex models — different response shape)
- `anthropic_messages` (native Anthropic format — `tool_use` blocks)
- Auto-detection from provider profile + model name

## 1.8 — Reasoning/Thinking Configuration [P1]

GitHub Models support `extra_body: {reasoning: {effort: "medium"}}` for models that support it. CVC should:

- Auto-detect which models support reasoning (catalog lookup)
- Allow `reasoning_config: {effort: "none" | "low" | "medium" | "high" | "xhigh"}`
- Normalize `xhigh` → `high` for providers that don't support it

## 1.9 — Prompt Caching (System+3 Strategy) [P1]

Anthropic gives ~75% token discount on cached prefix. Hermes uses 4 breakpoints:

- System prompt (stable)
- Last 3 non-system messages (rolling window)
- `apply_anthropic_cache_control()` pure function adding `cache_control` markers
- Applies to both native Anthropic and Copilot-wrapped Claude

## 1.10 — Cross-Session Rate Limit Guard [P1]

`nous_rate_guard.py` — writes rate-limit state to a shared file so ALL sessions (CLI, gateway, cron) can check before retrying. CVC needs this for NVIDIA NIM free tier management:

- Shared state file: `~/.cvc/rate_limits/{provider}.json`
  - Prevents retry amplification (each 429 triggering 9 more API calls)
- 

## CATEGORY 2: AGENTIC LOOP (P0)

### 2.1 — Iteration Budget with Thread Safety [P0]

`IterationBudget` — thread-safe counter, parent creates → children inherit, `execute_code` turns refunded:

- `consume()` → returns False if exhausted
- `refund()` → give back one iteration
- Parent max: 90, subagent max: 50 (configurable)
- Shared budget means parent + subagents share one pool

### 2.2 — Parallel Tool Execution [P0]

Tool calls that are safe to run concurrently are dispatched in parallel (up to 8 worker threads):

- `_PARALLEL_SAFE_TOOLS` set: `read_file`, `search_files`, `web_search`, `session_search`, `skill_view`, `vision_analyze`, etc.
- `_NEVER_PARALLEL_TOOLS`: `clarify` (interactive)
- `_PATH_SCOPED_TOOLS`: file tools run concurrently when targeting different directories
- Conflict detection: if two parallel calls touch same path → sequential fallback

### 2.3 — Error Classification & Smart Failover [P0]

`ErrorClassifier` — structured taxonomy replacing scattered string matching:

- `FailoverReason` enum: `auth`, `auth_permanent`, `billing`, `rate_limit`, `overloaded`, `server_error`, `timeout`, `context_overflow`, `payload_too_large`, `image_too_large`, `model_not_found`, `provider_policy_blocked`
- Each classification maps to a recovery action: retry, rotate credential, fallback provider, compress context, abort
- HTTP status + response body → structured `Classification` dataclass

### 2.4 — Jittered Backoff Retry [P0]

Replace fixed exponential backoff with jittered delays to prevent thundering-herd:

- `jittered_backoff(attempt, base_delay=5.0, max_delay=120.0, jitter_ratio=0.5)`
- Thread-safe monotonic counter for seed uniqueness

### 2.5 — Context Window Compression [P0]

`ContextCompressor` — automatic summarization when context approaches model limit:

- Smart mode: preserves decisions/code/architecture, summarizes routine conversation
- Target ratio: 50% (configurable)
- `should_compress(prompt_tokens)` → triggers auto-compress
- Summarization uses auxiliary (cheap) model
- Tool result pruning: old tool results truncated to summary
- Full message structure preservation (`tool_call_id` pairing)

## 2.6 — Trajectory Saving (JSONL) [P1]

Optionally save full conversation trajectories to JSONL files for training/analysis:

- Turn on with `save_trajectories=True`
- Each turn = one JSONL line with messages, tool calls, results
- Token usage metadata per turn

## 2.7 — Tool Call Guardrails [P1]

`ToolCallGuardrailController` — per-turn observation tracking:

- Detect repeated identical tool calls (loop prevention)
- Idempotent tool whitelist (safe to retry)
- Configurable max identical calls per turn
- Returns `ToolGuardrailDecision`: allow / warn / halt

## 2.8 — Destructive Command Detection [P1]

Regex-based detection of dangerous terminal commands before execution:

- Patterns: `rm`, `rmdir`, `git reset`, `git clean`, `dd`, `shred`, `sed -i`, `truncate`
- Overwrite redirect detection (`>` not `>>`)
- Used to gate terminal tool execution

## 2.9 — Tool Output Size Limits [P1]

`tool_output_limits.py` — prevents context flooding:

- Max output size per tool (configurable)
- Truncation with "[truncated X chars]" notice
- Separate limits for file content, terminal output, web results

## 2.10 — Structured JSON Repair for Tool Arguments [P1]

`_repair_tool_call_arguments()` — models frequently produce malformed JSON in tool call args. Hermes has a sophisticated repair pipeline:

- Unescape escaped quotes embedded in string values
- Handle truncated JSON (missing closing braces)
- Handle nested quoting corruption
- Apply `json.loads` with `strict=False` as final fallback

## 2.11 — Unicode/Surrogate Sanitization [P1]

`_sanitize_messages_surrogates()` — models (especially streaming) can produce invalid surrogate pairs:

- Detect and replace invalid surrogates in messages, tool calls, tool results
- Non-ASCII stripping mode for providers that reject it (e.g. some Chinese providers)

## 2.12 — Multimodal Tool Result Handling [P1]

`_is_multimodal_tool_result()`, `_multimodal_text_summary()` — handle tool results that contain images:

- Detect image blocks in tool output
- Convert to text summary for non-vision models
- Cache MCP image blocks to disk

## 2.13 — Thinking/Reasoning Block Scrubbing [P1]

`ThinkScrubber` — stateful streaming scrubber for `<think>` blocks:

- State machine handles partial tags across stream deltas
- Prevents reasoning leakage to CLI/gateway consumers
- Works at the upstream layer so all consumers get clean text

## 2.14 — Secret Redaction [P1]

`redact.py` — pattern-based secret masking in logs and tool output:

- API keys, tokens, credentials masked before logging
- Short tokens (< 18 chars): fully masked
- Long tokens: first 6 + last 4 chars preserved for debuggability
- Sensitive query params stripped from URLs

## 2.15 — Prefill Messages Support [P1]

Allow prepending few-shot examples or priming messages:

- `prefill_messages`: `[{"role": "user", "content": "Hi!"}, {"role": "assistant", "content": "Hello!"}]`
- Validation: reject trailing-assistant prefill for Anthropic Sonnet/Opus 4.6+

## 2.16 — Service Tier Support [P1]

OpenRouter `service_tier` parameter for priority vs standard routing:

- `service_tier`: `"priority"` → higher cost, faster
- `service_tier`: `"auto"` → let OpenRouter decide

## 2.17 — Request Overrides [P1]

`request_overrides`: `Dict[str, Any]` — per-request parameter injection:

- Provider-specific parameters not in standard schema
- Example: `{"extra_body": {"google": {"thought_tag_marker": "think"}}}`

## 2.18 — Streaming Delta Callbacks [P1]

Pluggable callbacks for streaming:

- `stream_delta_callback(delta)` — per-token
- `interim_assistant_callback(text)` — per-message
- `thinking_callback(text)` — reasoning stream
- `status_callback(status)` — agent status changes

## 2.19 — Platform-Aware System Prompts [P1]

Inject platform-specific formatting hints:

- `platform`: `"cli"` → full ANSI, tables, rich formatting
- `platform`: `"telegram"` → markdown, no tables, spoiler support
- `platform`: `"discord"` → markdown, code blocks
- Auto-detected from gateway session context

## 2.20 — Safe Stdio Installation [P1]

`_install_safe_stdio()` — UTF-8 stdio on Windows, no-op on POSIX:

- Must be called before any other I/O
  - Handles Windows console encoding edge cases
- 

## CATEGORY 3: SKILLS SYSTEM (P1)

### 3.1 — Skill Manifest & Auto-Discovery [P1]

`build_skills_system_prompt()` — scans `~/ .cvc/skills/` directory:

- Parses YAML frontmatter from each `SKILL.md`
- Builds compact manifest injected into system prompt
- Snapshot caching: only re-scan when files change (mtime check)
- Skills sorted by relevance to current task

### 3.2 — Skill Usage Tracking [P1]

`skill_usage.py` — persistent usage analytics per skill:

- `bump_view(skill_name)` — skill was shown in manifest
- `bump_use(skill_name)` — skill was actually loaded
- `bump_patch(skill_name)` — skill was modified
- `activity_count()`, `latest_activity_at()` — for curator decisions
- Stored in `~/ .cvc/skill_usage.json`

### 3.3 — Skill Lifecycle States [P1]

Skills transition through states automatically:

- `active` → `stale` (30 days no use) → `archived` (90 days no use)
- Pinned skills bypass all auto-transitions
- Only agent-created skills are auto-curated (user-created are sacred)

### 3.4 — Skill Provenance (Write Origin Tracking) [P1]

`skill_provenance.py` — `ContextVar` distinguishing:

- `"foreground"` — user asked agent to write this skill (never auto-curate)
- `"background_review"` — curator fork wrote this (auto-curated)
- Prevents curator from consolidating user's skills

### 3.5 — Skill Security Scanning [P1]

`_security_scan_skill()` — pre-write validation:

- Scan for prompt injection patterns in skill content
- Block skills containing hidden directives
- Validate frontmatter structure
- Size limits (max content size per file)

### 3.6 — Skill Pinning [P1]

Pinned skills are protected from:

- Auto-archive

- Auto-consolidation
- Deletion (returns error pointing to unpin command)
- Still allow patching/editing

### 3.7 — Skill Categories [P1]

Organize skills into subdirectories:

- `skills/devops/`, `skills/mlops/`, `skills/research/`
- Category validation on creation
- Category-based filtering in manifest

### 3.8 — Skill Archive & Restore [P1]

- `archive_skill(name)` → moves to `~/.cvc/skills/.archive/`
- `restore_skill(name)` → moves back
- Archive is recoverable (not deleted)
- `list_archived_skill_names()` → browse archived

### 3.9 — Skill Atomic Writes [P1]

`_atomic_write_text()` — write to temp file + rename:

- Prevents corruption on crash mid-write
- Preserves file permissions
- UTF-8 enforced

### 3.10 — Skill Validation [P1]

`_validate_frontmatter()` — YAML frontmatter schema validation:

- Required fields: `name`, `description`
- Optional: `category`, `triggers`, `version`
- Reject malformed YAML

### 3.11 — Skill Hub Integration [P1]

`skills_hub.py` — browse and install skills from remote hub:

- List available skills from hub
- Install by name
- Update installed skills
- Track hub-installed vs local

### 3.12 — Skill Guard [P1]

`skills_guard.py` — rate-limit skill loading:

- Max N skills loaded per turn
- Prevent context flooding from too many skill bodies
- Priority: skills matching current task first

---

## CATEGORY 4: MEMORY & SELF-IMPROVEMENT (P1)

## 4.1 — Persistent Memory Store [P1]

`MemoryStore` class — append-only memory files:

- `~/.cvc/memory/` directory
- `MEMORY.md` — always-injected context (user prefs, environment facts)
- `USER.md` — user profile
- `memory_tool()` — add/replace/remove/search operations
- Sanitize context before storing (no secrets)

## 4.2 — Holographic Memory (HRR) [P2]

`plugins/memory/holographic/` — Vector Symbolic Architecture:

- Phase-vector encoding (SHA-256 deterministic)
- Bind/unbind/bundle operations for compositional structure
- Cross-session semantic search via cosine similarity
- Entity resolution (Jai → user → principal)
- Trust scoring per fact (helpful/unhelpful feedback)
- `fact_store` with `probe()`, `reason()`, `contradict()` queries

## 4.3 — Session Search [P1]

`session_search_tool.py` — SQLite-backed session history:

- Full-text search across all past sessions
- `list_recent_sessions(limit=N)` — browse recent
- `session_search(query, limit=N)` — keyword/semantic search
- Truncate around matches for relevance
- Max concurrency control (prevent DB contention)

## 4.4 — Curator (Background Self-Improvement) [P2]

`Curator` — background skill maintenance orchestrator:

- Runs when agent is idle (no cron daemon needed)
- Spawns forked AI Agent with auxiliary model
- Reviews agent-created skills: pin/archive/consolidate/patch
- Strict: only touches agent-created skills, never auto-deletes
- State persisted in `~/.cvc/.curator_state`
- Configurable: `interval_hours`, `min_idle_hours`, `stale_after_days`, `archive_after_days`

## 4.5 — Checkpoint Manager [P2]

`CheckpointManager` — git-based state snapshots:

- Auto-commit working directory state before dangerous operations
- Shadow repo per project (stored in `~/.cvc/checkpoints/`)
- `create_checkpoint(label)` → snapshot
- `restore_checkpoint(hash)` → rollback
- Max snapshots + total size limits
- Project registration and listing

## 4.6 — Context File Auto-Injection [P1]

`build_context_files_prompt()` — auto-load project context:



- `SOUL.md` — agent identity (always loaded)
- `AGENTS.md` — project instructions
- `CLAUDE.md` — Claude-specific instructions
- `.cursorrules` — Cursor IDE rules
- Git root detection (walk up from cwd)
- Skip option for batch processing

## 4.7 — Memory Context Block Building [P1]

`build_memory_context_block()` — format memory for injection:

- Scans raw context for sensitive patterns
- Formats as compact preamble
- Token-budget aware truncation

## 4.8 — Streaming Context Scrubber [P1]

`StreamingContextScrubber` — real-time PII scrubbing:

- Scrub secrets from streaming output
- Pattern matching for API keys, tokens
- Operates on delta stream (not post-hoc)

---

# CATEGORY 5: TOOLS & MCP (P1)

## 5.1 — MCP Client (Native) [P1]

`mcp_tool.py` — full MCP protocol support:

- stdio and HTTP transports
- Tool registration from MCP servers
- OAuth flow for MCP servers (`mcp_oauth.py`)
- Server lifecycle management (start/stop/restart)
- Error handling: auth errors, session expired, connection lost
- Image caching from MCP tool results
- Sampling handler (MCP server can request LLM sampling)
- Description security scanning (reject malicious tool descriptions)

## 5.2 — MCP OAuth Manager [P1]

`mcp_oauth_manager.py` — OAuth for MCP servers:

- Token storage per server
- Auto-refresh before expiry
- Consent prompt on first use

## 5.3 — Tool Registry [P1]

`tools/registry.py` — centralized tool registration:

- `register_tool(name, fn, schema)` — add tool
- `get_tool(name)` — retrieve
- `list_tools()` — all registered

- Schema validation on registration

## 5.4 — Toolset System [P1]

`toolsets.py` — group tools into sets:

- `enabled_toolsets`: `["web", "file", "terminal"]` — only load these
- `disabled_toolsets`: `["discord"]` — exclude these
- Per-toolset: tool names, description, default enabled
- Reduces context window (only relevant tool schemas in prompt)

## 5.5 — File Safety Rules [P1]

`file_safety.py` — shared safety rules:

- Deny writes to: `.ssh/authorized_keys`, `.ssh/id_rsa`, `.ssh/config`, `~/.hermes/.env`, `.bashrc`
- Deny reads to sensitive paths
- Home directory resolution (profile-aware)

## 5.6 — Path Security [P1]

`path_security.py` — prevent path traversal:

- Resolve all paths to real paths
- Block writes outside allowed directories
- Symlink resolution before access check

## 5.7 — File State Tracking [P1]

`file_state.py` — track file modifications:

- Per-task file read/write cache
- Detect conflicting parallel writes
- Snapshot before write for rollback

## 5.8 — Process Registry [P1]

`process_registry.py` — manage background processes:

- `list`, `poll`, `log`, `wait`, `kill`, `write`, `submit`, `close`
- Session-based process management
- Notify on completion

## 5.9 — Code Execution Tool [P1]

`code_execution_tool.py` — sandboxed Python execution:

- `execute_code(code)` — run Python with tool access
- 5-minute timeout, 50KB stdout cap
- Iteration refund (doesn't count against budget)
- Hermes tools available via `from hermes_tools import ...`

## 5.10 — Mixture of Agents [P2]

`mixture_of_agents_tool.py` — route hard problems through multiple LLMs:

- 4 reference models + 1 aggregator
- Maximum reasoning effort

- Best for: complex math, algorithms, multi-step reasoning

## 5.11 — Web Tools [P1]

`web_tools.py` — search + extract:

- `web_search(query)` — search the web
- `web_extract(url)` — extract readable content from URL
- URL safety checking (`url_safety.py`)
- Provider abstraction (Serper, Perplexity, etc.)

## 5.12 — Vision Tools [P1]

`vision_tools.py` — image analysis:

- `vision_analyze(image_url, question)` — analyze image
- Support for local files, URLs, data: URLs
- Provider abstraction (native vision vs auxiliary model)

## 5.13 — Image Generation [P2]

`image_generation_tool.py` — text-to-image:

- Provider abstraction (FAL, OpenAI, etc.)
- Aspect ratio control
- Returns URL or local file path

## 5.14 — Send Message Tool [P1]

`send_message_tool.py` — cross-platform messaging:

- Platform abstraction: Telegram, Discord, Slack, Signal
- Target resolution (channel, user, thread)
- Media attachment support

## 5.15 — Cron Job Tools [P1]

`cronjob_tools.py` — scheduled task management:

- `create, list, update, pause, resume, remove, run`
- Schedule parsing (ISO, cron expressions, relative)
- Delivery targets (origin, local, all, specific channel)
- Script mode (no-agent watchdog pattern)

## 5.16 — Todo Tools [P1]

`todo_tool.py` — task list management:

- `read, write(merge=True/False)`
- Status: pending, in\_progress, completed, cancelled
- Only one in\_progress at a time

## 5.17 — Patch Parser [P1]

`patch_parser.py` — V4A multi-file patch format:

- `*** Begin Patch / *** End Patch`
- `*** Update File: path`

- Context-based fuzzy matching
- Replace-all mode

### 5.18 — Fuzzy Match [P1]

`fuzzy_match.py` — file content matching:

- 9 strategies for finding text in files
- Tolerates whitespace/indentation differences
- Used by patch tool

### 5.19 — Delegate Tool (Subagent Architecture) [P1]

`delegate_tool.py` — spawn child agents:

- Single task or batch (parallel, up to 3)
- Isolated context + terminal session
- Restricted toolset (blocked: `delegate`, `clarify`, `memory`, `send_message`, `execute_code`)
- Parent blocks until children complete
- Results returned as summaries (not full context)

### 5.20 — Kanban Tools [P2]

`kanban_tools.py` — structured task management for worker agents:

- Only registered when `HERMES_KANBAN_TASK` env var set
- Tools: `kanban_claim`, `kanban_complete`, `kanban_fail`, `kanban_list`
- SQLite backend at `~/cvc/kanban.db`
- Metadata passing via JSON

## CATEGORY 6: CONTEXT & PROMPT ENGINEERING (P1)

### 6.1 — Prompt Builder [P1]

`prompt_builder.py` — modular system prompt construction:

- `build_environment_hints()` — OS, hostname, timezone, git info
- `build_skills_system_prompt()` — skills manifest
- `build_context_files_prompt()` — SOUL.md, AGENTS.md, etc.
- `build_nous_subscription_prompt()` — subscription status
- `load_soul_md()` — identity loading
- Each component is independently cacheable

### 6.2 — Skills System Prompt Caching [P1]

`_write_skills_snapshot()` / `_load_skills_snapshot()`:

- Cache parsed skills manifest to disk
- Only re-parse when SKILL.md files change (mtime)
- Eliminates redundant parsing every turn

### 6.3 — Context File Truncation [P1]

`_truncate_content()` — max chars per context file:

- Default: 2000 chars per file
- Preserves beginning (most important) + end (most recent)
- Configurable per-file-type limits

## 6.4 — Environment Hints [P1]

`build_environment_hints()` — inject runtime context:

- OS, hostname, username
- Current timezone + time
- Git branch + status
- Remote backend detection (Docker, Modal, SSH)

## 6.5 — Nous Subscription Prompt [P1]

`build_nous_subscription_prompt()` — show subscription status:

- Remaining requests
- Model access level
- Only injected for Nous Portal provider

## 6.6 — Context References (@-mentions) [P2]

`context_references.py` — inline file/folder/git references:

- `@file:"path/to/file"` — inline file content
- `@folder:"path"` — list folder contents
- `@git:diff` — git diff
- `@git:staged` — staged changes
- `@url:"https://..."` — fetch URL content
- Sensitive path blocking (.ssh, .aws, etc.)

---

# CATEGORY 7: CONFIGURATION & CLI (P1)

## 7.1 — Config YAML [P1]

`hermes_cli/config.py` — hierarchical configuration:

- `~/.cvc/config.yaml` — global
- `.cvc/config.yaml` — project-local (overrides)
- Profile support (`--profile work`)
- Env var substitution (`${HOME}`, `${COPILOT_TOKEN}`)
- Hot-reload on change

## 7.2 — Model Catalog [P1]

`model_catalog.py` — curated model list:

- Fetch from remote manifest URL
- Disk cache with TTL
- Provider-specific overrides
- Curated OpenRouter models list
- Curated Nous models list

- Force refresh option

### 7.3 — Model Switch Command [P1]

`model_switch.py` — interactive model picker:

- List available models per provider
- Show current model
- Switch provider + model in one command
- Persist choice to config

### 7.4 — Auth Commands [P1]

`auth_commands.py` — provider authentication:

- `cvc auth copilot` — OAuth device flow
- `cvc auth nvidia` — API key setup
- `cvc auth status` — show all provider auth status
- Token validation on entry

### 7.5 — Doctor Command [P1]

`doctor.py` — system health check:

- Check all provider connectivity
- Verify auth tokens
- Check disk space
- Check skill directory integrity
- Report: ■ OK / ■■ Warning / ■ Critical

### 7.6 — Backup & Rollback [P1]

`backup.py` — config and state backup:

- `cvc backup create` — timestamped backup
- `cvc backup list` — list backups
- `cvc backup restore <timestamp>` — rollback
- Includes: config, skills, memory, credential pool

### 7.7 — Profile Distribution [P1]

`profile_distribution.py` — toolset profiles:

- Predefined toolset combinations per use case
- `coding`: file + terminal + web + search
- `research`: web + search + session\_search + file
- `ops`: terminal + file + cron + delegate

### 7.8 — CLI Completion [P1]

`completion.py` — shell completion:

- Bash, Zsh, Fish
- Dynamic completion for models, skills, providers
- Auto-generated from registry

## 7.9 — Logs Command [P1]

`logs.py` — log viewing:

- `cvc logs` — recent agent logs
- `cvc logs --follow` — tail
- `cvc logs --session <id>` — specific session
- Filtering by level, tool, provider

## 7.10 — Dump Command [P1]

`dump.py` — state inspection:

- `cvc dump config` — show effective config
  - `cvc dump skills` — show skill manifest
  - `cvc dump providers` — show provider status
  - `cvc dump memory` — show memory files
- 

# CATEGORY 8: GATEWAY & TRANSPORTS (P2)

## 8.1 — Gateway Server [P2]

`gateway/` — WebSocket gateway for UI connections:

- Session management (per-chat keys)
- Message routing (inbound → agent → outbound)
- Hook system (pre/post message)
- Channel directory (Telegram, Discord, Slack)
- Delivery queue with retry

## 8.2 — Transport Abstraction [P2]

`agent/transport/` — provider-specific API adapters:

- `base.py` — abstract transport
- `chat_completions.py` — OpenAI-compatible
- `anthropic.py` — native Anthropic
- `codex.py` — Codex responses format
- `bedrock.py` — AWS Bedrock
- `types.py` — shared type definitions

## 8.3 — Shell Hooks [P2]

`shell_hooks.py` — event-driven shell script execution:

- Config-driven hooks: `on_session_start`, `on_session_end`, `on_tool_call`
- First-use consent per `(event, command)` pair
- Allowlist in `~/.cvc/shell-hooks-allowlist.json`
- `shlex.split()` + `shell=False` (no injection)

## 8.4 — Plugin System [P2]

`plugins/` — discoverable Python plugins:

- Auto-discover from `cvc/plugins/`

- Plugin YAML manifest ([plugin.yaml](#))
- Hook registration via [invoke\\_hook\(\)](#)
- Plugin commands via [register\\_command\(\)](#)
- Priority: Python plugins win ties over shell hooks

## 8.5 — Platform Adapters [P2]

[hermes\\_cli/platforms.py](#) — platform-specific behavior:

- Formatting rules per platform
  - Message size limits
  - Media handling
  - Command prefix handling
- 

# CATEGORY 9: INSIGHTS & ANALYTICS (P2)

## 9.1 — Usage Insights Engine [P2]

[agent/insights.py](#) — comprehensive usage analytics:

- Token consumption per session/day/week
- Cost estimates per provider
- Tool usage frequency
- Model/platform breakdown
- Activity trends over time
- Inspired by Claude Code's [/insights](#)

## 9.2 — Usage Pricing Database [P2]

[agent/usage\\_pricing.py](#) — model pricing:

- [CanonicalUsage](#) dataclass (input/output/cache tokens)
- [PricingEntry](#) per model (per-token costs)
- [estimate\\_usage\\_cost\(\)](#) — cost per turn
- [has\\_known\\_pricing\(\)](#) — check if model has pricing data
- Official docs pricing + OpenRouter pricing fallback

## 9.3 — Rate Limit Display [P2]

[rate\\_limit\\_tracker.py](#) — visual rate limit status:

- Parse [X-RateLimit-\\*](#) headers
  - [RateLimitBucket](#) (requests, tokens, reset time)
  - [format\\_rate\\_limit\\_display\(\)](#) — full bar chart
  - [format\\_rate\\_limit\\_compact\(\)](#) — one-line summary
- 

# CATEGORY 10: VOICE & TTS (P3)

## 10.1 — TTS Tool [P3]

[tts\\_tool.py](#) — text-to-speech:



- Provider abstraction (Edge, OpenAI, xAI, MiniMax, ElevenLabs)
- Character caps enforced per provider
- Output to file or stream
- Voice selection

## 10.2 — Voice Mode [P3]

`voice_mode.py` — interactive voice session:

- Record → transcribe → agent → TTS response
- Push-to-talk or continuous
- Interrupt handling

## 10.3 — Transcription [P3]

`transcription_tools.py` — audio transcription:

- Whisper local or API
  - File + microphone input
  - Timestamp output
- 

# CATEGORY 11: SECURITY & SAFETY (P1)

## 11.1 — OSV Security Check [P1]

`osv_check.py` — vulnerability scanning:

- Check installed packages against OSV database
- Pre-install scan for new dependencies
- Report: CVE ID, severity, fixed version

## 11.2 — Tirith Security [P1]

`tirith_security.py` — content safety:

- Pre-send content scanning
- Block harmful output
- Configurable safety levels

## 11.3 — Approval System [P1]

`approval.py` — destructive operation approval:

- Per-operation-type approval policies
- Allowlist for known-safe commands
- Interactive approval prompt
- Auto-approve mode for trusted environments

## 11.4 — Credential Files [P1]

`credential_files.py` — secure credential storage:

- Encrypted at rest
- Per-provider credential files
- Permission enforcement (600)

---

---

## CATEGORY 12: BATCH & AUTOMATION (P2)

### 12.1 — Batch Runner [P2]

`batch_runner.py` — parallel batch processing:

- Load dataset from JSONL
- Process N items in parallel
- Checkpointing for fault tolerance
- Trajectory saving
- Tool usage statistics aggregation

### 12.2 — RL Training Tool [P2]

`rl_training_tool.py` — reinforcement learning:

- GRPO/DPO data generation
- Reward model scoring
- Training data export

### 12.3 — Environment Runner [P2]

`environments/agentive_opd_env.py` — agentive environment:

- Gym-like interface for agent tasks
  - Step/reset/observation/reward
  - Custom environment registration
- 
- 

## CATEGORY 13: DEVELOPER EXPERIENCE (P2)

### 13.1 — Debug Helpers [P2]

`debug_helpers.py` — debugging utilities:

- `cvc debug session <id>` — inspect session state
- `cvc debug tool <name>` — test tool directly
- `cvc debug provider <name>` — test provider connectivity

### 13.2 — Curses UI [P2]

`curses_ui.py` — terminal UI:

- Full-screen terminal interface
- Split panes (chat + tools + status)
- Keyboard navigation
- Color support

### 13.3 — Banner [P2]

`banner.py` — startup banner:

- ASCII art version display
- Provider status

- Quick tips
- System info

### 13.4 — Clipboard Integration [P2]

`clipboard.py` — system clipboard:

- Copy agent output to clipboard
- Paste from clipboard as input
- Cross-platform (macOS, Linux, Windows)

### 13.5 — PTY Bridge [P2]

`pty_bridge.py` — pseudo-terminal for interactive CLI:

- Run interactive tools (Codex, Claude Code, Python REPL)
  - Full terminal emulation
  - Input/output forwarding
- 

## CATEGORY 14: CVC-SPECIFIC ENHANCEMENTS (BEYOND HERMES)

These are features CVC should have that Hermes doesn't:

### 14.1 — Cognitive Version Control Integration [P0]

- Native CVC commit/recall/restore as first-class agent tools
- Auto-commit before major operations
- Branch per task/squad
- Hive memory for multi-agent sharing

### 14.2 — Squad-Based Multi-Agent [P0]

- Register agents with squad/rank
- Inter-agent communication via hive
- Squad-level branch permissions
- Agent spawning with squad inheritance

### 14.3 — Python-Native Dashboard [P0]

- Replace Node/Vite dashboard with Python (Textual or Rich)
- Real-time agent visualization
- Provider/usage monitoring
- Skill browser

### 14.4 — Zero-Config Provider Auto-Detection [P1]

- Scan env vars → auto-configure providers
- Test connectivity on startup
- Suggest missing providers
- One-command setup: `cvc setup`

### 14.5 — Provider Usage Dashboard [P1]

- Real-time token/cost tracking per provider
- Visual rate limit bars
- Daily/weekly usage reports
- Cost optimization suggestions

## 14.6 — Skill Marketplace [P2]

- Browse community skills
- One-command install
- Rating system
- Auto-update

## 14.7 — Agent Personality System [P2]

- Configurable personality traits
- Tone adjustment (formal/casual/technical)
- Humor level
- Language preference

## 14.8 — Project Memory [P2]

- Per-project memory files
- Auto-load when entering project directory
- Project-specific skills
- Project usage analytics

## 14.9 — Time-Aware Scheduling [P2]

- Agent knows current time + timezone
- Schedule tasks for specific times
- "Remind me in 20 minutes"
- Calendar integration

## 14.10 — Learning Mode [P2]

- Agent observes user corrections
- Builds preference model
- Auto-applies learned patterns
- Explicit "remember this" trigger

---

# SUMMARY COUNT

Category | Count | Priority

1. Model Providers & Copilot | 10 | P0-P1
2. Agentic Loop | 20 | P0-P1
3. Skills System | 12 | P1
4. Memory & Self-Improvement | 8 | P1-P2
5. Tools & MCP | 20 | P1-P2
6. Context & Prompt Engineering | 6 | P1-P2
7. Configuration & CLI | 10 | P1
8. Gateway & Transports | 5 | P2

- 9. Insights & Analytics | 3 | P2
- 10. Voice & TTS | 3 | P3
- 11. Security & Safety | 4 | P1
- 12. Batch & Automation | 3 | P2
- 13. Developer Experience | 5 | P2
- 14. CVC-Specific Enhancements | 10 | P0-P2
- \*\*TOTAL\*\* | \*\*119\*\*

---

## RECOMMENDED IMPLEMENTATION ORDER

- Phase 1 — Core Agentic Engine (Week 1-2):*
- Provider registry → Copilot auth → Credential pool → Agentic loop → Error classification → Context compression
- Phase 2 — Skills & Memory (Week 3):*
- Skill system → Usage tracking → Memory store → Session search → Context file injection
- Phase 3 — Tools & Safety (Week 4):*
- MCP client → Tool registry → Toolset system → File safety → Tool guardrails → Delegate tool
- Phase 4 — CLI & Config (Week 5):*
- Config system → Auth commands → Doctor → Model switch → Backup/rollback
- Phase 5 — Advanced (Week 6+):*
- Curator → Checkpoints → Insights → Kanban → Batch runner → CVC-specific features

---

*Document generated by Sofia after deep audit of Hermes Agent repo at `/Users/jkm/.hermes/hermes-agent`*

*119 recommendations across 14 categories*

*Ready for implementation with Tina*