

ARCHON AGENTIC OS 2026 — COMPLETE FEATURE LIST

System & Module Catalog, Architecture Overview, and Feature Matrix

As of: November 11, 2026 | Version: v0.5.1 ("Archon Hardened") | Language: English

System & Module Catalog, Architecture Overview, and Feature Matrix

© Michael Keller

Table of Contents

1. Version Status & Release Overview (v0.5.1) 3
2. System Views & Dedicated Studios (UI Views) 3
3. Token Savings & FinOps Ecosystem (Cost Minimization) 8
4. Kernel, Scheduling & Daemon Architecture (Pillar 1) 10
5. Multi-Agent Process Model, Hierarchy & Synthesis (Pillar 2) 12
6. MCP Drivers, External Services & Multimodal Tools (Pillar 3) 13
7. Memory, Database Hygiene & Obsidian Second Brain (Pillar 4) 14
8. Security, Zero-Trust Governance & System Diagnostics (Pillar 6) 16
9. Self-Evolution, Marketplaces & Auto-Documentation (Pillar 7) 17
10. Reference Matrix: Controllable Evey/Hermes Plugins 18
11. Enterprise & Resilience Architecture 19
12. Autonomy, Sandbox & Governance Architecture 21
13. Performance Core, Throughput Scaling & Token Efficiency 22
14. Modern Memory & RAG Architecture 23
15. Reliable Execution & Determinism (Durable Execution) 25
16. Security, Data Privacy & Enterprise Guardrails 26
17. Self-Optimization, Evaluation & Swarm Intelligence 27
18. Pillar 3: Business & ERP (alpha) & Multi-Tenant Suite 28
19. Architecture, Security & Deployment Features (v0.5.0 – v0.5.1) 29
20. Glossary & Alphabetical Index 30

© Michael Keller

1. Version Status & Release Overview (v0.5.1)

Version	Codename	Key Architecture & Feature Milestones
v0.5.1	Archon Hardened	Modular FastAPI routers (main.py -88%), 25 critical + 68 high security findings resolved, Class A Docker admin proxy, RBAC user management (PBKDF2), 2-phase system update (?local_only=true), AnythingLLM proxy integration, portable 1-click installer.
v0.5.0	Business & ERP (alpha)	Complete Pillar 3 (CRM/PocketBase sync, items/services, quotes/orders ANG/AB, accounting with DATEV & GiroCode, corporate tenants), Research Lab v2 & 55 Boardroom frameworks.
v0.3.x	Sentinel / Foundation	7-pillar OS kernel, 70-specialist fleet, 34+ Evey plugins, FinOps 3-tier routing, ephemeral code sandboxes, replay debugging, PII scrubber, semantic cache & FTS5 hybrid search.

2. System Views & Dedicated Studios (UI Views)

1. View: Mission Control (Monitoring Cockpit & Control Station)

Problem: Without a central real-time cockpit, visibility into active agents, token throughput, daily spend, and blocked approvals is missing.

Mechanics: Combines 5 vital KPI tiles (fleet load, throughput TPM, cost guard, stability, Approval Gate), the 5-column board projection, the Active Process Inspection Bar (3-slot footer for heartbeat, tool concurrency, and context saturation), the 15-minute token activity band (SVG sparkline), and the live audit stream.

Benefit: *Delivers zero-latency 360° monitoring of the entire operating system on a single screen.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

2. View: Operations – Kanban Board & Task Engine

Problem: Tasks get lost in raw chat histories and cannot be managed by status or ownership.

Mechanics: 5 lifecycle columns (To-Do, In Progress, Review & QA, Blocked, Done) directly on kanban.db (single source of truth). Supports configurable WIP limits per column, priority filters (P0–P3), agent assignment, time tracking, and 1-click Obsidian linking. Integrates the 4-eyes quality gate: Before moving a task to 'done', a peer reviewer (Katharina Völk for QM/compliance or Robert) inspects the deliverable for completeness, substance, and actionability (1–100% score). If 'changes_requested' occurs, a status lock activates; a Human-in-the-Loop override (force=True) overrides if needed.

Benefit: *Structures all operational projects transparently and prevents system overload via work-in-progress limits.* Prevents unfinished work from entering 'done' status through mandatory four-eyes review.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

3. View: Operations – Interactive 2D Org Chart & Reporting Lines

Problem: Flat lists of agents fail to show authority structures or escalation paths for blocked processes.

Mechanics: Visualizes 56 nodes and 55 reporting lines as an interactive 2D graph (Apex -> CEO Robert -> Domain Leads -> Specialists). Supports drag-and-drop reassignment of supervisors, grid snapping, team card view, and persistent storage in config/org_hierarchy.json.

Benefit: *Defines binding escalation and delegation chains that the system follows automatically when blockers arise.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the

module).

4. View: Operations – Internal Agent Communication (Radio Traffic)

Problem: When specialists hand off subtasks among themselves, it either floods user chat or remains invisible.

Mechanics: Dedicated monitoring center with channel list (#allgemein, #tech-dev, #legal, #marketing) and chronological message feed of all inter-agent coordination from kanban.db.

Benefit: *Makes agent fleet collaboration transparent without cluttering the primary user inbox.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

5. View: Inbox & Event Triage (3-Column HITL Gatekeeper & Inbox)

Problem: Uncontrolled write actions by agents (e.g., sending emails) and scattered inputs across email and Telegram present security risks.

Mechanics: Three-column triage hub: Column 1 for yellow-flagged Human-in-the-Loop approvals (Approve/Reject), Column 2 for IMAP/SMTP email synchronization with auto-triage & delegation, Column 3 for the mirrored live Telegram stream (@Beatman_bot).

Benefit: *Ensures that no external write operation executes without explicit operator approval.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

6. View: Specialist Fleet & Fleet Management (8-Tab Cockpit)
Problem: Specialists require individual models, system prompts, tools, and standard operating procedures in one centralized place.
Mechanics: Manages 70 specialist profiles with quick filters (Active, Favorites, All, 7 Domains), hot-swap for provider & model, and 8 dedicated tabs: Overview (360° Dossier & Telegram handle), Direct Chat, Skills, Operations (task history), SOUL (system prompt editor), Personnel File (SOPs), Talent Marketplace, and Skill Marketplace.
Benefit: Centralized HR department for rapid calibration and expansion of the entire agent roster.
Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

7. View: Master Chat (Dual-Mode Workspace & Multi-Session Sidebar)

Problem: Direct collaboration with agents lacked a focused full-screen environment with fast switching between 1:1 dialog, group chat, and mission dispatch.

Mechanics: Provides a dual-mode toggle ('Direct' for immediate response vs. 'Mission' for structured task creation), multi-file upload (up to 5 files), browser screen capture (camera button), voice input (dictation), reasoning budget selector (0 to 16k tokens), AI prompt optimizer, and right-hand session history (Agents vs. Group). Includes the Telegram voice pipeline: Incoming voice notes (.ogg/Opus) received on the go are transcribed via Whisper/Gemini, routed to the target specialist agent using handles (/robert, /anwalt, etc.), and responses are returned directly to Telegram as natural voice memos via edge-tts (de-DE-ConradNeural).

Benefit: *Unifies high-speed dialog, multimodal input, and mission control in a cohesive workspace.* Enables full mobile voice control and bidirectional audio exchange via Telegram.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

8. View: AI Mastermind & Boardroom (55 Methodology Frameworks)

Problem: Complex strategic decisions require structured management methodologies and contrasting perspectives rather than isolated opinions.

© Michael Keller

Mechanics: Simulates multi-stage board sessions (Chairman + 3 real specialist agent slots + Red Team Challenger) across 34 selectable frameworks in 7 categories (35 base) (e.g., CLEAR, Design Sprint, Double Diamond, Lean Startup, First Principles, Game Theory, Blue Ocean, Pre-Mortem, Six Sigma).

Benefit: Enforces methodological rigor, uncovers blind spots via the Red Team, and delivers actionable board memos ready for decision-making.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

9. View: Showroom (Project Explorer, Artifact Gallery & Python Sandbox)

Problem: Generated reports, HTML prototypes, calculations, and scripts get lost in lengthy conversations.

Mechanics: Catalogs all deliverables by creator, date, and type. Includes the Workspace Vault Reader with live rendering for Markdown/HTML/JSON and an isolated Python sandbox for secure code execution. Extended with the interactive Showroom Live-Runner: Enables direct execution of Python and code files in the browser with a 12s timeout, isolated scratch directory, and expandable terminal drawer (STDOUT, STDERR, exit code, and millisecond runtime).

Benefit: *Acts as an inspection and presentation hall for all finished deliverables before client delivery or production deployment.* Code artifacts and scripts can be verified and tested directly in the Showroom without switching consoles.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

10. View: Central Library & 2nd Brain (RAG, Auto-OCR & Obsidian Vault)
Problem: Specialist books, DIN standards, and zoning plans in PDF format could previously not be searched by agents without manual copying.
Mechanics: Unifies 3 sub-tabs and 4 core hubs (00_HR_Agenten, 02_Wissensbibliothek, Protokolle/, 00_Notebook_Synthesen): 1. AnythingLLM RAG index (Port 3001), 2. Document intake with drag-and-drop, auto-OCR (Docling), and AI topic classification, 3. Direct Obsidian file tree & Markdown viewer including vault switcher.
Benefit: Fully automatically transforms PDF appraisal reports into searchable Markdown and vector knowledge for all 70 agents.
Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

11. View: Workflow Engine (n8n Automation Mesh on Port 5678)

Problem: Recurring integration processes (cron jobs, webhooks, database reconciliations) waste tokens unnecessarily when run through LLMs.

Mechanics: Directly embeds the visual n8n workflow editor and connects it to CEO agent Robert via the n8n-automation-mesh skill (FastMCP). Activated 0-token routines: 1. Morning Briefing (weekdays at 07:30 with Obsidian archiving and Telegram delivery), 2. Weekly Maintenance (Sundays at 23:00 with SQLite backup & health audit), 3. RSS & News Watchdog (hourly scan), plus 5 active function webhooks for web extraction, messaging, document rendering, feed search, and CRM sync.

Benefit: *Combines deterministic 0-token background processes with cognitive agent decisions at key junctures.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

12. View: LLM Gateway (OmniRoute AI Router Dashboard)

Problem: Hard vendor lock-in to a single cloud provider causes immediate system outages during rate limits or downtime.

Mechanics: Central proxy dashboard (Port 20128 / 8000) with multi-provider vault, real-time latency measurement (ms), HTTP error monitor (429/500), automatic circuit breaker, and 4-tier fallback cascade

(GLM-4.7 Flash -> Gemini 2.5 Flash -> Claude 3.5 Sonnet -> local Llama 3.3 70B).

Benefit: *Ensures 99.9% uptime for all cognitive processes at minimal API costs.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

13. View: Token Guard (Guardian & Token Analytics in 3 Sub-Views)

Problem: Lack of visibility into which agent, model, or individual session generates the most tokens and costs.

Mechanics: Offers time filters (24h, 7d, 30d, Total), inactivity timer, and 3 sub-views: 1. Overview & daily trend chart with profile/model tables and protection threshold form, 2. Single-session usage inspector (sortable by input, output, cache-read tokens), 3. Resources & task throttling (slider from 1 to 16 tasks).

Benefit: *Complete cost control down to individual session IDs, including immediate budget and concurrency caps.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

14. View: Extensions & Module Center (34 Evey Specialty Plugins)

Problem: Security, storage, and analytics functions could previously not be toggled modularly per domain.

Mechanics: Controls 34 specialized Hermes/Evey plugins across 8 domain tabs: Security & Guards (4), Performance & Cache (2), Monitoring & Analytics (3), Agent Intelligence (6), Planning & Tasks (2), Bridges & Interfaces (4), Memory & Knowledge (3), Tools & Execution (3), plus Finance/Quality/Maintenance. Includes the standardized plugin system & runtime hooks: Modular directory structure (plugins/<plugin_id>/plugin.json), 4 lifecycle hooks (on_load, on_enable, on_disable, on_unload), and 2 tool interceptors (before_tool_execute, after_tool_execute) with pre-installed starter plugins (System Metrics, Security Audit Hook, Cost Guard).

Benefit: *Allows granular toggling of specific system capabilities at runtime with a single click.* Enables isolated runtime extensions and global tool interception without modifying kernel code.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

15. View: Research Lab (Deep Search & Evidence Triangulation)

Problem: Simple search queries return unverified single hits without source reconciliation or confidence scoring.

Mechanics: Multi-stage research cockpit with selectable search depth, source monitor, evidence score, live synthesis log for triangulating conflicting claims, and 1-click export to the Obsidian vault. Includes Autonomous Deep Research & Reflection Loop v2 (Re-Search Loop): Decomposes complex investigations into 3–5 sub-questions, runs autonomous gap analysis, performs up to 3 iterative follow-up search passes, multi-provider fallback (SearXNG / Jina Reader), and SSRF protection.

Benefit: *Produces solid, citable market, legal, and technical dossiers.* Autonomously resolves knowledge gaps through recursive follow-up investigations when facing conflicting or incomplete sources.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

16. View: Telemetry & Logs (3-Source Live Audit Stream)

Problem: Debugging errors in containers or agent prompts previously required tedious manual searches via terminal Docker commands.

Mechanics: Aggregates 3 log sources in real time: Physical Docker container logs (stdout/stderr across the 11 containers), cognitive agent logs, and the cryptographic audit trail. Includes log-level filtering (DEBUG, INFO, WARN, ERROR), auto-scroll lock, and export.

Benefit: *Tamper-evident flight recorder for rapid troubleshooting and compliance verification.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

17. Dedicated Studio: Archon Prompt Studio (832 Templates & Fabric SOP)

Problem: Imprecisely phrased tasks lead to hallucinations, incorrect output formats, and costly correction loops.

Mechanics: Built-in builder with 824 enterprise templates across 16 professional categories, over 255 Fabric SOP patterns, the 5-field formula (1. Role, 2. Context, 3. Task, 4. Format, 5. Constraints/Negative Rules), and an AI prompt optimizer.

Benefit: Enforces crystal-clear structure on every delegation and transfers it to chat or a Kanban task with one click.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

18. Dedicated Studio: Visual CSS Live Studio & In-Place Transform Gizmo (Alt + S)

Problem: Design adjustments previously required hunting through CSS files and repeatedly refreshing the browser.

Mechanics: Floating real-time design studio (Alt + S) with crosshair DOM inspector, interactive transform gizmo (drag position, scale 4 corners/edges, adjust corner radius), theme profile manager, and direct CSS export to custom_studio.css.

Benefit: *Enables pixel-accurate live styling of the entire user interface at runtime without page reloads.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

19. Dedicated Studio: Design System Lab, AppShell Architect & 15-Component Studio

Problem: Inconsistent header heights, spacing, or contrasts between light and dark modes disrupt visual harmony.

Mechanics: Includes the interactive AppShell Architect (56px header baseline, sidebar width, viewport padding with live miniature simulation) and the 15-component design studio (os-workspace through os-badge) with a dual-theme engine (Light/Dark).

Benefit: *Guarantees a consistent, glare-free B2B SaaS appearance across all 20 views.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

20. Dedicated Views: System Control Center (11 Domain Tabs, Live Dock) & Global Omnibox (Ctrl + K)

Problem: Settings were scattered, and navigating between 20 views and 70 agents required numerous clicks.

Mechanics: The System Control Center combines 7 administration tabs (Hermes Core including YAML editor, Containers & Services, Providers & Gateway, Methodologies, Plugins, Communication, Design) and can dock on the right via 'Live Dock'. The Omnibox (Ctrl + K) provides instant keyboard navigation to any tab (#), agent (@), or command. Integrates the 1-click update system & continuous deployment: Automatic GitHub sync with pre-update snapshots of all SQLite databases (WAL) in data/snapshots/ before each git pull, 1-click rollback, and minimal git-stat checks against mount timeouts.

Benefit: Maximum operating speed for administrators and power users. Enables safe software updates with automatic database rollback if errors occur.

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

3. Token Savings & FinOps Ecosystem (Cost Minimization)

21. 3-Tier Automatic Cost Routing Caps (LOW / MEDIUM / HIGH)

Problem: Running simple routine tasks on expensive top-tier models (such as Claude Sonnet or GPT-4o) causes daily costs to skyrocket.

Mechanics: Configurable price caps per 1 million tokens in the System Control Center: LOW (max. $0.10/1M -> forces Free-Tier/Flash-Lite), MEDIUM (max. $1.00/1M -> Haiku/Gemini Flash), HIGH (No Cap -> best reasoning models for complex architecture).

Benefit: *Reduces routine daily API spending by up to 85% without compromising quality on difficult tasks.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

22. FreeLLM-API Docker Proxy & 'auto/best-free' Routing (Port 3002)

Problem: Free tiers across different providers are difficult to monitor manually and change frequently.

Mechanics: A dedicated Docker container (freellmapi on Port 3002) aggregates free LLM endpoints. The 'auto/best-free' routing profile automatically selects the strongest available zero-cost model.

Benefit: *Enables continuous 24/7 operation of background and triage agents at exactly $0.00 in API costs.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

23. Variable Reasoning Budget Control in Master Chat & Router (0 to 16k Tokens)

Problem: Reasoning models (thinking mode) often consume thousands of invisible reasoning tokens on trivial questions.

Mechanics: 6-position slider (Off: 0 tok, Minimal: ~1k, Low: ~2k, Medium: ~4k, High: ~8k, Maximum: ~16k) in the chat footer and in agent_router.py.

Benefit: *Disables expensive reasoning loops entirely for simple tasks and reserves deep reasoning for genuinely complex problems.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

24. Evey-Cache & Cache TTL Control (24h Semantic & Exact Response Cache)

Problem: Recurring identical or closely related system queries repeatedly triggered new, billable API calls.

Mechanics: The evey-cache plugin stores model responses with configurable time-to-live (default: 24-hour TTL) and breaks out saved cache-read tokens separately in the Session Inspector.

Benefit: *Answers repeated queries in 0 milliseconds at 0 token costs.*

Cost & Trade-off: If the cache TTL is set too high (> 72h), the system serves stale responses for dynamic topics.

When to disable & thresholds: Disable during real-time queries (live exchange rates, server status). Default TTL: 24h.

25. Max-Context & Max-History Hard Clipper in Guardian (32k / 20 Turns)

Problem: Endlessly growing chat sessions re-send the full history with every new prompt, driving input tokens up exponentially.

Mechanics: Hard bounds configurable in Guardian for maximum context window (e.g., 32,000 tokens) and maximum transmitted chat turns (e.g., last 20 turns).

Benefit: *Reliably caps the maximum possible input cost per turn.*

Cost & Trade-off: Hard-truncates messages exceeding 32k tokens or 20 turns, dropping very early context.

When to disable & thresholds: Increase to 64k–128k in Hermes Core for in-depth document analysis.

26. Dynamic Token Clipper for Low-Context Models (backend/token_clipper.py)

Problem: Compact or local models (8k/16k) crash when large tool outputs exceed the context window.

Mechanics: Strictly preserves the system prompt (index 0) and the latest user instruction, intelligently trims oversized individual blocks, and prunes older intermediate steps.

Benefit: *Enables stable operation of ultra-low-cost small models even on extensive tasks.*

Cost & Trade-off: Trims oversized tool outputs from the middle ([...SNIP...]), which can omit specific lines of code from the raw text.

When to disable & thresholds: Temporarily disable when complete, untruncated raw logs are required.

27. Sliding Window Memory Compression (Consolidated Decision Context)

Problem: Simple message truncation loses earlier project decisions.

Mechanics: Condenses older conversation turns into a compact summary and injects it as a block labeled [CONSOLIDATED CONVERSATION & DECISION CONTEXT] at position 1.

Benefit: *Saves up to 70% of historical conversation tokens while fully preserving every decision made.*

Cost & Trade-off: The compression step costs ~800 tokens one time and smooths out verbatim wording from earlier rounds.

When to disable & thresholds: Fine-tune via protect_last_n (default: 20) in Hermes Core as needed.

28. LRU-K Page Swapping for Subagent RAM (backend/context_swapper.py)

Problem: Parallel subagents keep massive amounts of raw data in active prompt memory simultaneously.

Mechanics: Swaps context pages that are no longer actively needed out to SQLite once memory usage reaches 80% using the Least-Recently-Used policy, and reloads them only on demand (rehydration).

Benefit: *Keeps prompts lean and prevents unnecessary token costs for inactive background data.*

Cost & Trade-off: Swapping agent RAM out to disk and reloading it adds ~10–25 ms of I/O latency during context switches.

When to disable & thresholds: Keep active above 80% RAM utilization to prevent out-of-memory crashes.

29. JEV Router — 0-Token Deterministic Intent Pre-Filtering (jev_router.py)

Problem: Classifying every simple system command through a large language model wastes tokens and time.

Mechanics: Classifies incoming inputs prior to LLM invocation using a deterministic JSON Execution Vector and keyword heuristics.

Benefit: *Routes unambiguous system commands directly to the target routine without consuming a single LLM token.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

30. ASD-STE100 Simplified Technical English Prompter (.agents/skills/asd-ste100-prompter)

Problem: Wordy prose in system prompts and agent instructions inflates every single API call unnecessarily.

Mechanics: Transforms instructions according to the aerospace standard ASD-STE100 (one meaning per word, active verbs, concise sentences, zero filler words).

Benefit: *Reduces system prompt token length by roughly 30–40% while lowering tool call error rates.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

31. Anti-AI-Slop & Output Compression (.agents/skills/anti-ai-slop)

Problem: By default, language models generate lengthy polite pleasantries, introductory filler, and redundant summaries (costly output tokens).

Mechanics: Suppresses AI clichés, empty phrasing, and artificial padding directly in the system prompt, sanitizing generated outputs.

Benefit: *Saves up to 25% of premium output tokens and yields immediately actionable technical text.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

32. Pre-Flight Cost Estimator & Budget Forecast (backend/cost_estimator.py)

Problem: Before kicking off large tasks, runs had unpredictable costs.

Mechanics: Calculates projected input/output tokens and dollar costs upfront via POST /api/router/estimate-cost based on the current model price list and compares them against the task budget.

Benefit: *Warns about expensive runs before the first cent is spent.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

33. Dynamic Task Budget with 80% Economy Downgrade & 100% Stop

Problem: A single complex task could unnoticed exhaust the entire daily budget.

Mechanics: Monitors cumulative cost per task (e.g., limit $0.25): Automatically steps down to an economical model at 80% and pauses the task at 100% with a Telegram approval request.

Benefit: *Prevents cost outliers during multi-stage autonomous research runs.*

Cost & Trade-off: Upon reaching 80% of the task budget, execution downgrades to an economy model; at 100%, a hard stop is triggered.

When to disable & thresholds: Raise task budget (default: $0.50) to $2.00 for large architectural runs.

34. Loop Killer & Max Steps Limit (15 Steps / Jaccard > 90%)

Problem: When an agent enters an infinite loop after a failed tool call, it burns credit by the second.

Mechanics: Checks each step for exact repetitions (>= 3) or semantic circularity (> 90% similarity) and terminates tasks at the latest upon reaching the configurable max-steps limit (default: 15 steps).

Benefit: *Protects accounts against runaway overnight loops.*

Cost & Trade-off: Overly strict thresholds (Max Identical = 2 or Jaccard < 0.80) prematurely stop legitimate iterative tasks such as step-by-step test fixing or pagination.

When to disable & thresholds: Temporarily disable when an agent intentionally executes numerous uniform loop steps (e.g., bulk migration of 50 files). | Thresholds: Max Identical: 2 = stops retries | > 5 = expensive loops (default: 3). Jaccard: < 0.80 = false positives | > 0.96 = late loop detection (default: 0.90). Max steps: default 15.

4. Kernel, Scheduling & Daemon Architecture (Pillar 1)

35. Hermes Master Orchestrator (Hermes-Core)
Problem: Uncoordinated individual agents work at cross-purposes or lose sight of the overarching objective.
Mechanics: Central process command center for intake analysis, role selection, oversight, and deliverable sign-off.
Benefit: Maintains strategic coherence across all 70 specialists.
Cost & Trade-off: Additional active background services and high reasoning settings ('high') increase baseline load and token expenditures.
When to disable & thresholds: Granularly controllable across the 14 sections in the 'HERMES CORE' tab.

36. Tickless Kernel Daemon & Asyncio Event Loop (Port 8420 / 8501)

Problem: Constant second-by-second polling wastes CPU cycles and system power.

Mechanics: Calculates the exact moment of the next due task ('sleep-until-next-due') and suspends the kernel until then.

Benefit: *Minimal idle system load coupled with zero-latency wake-up response.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

37. Priority Preemption Scheduler (P0–P3 Queue)

Problem: Urgent user inquiries in direct chat had to wait when background jobs occupied every slot.

Mechanics: Prioritized queue: P3 tasks (direct chat / emergencies) immediately preempt active P1/P2 routine jobs.

Benefit: *Guarantees rapid conversational response times despite heavy background workload.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

38. Freely Scalable Concurrency Throttling (backend/concurrency_throttle.py)

Problem: Too many concurrent agents trigger HTTP 429 rate limits and 99% CPU spikes.

Mechanics: Regulates active workers through an Asyncio semaphore (adjustable via slider from 1 to 16 parallel tasks in Guardian and Control Center).

Benefit: *Allows throttled safe operation on laptops as well as high-throughput workloads on workstations.*

Cost & Trade-off: Tight stagnation limits (< 30 sec) prematurely abort slow deep-reasoning models (e.g., o3/R1) mid-thought; additional workers increase peak RAM and API load.

When to disable & thresholds: Disable during extremely long-running single analyses or local LLM runs on slower hardware. | Thresholds: Max Workers: 1–2 = backlog | > 8 = RAM spikes & HTTP 429 (default: 4). Stagnation: < 30s = aborts reasoning | > 180s = delayed recovery (default: 45s).

39. Autonomous 30-Second Watchdog & Crash Recovery (watchdog_daemon.py)

Problem: Crashed or frozen tasks remained stuck indefinitely in the in_progress status.

Mechanics: Inspects all running processes every 30 seconds, aborts zombie tasks following a timeout (120s), and transfers orphaned tasks into automatic crash recovery.

Benefit: *Keeps all worker slots free of deadweight processes.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

40. Deadlock & Stagnation Supervisor (backend/deadlock_supervisor.py)

Problem: Mutual dependencies between two agents can cause a silent system freeze.

Mechanics: Detects circular wait states and tasks stalled without progress (> 45s), resolving the deadlock autonomously with exponential backoff.

Benefit: *Prevents pipeline stalls during complex multi-agent workflows.*

Cost & Trade-off: An overly aggressive stagnation limit (< 30s) aborts slow deep-reasoning models (o3/R1) in the middle of generation.

When to disable & thresholds: When using reasoning models, increase stagnation limit to 90–120s (default: 45s).

41. Fleet Self-Healing & Container Auto-Restart (backend/fleet_self_healing.py)

Problem: If one of the 11 Docker containers failed, it had to be restarted manually.

Mechanics: Monitors all core containers via the secured docker-socket-proxy and autonomously executes a restart with an audit log entry upon failure (> 60s).

Benefit: *Ensures unattended 24/7 operation across the entire Docker stack.*

Cost & Trade-off: A persistently failing container can get caught in a restart loop due to auto-restart.

When to disable & thresholds: Temporarily disable during manual container maintenance.

42. Proactive Morning Standup & Wake-Up Service (backend/autonomous_standup.py)

Problem: In the morning, a concise summary of overnight runs, expenses, and pending decisions was missing.

Mechanics: Generates a consolidated status report in the Obsidian vault (01_PROJEKTE/Daily_Briefings/) daily at 08:00 and pushes it to the dashboard and Telegram.

Benefit: *Starts every workday with an actionable decision memo.*

Cost & Trade-off: Each automated morning briefing consumes ~1,500–3,000 tokens at the scheduled time.

When to disable & thresholds: Disable if daily morning reports are not needed (default: 08:00).

5. Multi-Agent Process Model, Hierarchy & Synthesis (Pillar 2)

43. 70 Declarative Agent Profiles & Markdown Personnel Files
Problem: Generic AI assistants lack role awareness and defined quality standards.
Mechanics: Each of the 70 specialists is anchored in agent_profiles.json and an individual personnel file (00_HR_Agenten/) detailing system prompt, model routing, tool scope, and SOPs.
Benefit: Establishes dependable virtual domain experts for strategy, legal, construction, finance, marketing, and IT.
Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

44. Robert Delegation, Fan-In Synchronization & Lisa Marker Contract (SYNTHESE-BEREIT)

Problem: During parallel subtasks, intermediate results were lost or consolidated incompletely.

Mechanics: Robert breaks down major projects; a process lock (_MC_DATA_LOCK) waits for completion of all subtasks (fan-in). Lisa aggregates the reports while verifying the mandatory SYNTHESE-BEREIT marker; Robert grants formal approval (FREIGABE) in line 1.

Benefit: *Multi-tiered quality assurance modeled after enterprise executive management.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

45. Standardized Agent-to-Agent Handoff Desk (backend/handoff_desk.py)

Problem: Silent context loss when handing off tasks from one specialist to the next.

Mechanics: Generates formal handoff protocols with deliverables, open assumptions, and SHA-256 checksums that must be acknowledged by the receiving agent.

Benefit: *Audit-proof interfaces across all departments.*

Cost & Trade-off: Strict SLA checks reject very concise but accurate factual answers when min_content_length is set too high (> 300).

When to disable & thresholds: Default: 100-character minimum length, 60-minute turnaround.

46. Two-Stage DAG Scheduler for Parallel Project Execution (backend/dag_scheduler.py)

Problem: Sequential processing of multiple subtasks takes too long on large projects.

Mechanics: Computes the directed acyclic graph (DAG): Executes independent research tasks (Stage 1) in parallel and automatically triggers synthesis (Stage 2) afterward.

Benefit: *Substantially reduces the turnaround time for complex analyses.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

47. Self-Correction & Tool Execution Reflection Loop

Problem: Transient syntax or parameter errors during tool calls previously caused immediate failures.

Mechanics: Catches tool errors, reflects on the error output, and autonomously corrects call parameters in up to 3 attempts.

Benefit: *Markedly higher reliability across file, database, and web operations.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

48. Interruptible Long-Running Workflows with SQLite State Snapshotting

Problem: When a task waits hours for user approval, it consumes memory unnecessarily or loses state across a restart.

Mechanics: Freezes the exact execution state in the workflow_checkpoints table and seamlessly resumes work once approval is granted.

Benefit: *Enables multi-day business processes to persist across restarts.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

49. Dual-Agent Consensus Gate & Adversarial Critic Review (backend/consensus_gate.py)

Problem: An individual agent often misses its own reasoning errors or hallucinations on critical tasks.

Mechanics: Enforces a formal peer review by an independent auditor agent (Red Team / Critic) for P3 tasks and critical operations. Wired into the Kanban board as an automated 4-eyes quality gate (Katharina Völk / Robert review with 1–100% score and status lock when revisions are required).

Benefit: *Four-eyes principle for all mission-critical deliverables.*

Cost & Trade-off: Triples (3x) token consumption and increases response time by +6 to +20 seconds; requiring unanimity risks deadlocks.

When to disable & thresholds: Activate only from priority P3/P4 onward and require a 2/3 majority.

6. MCP Drivers, External Services & Multimodal Tools (Pillar 3)

50. Archon CRM & PocketBase Ecosystem (Ports 3005, 8090, 8092)

Problem: Customer contacts, email histories, and media assets were isolated from AI agents.

Mechanics: Dedicated CRM frontend (Port 3005) with PocketBase backend (Port 8090) and media server (Port 8092), fully integrated via scripts/archon_crm_mcp.py.

Benefit: *Agents can retrieve contacts, conduct background research, score leads, and assign correspondence.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

51. Automatic Email Matching & LinkedIn Archive Importer

Problem: Thousands of contacts from LinkedIn exports and email mailboxes had to be reconciled manually.

Mechanics: Synchronizes mailboxes (--fetch-mails), automatically maps emails to existing CRM contacts, and imports entire LinkedIn connections CSVs with deduplication.

Benefit: *Builds an enriched, complete relationship database in minutes.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

52. Robert Tool-Permission Manager (backend/tool_permission_manager.py)

Problem: Agents should not fail abruptly when Python/Node packages are missing, yet they must not install packages without permission.

Mechanics: Robert submits a formal installation request (pending_permission) detailing the command and rationale; the package is installed only after your 1-click approval.

Benefit: *Controlled extensibility while fully protecting the host system.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

53. Event-Driven Webhook Mesh (backend/event_mesh.py)

Problem: External events from GitHub, web forms, or third-party systems could not trigger agent actions.

Mechanics: Universal webhook receiver that validates external JSON payloads and converts them directly into executable Kanban tasks.

Benefit: *Connects Archon OS in real time with the rest of your IT landscape.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

54. Central Multi-Channel Notification Hub (backend/notification_hub.py)

Problem: Important alerts were lost in the UI, while trivial notifications created noise on mobile devices.

Mechanics: Routes messages strictly by severity: INFO -> UI toast, WARNING -> dashboard log, CRITICAL -> immediate Telegram push.

Benefit: *Targeted alerting without notification fatigue.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

55. Multi-Modal Vision & Screen-Capture Pipeline (backend/vision_pipeline.py)

Problem: Visual issues or tables in images had to be described manually and tediously.

Mechanics: Direct in-browser screen capture (camera button in chat) along with automated OCR and structural analysis for uploaded images (.png, .jpg, .webp).

Benefit: *Agents see exactly what you see on your screen.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

56. Apollo Voice Engine — Edge-TTS & Whisper Dictation (apollo_voice_engine.py)

Problem: Typing long texts on a smartphone or desktop takes time, and users often prefer listening to reports on the go.

Mechanics: Bidirectional voice pipeline: microphone dictation (speech-to-text) directly in the chat input field and natural voice output (text-to-speech) for briefings. Full Telegram voice integration (POST /api/telegram/voice-relay and POST /api/voice/synthesize) for mobile voice memos via edge-tts (de-DE-ConradNeural).

Benefit: *Transforms Archon OS into a voice-controlled executive assistant.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

57. Oracle News Radar & SEO Execution Agent

Problem: Market shifts and SEO weaknesses on your own websites were checked only sporadically.

Mechanics: Oracle News Radar continuously scans industry sources for project relevance; the SEO Agent audits pages for meta tags, structure, and keyword density.

Benefit: *Continuous market and visibility optimization in the background.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

7. Memory, Database Hygiene & Obsidian Second Brain (Pillar 4)

58. Central SQLite WAL Memory Layer (5 Standard Tables)

Problem: Concurrent read and write operations from multiple agents caused conflicts with plain files.

Mechanics: Transactional SQLite database (data/agentic_memory.db) in Write-Ahead Logging (WAL) mode with the 5 core tables tasks, conversations, learnings, kv_store, and user_profile.

Benefit: *Non-blocking concurrent access with maximum data integrity.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

59. Process-Safe Kanban Sidecar with Atomic File Swapping (backend/kanban_sidecar.py)

Problem: Metadata such as WIP limits or tags could get corrupted during parallel write operations.

Mechanics: Protects config/kanban_meta.json via a process-wide mutex and atomic write-replace operations.

Benefit: *Prevents data loss during high-frequency board updates.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

60. Docker RAG Bridge to AnythingLLM (5,068 Indexed Documents on Port 3001)

Problem: Knowledge stored in the Docker vector store was often not automatically retrieved during standard chat queries.

Mechanics: backend/rag_bridge.py connects the agent core directly to the AnythingLLM ARCHON workspace, delivering relevant text passages as grounded context.

Benefit: *Informed responses grounded in your actual enterprise documents rather than generic guesses.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

61. Three-Tier Project Folder Disposition in Obsidian Vault

Problem: When agents generate files without strict rules, clutter quickly accumulates in the vault.

Mechanics: Automatically creates a 3-tier structure for every project: 01_Zwischenberichte, 02_Medien, and 03_Endprodukte.

Benefit: *Clean separation between rough drafts and approved final deliverables.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

62. Obsidian Vault Indexer & Automated Map of Content (backend/vault_hygiene.py)

Problem: New project reports frequently remained unlinked in the Obsidian graph ('orphan notes').

Mechanics: Scans the vault, detects orphaned notes, and automatically updates the central _MOC_Projekt_Index.md with wikilinks.

Benefit: *Keeps the knowledge graph in Obsidian fully interlinked at all times.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

63. Autonomous Fact Harvesting & 85% Similarity Deduplication (backend/post_task_pipeline.py)

Problem: Key insights from completed tasks were lost or saved multiple times as duplicates.

Mechanics: Automatically extracts new facts for learnings and user_profile after task completion and blocks duplicates above 85% text similarity.

Benefit: *The system grows smarter with every completed task without bloating memory storage.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

64. Automated Online DB Backup with 7-Day Rotation (backend/db_backup_manager.py)

Problem: Without regular database backups, hardware failures risk the loss of all tasks and profiles.

Mechanics: Executes PRAGMA wal_checkpoint(TRUNCATE) during live operations and saves compressed .db.gz archives in .hermes/backups/db/.

Benefit: *Seamless data backups without a single second of downtime.*

Cost & Trade-off: Daily full backups consume ~50–500 MB of disk space depending on the retention period.

When to disable & thresholds: Default: 7-day retention (db_backup_keep_days).

65. Autonomous SQLite Vacuum & Fragmentation Optimizer (backend/db_optimizer.py)

Problem: Deleted records and frequent updates inflate SQLite files unnecessarily over months.

Mechanics: Executes PRAGMA optimize and VACUUM across all databases via POST /api/system/optimize-db and calculates reclaimed disk space.

Benefit: *Keeps query latencies consistently in the millisecond range.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

66. Atomic Pre-Write Snapshotting & 1-Click Rollback (backend/snapshot_manager.py)

Problem: When an agent erroneously overwrote a file, recovering the previous state was cumbersome.

Mechanics: Automatically backs up the original to .hermes/snapshots/ before any write operation and restores it instantly via click or API. Also integrated as pre-update snapshots of all SQLite databases in data/snapshots/ before GitHub pulls and container updates.

Benefit: *Full safety net for all file and code changes.*

Cost & Trade-off: Each pre-write snapshot incurs ~5–15 ms of disk I/O before overwriting a file.

When to disable & thresholds: Ring buffer limited to 50 snapshots by default.

67. Git-Compatible Live Diff Preview (backend/diff_engine.py)

Problem: Prior to approval, operators want to see exactly which lines change instead of re-reading the entire file.

Mechanics: Generates a color-coded line-by-line diff (red = removed, green = added) between existing content and the agent draft.

Benefit: *Second-fast review of code and document changes in the Human-in-the-Loop (HITL) Approval Gate.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

8. Security, Zero-Trust Governance & System Diagnostics (Pillar 6)

68. Cryptographically Chained SHA-256 Audit Log (backend/audit_log.py)

Problem: Plain text logs can be tampered with or deleted retroactively.

Mechanics: Every event is chained with the SHA-256 hash of its predecessor (blockchain principle) and can be verified for tampering via GET /api/audit/verify.

Benefit: *Audit-proof trail of all system and approval actions.*

Cost & Trade-off: Minimal write overhead (~2 ms per entry); the cryptographic chain grows over months.

When to disable & thresholds: Keep permanently enabled for audit-proof governance.

69. Local Security Guard — 25 Prompt Injection & Jailbreak Signatures

Problem: Malicious emails or websites may contain hidden commands ('Ignore previous instructions...').

Mechanics: Scans every external text input against 25 attack patterns before it enters an agent's context.

Benefit: *Defends against indirect prompt injections at the system perimeter.*

Cost & Trade-off: Strict heuristics may trigger false positives on IT security texts or penetration testing prompts.

When to disable & thresholds: Loosen temporarily only for authorized red-teaming tests.

70. Secret Masking & Red-Teaming Output Guard (backend/secret_masker.py)

Problem: API keys or passwords could inadvertently appear in logs, reports, or chat outputs.

Mechanics: Filters outbound text in real time for key patterns (sk-..., Bearer, passwords) and replaces them with [GESCHÜTZTER_KEY].

Benefit: *Prevents accidental disclosure of confidential credentials.*

Cost & Trade-off: Aggressive regex masking may occasionally redact benign Base64 or hash strings in code outputs.

When to disable & thresholds: Keep permanently active in production to protect API keys.

71. Hardened Docker Socket Proxy (Port 2375 internal)

Problem: Direct access to /var/run/docker.sock grants containers unrestricted root privileges over the host system.

Mechanics: Interposes a restrictive proxy container that permits only explicitly whitelisted status and restart commands.

Benefit: *Protects the host operating system under the zero-trust principle.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

72. A/B Model Benchmarker & Empirical Performance Scorer (backend/model_benchmarker.py)

Problem: New language models arrive weekly, making their fit for specific enterprise tasks difficult to assess.

Mechanics: Tests two models in parallel with identical prompts, comparing response latency, token costs, and output quality.

Benefit: *Objective empirical baseline for model selection across the Specialist Fleet.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

73. System-Wide 360° Health Score & Diagnostic Engine (backend/system_diagnostics.py)

Problem: Individual metrics rarely reveal at a glance whether the entire system is truly healthy.

Mechanics: Calculates an aggregate score from 0–100% with actionable recommendations based on Docker fleet status (35%), concurrency queue (20%), audit integrity (20%), deadlock freedom (15%), and RAG reachability (10%).

Benefit: *Instant operating system health diagnostics via GET /api/system/health-score.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

9. Self-Evolution, Marketplaces & Auto-Documentation (Pillar 7)

74. Autonomous Skill Synthesis with Personnel File Feedback (backend/skill_synthesizer.py)

Problem: Previously, successful multi-step workflows had to be coded manually into new skills.

Mechanics: Detects recurring successful tool chains, automatically drafts a SKILL.md, logs it into the agent's personnel file, and submits it for Human-in-the-Loop (HITL) approval.

Benefit: *The system autonomously expands its capabilities during runtime.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

75. Dynamic Talent & Skill Marketplace (skill_markt.js & AgentSkillsHub)

Problem: Writing new roles or workflows from scratch for new business areas consumes significant time.

Mechanics: Searchable catalog for 1-click import and export of complete agent profiles and external skills (including agentskillshub.dev integration).

Benefit: *Scales the agent fleet for new projects in seconds.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

76. Complete Frontend-Backend Synchronization (238 Fetches / 581 Paths / 29 Modals)

Problem: In large web applications, orphaned buttons or disconnected modals often lead to silent failures.

Mechanics: Automated validation pipeline ensures all 238 frontend API calls and 29 modals are wired precisely to the 368 backend routes.

Benefit: *Guarantees 0 dead links and 100% functional UI controls across all views.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

77. Autonomous Master Dossier & Manual Generator (backend/system_dossier_generator.py)

Problem: Architecture documentation typically goes out of date within days of creation.

Mechanics: Generates the current system architecture dossier on demand (POST /api/system/generate-dossier) simultaneously in the project root and in the Obsidian vault (00_DOKUMENTATION/).

Benefit: *Maintains up-to-date system documentation without manual upkeep overhead.*

Cost & Trade-off: Integrated into the dashboard as permanent core view and system infrastructure (negligible baseline RAM/CPU footprint; token costs occur only during active execution of LLM actions within the module).

10. Reference Matrix: Controllable Evey/Hermes Plugins

No. & Plugin ID	Category	Core Mechanism & Operating System Function
78. evey-cost-guard	Security & Sentinel	Real-time cost monitoring and hard budget cutoffs per day and task.
79. evey-session-guard	Security & Sentinel	Monitors session runtimes and token limits to prevent zombie sessions.
80. evey-email-guard	Security & Sentinel	Scans incoming emails for phishing and prompt injections prior to triage.
81. evey-watchdog	Security & Sentinel	Checks active processes every 30 seconds and automatically restarts stalled workers.
82. evey-cache	Performance & Cache	24-hour response cache for immediate, zero-cost responses to repeated queries.
83. disk-cleanup	Performance & Maintenance	Cleans temporary artifacts and stale logs to reclaim disk space.
84. evey-telemetry	Monitoring & Analytics	Captures token streams, latencies, and model calls for Mission Control and Guardian.
85. evey-status	Monitoring & Analytics	Continuous health pings across all 11 Docker containers and gateways.
86. evey-delegation-score	Monitoring & Analytics	Evaluates the success rate and quality of task handoffs between agents.
87. evey-autonomy	Agent Intelligence	Controls the autonomy tier for multi-stage background decisions.
88. evey-council	Agent Intelligence	Majority voting and consensus building across multiple specialists (mastermind council).
89. evey-reflect	Agent Intelligence	Self-reflection loop for autonomous error correction before result submission.
90. evey-learner	Agent Intelligence	Extracts durable insights from completed tasks into the learnings table.
91. evey-identity	Agent Intelligence	Enforces strict compliance with SOUL.md, tone, and role guardrails.
92. evey-habits	Agent Intelligence	Manages recurring routines such as the daily 08:00 AM standup briefing.
93. evey-proactive	Agent Intelligence	Enables agents to proactively suggest actions upon detecting bottlenecks.
94. evey-goals	Planning & Tasks	Links operational Kanban tasks to high-level corporate objectives.
95. evey-scheduler	Planning & Tasks	Time- and priority-scheduled queuing of background jobs.

© Michael Keller

96. evey-bridge	Bridges & Interfaces	Internal data hub bridging the FastAPI monolith, Hermes daemon, and sidecars.
97. evey-mqtt	Bridges & Interfaces	Lightweight messaging protocol for IoT and real-time status signals.
98. evey-moltbook	Bridges & Interfaces	Synchronizes structured workbooks and synthesis notes.
99. evey-telegram-ux	Bridges & Interfaces	Controls call-sign routing (/robert, /anwalt) and interactive approval buttons in Telegram.
100. evey-github	Bridges & Interfaces	Integration for repository operations, commit analysis, and issue tracking.
101. evey-research	Tools & Research	Deep-search driver for web scraping and source extraction in the Research Lab.
102. evey-news	Tools & Research	Background scanner for industry feeds and market updates (Oracle News Radar).
103. evey-digest	Tools & Research	Condenses extensive document collections into compact executive summaries.
104. evey-sandbox	Tools & Execution	Isolated runtime environment for generated Python and analytics code in the Showroom.
105. evey-rag	Memory & Knowledge	Vector retrieval driver connecting AnythingLLM and local knowledge archives.
106. evey-memory-adaptive	Memory & Knowledge	Dynamic weighting of short-term and long-term memory based on context pressure.
107. evey-memory-consolidate	Memory & Knowledge	Nightly run to merge and deduplicate related memory entries.
108. evey-wallet	Finance & Budgets	Manages API balances, quotas, and cost-based transaction limits.
109. evey-validate	Quality & Execution	Validates outputs against schema conformance, mandatory markers (SYNTHESE-BEREIT), and formatting constraints.
110. evey-verification	Quality & Execution	Fact-checking and cross-verification of critical claims prior to final sign-off.
111. evey-delegate-model	Quality & Execution	Automatically selects the most cost- and performance-optimal target model for subtasks.

11. Enterprise & Resilience Architecture

78. Replay & Time-Travel Debugging (backend/time_travel_debugger.py)

Problem: In complex agent workflows with numerous tool calls, diagnosing errors in text logs is cumbersome and branching execution paths is impossible.

Mechanics: Stores each intermediate step with inputs/outputs, snapshots, and token usage in 'execution_traces'. Enables interactive replay (GET /api/traces/{task_id}), state forking from step N (POST /api/traces/{task_id}/fork), and

1-click rollback of modified files.

Benefit: Rapid root-cause analysis and risk-free what-if experiments with alternative prompts or models without restarting the entire pipeline.

Cost & Trade-off: Increases disk I/O by ~5–15 ms per file edit and consumes ~50–250 MB of additional disk space depending on the snapshot limit.

When to disable & thresholds: Disable on systems with severely constrained disk space or during mass file generation (e.g. build artifacts). | Thresholds: Max Snapshots: < 10 = short history | > 200 = higher disk usage (Default: 50). Max Trace Steps: Default 100.

79. Multi-Model Eval Matrix & A/B Benchmarker (backend/model_benchmarker.py)

Problem: Model assignments to agents often rely on subjective assumptions without empirical benchmark data.

Mechanics: Dispatches the same prompt simultaneously to two configured LLMs (POST /api/router/benchmark), measures latency, token usage, and response length, and persistently logs history in 'model_benchmarks' (GET /api/router/benchmark/history).

Benefit: Objective, data-driven model selection per agent role (e.g. DeepSeek V3 for speed/cost vs. Claude 3.7 for complex architectures).

Cost & Trade-off: N candidate models consume N times the API tokens per test run.

When to disable & thresholds: When to disable? Run only intentionally for model evaluations, not continuously.

80. Tri-Agent Consensus Gate & Arbitration Council (backend/consensus_gate.py)

Problem: Critical interventions (code refactoring, migrations) carry risks of hallucinations or security flaws when made by single agents.

Mechanics: Convenes a three-member panel (System Architect, Security Officer, QA Lead) via POST /api/consensus/council-vote. Each reviewer votes independently; approval requires a 2/3 majority. All resolutions are archived in 'consensus_decisions' (GET /api/consensus/history).

Benefit: Reliable risk mitigation and elimination of single-point-of-failure hallucinations during sensitive system modifications.

Cost & Trade-off: Triples (3x) token consumption and increases response latency by +6 to +20 seconds. Requiring unanimity risks decision deadlocks.

When to disable & thresholds: Disable (or activate only starting from Priority 4 = Critical) when execution speed and low API costs take precedence. | Thresholds: Minimum Priority: Prio 2 = nearly every task costs 3x tokens | Prio 4 = critical tasks only (Default: Prio 3). Majority: 2/3 majority recommended.

81. Autonomous Task Self-Healing with Git Rollback (backend/task_self_healer.py)

Problem: Syntax errors or broken imports stall tasks immediately and require manual user intervention.

Mechanics: Isolates the stack trace, performs a Git rollback of modified files upon runtime errors, and feeds the agent a structured correction prompt (POST /api/healing/attempt). Up to 3 autonomous repair attempts; retrieve logs via GET /api/healing/logs.

Benefit: Maximum resilience during code and script execution with automated self-repair prior to escalating to the HITL approval board.

Cost & Trade-off: Each self-healing attempt consumes 1 additional LLM call (+3 to +12 s latency and token cost). On failure, an automated Git rollback also discards useful partial progress.

When to disable & thresholds: Disable Git rollback if unfinished code drafts should remain in the working tree for manual inspection. | Thresholds: Max Repair Attempts: 1 = gives up immediately | > 5 = expensive cascading repairs (Default: 3).

82. Event Mesh & Inbox Directory Watcher (backend/event_mesh.py)

Problem: Tasks previously had to be triggered manually via the dashboard or queried through scheduled IMAP polling.

Mechanics: Central inbound event bus for external webhooks (POST /api/events/inbound/{channel}) and automated directory watcher (POST /api/events/scan-directory) that transforms new files in 'data/inbox/' directly into Kanban triage tasks.

Benefit: Fully automated event and document intake from external systems (GitHub, n8n, Stripe, local document scanners) without manual intervention.

Cost & Trade-off: Very short scan intervals (< 3 s) generate continuous filesystem polling; the daily morning standup uses ~1,500–3,000 tokens per run.

When to disable & thresholds: Disable if the local inbox folder is not actively used or no morning briefing report is needed. | Thresholds: Scan Interval: 1–2s = unnecessary disk polls | > 30s = delayed file detection (Default: 5s). Standup Time: Default 08:00 AM.

12. Autonomy, Sandbox & Governance Architecture

83. Ephemeral Code Sandboxes (backend/ephemeral_sandbox.py)

Problem: Executing agent-generated code risks system crashes, infinite loops, or unintended host file modifications.

Mechanics: Executes code in short-lived, isolated Docker micro-containers with strict resource limits (512 MB RAM, 1 CPU, 30s timeout, no network) via POST /api/sandbox/execute. The sandbox is wiped completely upon completion.

Benefit: Total security and system stability during automated code execution and testing.

Cost & Trade-off: Container cold starts add ~300–800 ms latency per tool call. With networking disabled, pip install or external API calls inside the script fail immediately.

When to disable & thresholds: Disable (or enable networking) if trusted local scripts require direct host access or package installations. | Thresholds: RAM Limit: < 256 MB = OOM kill on Pandas/NumPy | > 2048 MB = host RAM exhaustion risk (Default: 512 MB). Timeout: < 10s = aborts builds prematurely (Default: 30s).

84. Nightly Memory Consolidation (backend/memory_consolidator.py)

Problem: Steadily accumulating traces, heartbeats, and stale dialog logs slow down database queries over long-term operations.

Mechanics: Prunes stale traces and non-critical logs during idle windows (default 03:00 AM) based on configurable retention policies (settings.json). Fully manageable via Hermes Core settings (execution time, retention days, pruning threshold, manual run button).

Benefit: Consistently lean databases, reduced latencies, and automated system hygiene with zero maintenance overhead.

Cost & Trade-off: During consolidation (~10–30 s), brief CPU/disk spikes occur; an excessively low pruning threshold (< 0.88) may merge related yet distinct facts.

When to disable & thresholds: Disable if complete raw traces must be retained unfiltered for forensic audits. | Thresholds: Retention: < 7 days = loss of audit trails | > 60 days = oversized SQLite DB (Default: 14 days). Pruning Threshold: 0.95 = conservative | 0.88 = aggressive (Default: 0.92).

85. Visual Web Agent with Screenshot Audit (backend/visual_web_agent.py)

Problem: Tasks on complex web portals lacking APIs (SPAs, web forms, interactive dashboards) cannot be handled by text-only scrapers.

Mechanics: Drives a full headless Chromium browser (POST /api/browser/browse-and-capture), captures screenshots for every click and input step in .hermes/visual_traces/{task_id}/, and extracts structured data.

Benefit: End-to-end visual traceability and automated interaction with modern web apps that lack API access.

Cost & Trade-off: Headless Chromium requires ~300 MB RAM and ~3–5 s rendering time per active audit run.

When to disable & thresholds: When to disable? In pure backend deployments without UI validation tasks.

86. Virtual War Room & Pre-Mortem Stress Test (backend/war_room_simulator.py)

Problem: Unforeseen security, user adoption, or compliance risks often go unnoticed until after rollout.

Mechanics: Simulates three adversarial personas (critical enterprise customer, offensive pentester, strict compliance auditor) before approval (POST /api/simulation/war-room). Builds an aggregated risk matrix (LOW to CRITICAL) with mandatory requirements.

Benefit: Proactively exposes vulnerabilities and averts costly missteps before going live.

Cost & Trade-off: Consumes ~1,500–3,000 extra tokens and adds +4 to +10 s latency before the actual task. Setting the blocking threshold to MEDIUM raises false-positive rates.

When to disable & thresholds: Disable auto pre-mortem during routine development; reserve it for production-critical changes (HIGH/CRITICAL). | Thresholds: Block Threshold: MEDIUM = blocks frequently out of caution | CRITICAL = halts only catastrophic risks (Default: HIGH).

87. Inter-Agent SLA & Handoff Contracts (backend/sla_contract_enforcer.py)

Problem: Loose delegation in multi-agent hierarchies often produces incomplete work that gets escalated to human operators.

Mechanics: Forms digital contracts specifying required fields, minimum lengths, and acceptance criteria during internal delegations (POST /api/contracts/create). Autonomously validates deliverables (POST /api/contracts/validate) and rejects substandard results for rework.

Benefit: Strict quality control across agent teams, preventing unfinished intermediate drafts from landing on the HITL approval board.

Cost & Trade-off: Setting minimum length thresholds too high (> 300 characters) rejects concise, accurate answers and triggers unnecessary rework loops.

When to disable & thresholds: Disable auto-reject if subagents are allowed to return short status updates or yes/no answers. | Thresholds: Min Length: < 30 = lets placeholders through | > 300 = rejects concise factual answers (Default: 100 characters). Max Turnaround: Default 60 min.

88. Central Autonomy & Threshold Controller (backend/autonomy_settings.py)

Problem: Hardcoded thresholds in program logic prevent swift adjustments to security, sandbox, and resilience policies during runtime.

Mechanics: Consolidates all 10 autonomy groups (concurrency, loop-killer, self-healing, Docker sandbox, time-travel snapshots, consensus gate, war room, SLA contracts, inbox/standup, and fleet hygiene) in settings.json and syncs them with the System Control Center via GET/POST /api/settings/autonomy-modules.

Benefit: Every protection and autonomy module can be toggled via click at runtime and fine-tuned to match specific risk and resource profiles.

Cost & Trade-off: A failing container may enter a crash loop under auto-restart; the visual browser audit spins up Chromium (+300 MB RAM, +4 s duration).

When to disable & thresholds: Disable auto-restart during manual container maintenance; deactivate browser auditing in backend-only workloads. | Thresholds: DB Backup Days: < 3 days = sparse history | > 30 days = ~500 MB+ backup footprint (Default: 7 days).

13. Performance Core, Throughput Scaling & Token Efficiency

89. Semantic Prompt & Response Caching (backend/semantic_cache.py)

Problem: Recurring or slightly rephrased agent queries incur unnecessary LLM API costs and multi-second wait times.

Mechanics: Computes a SHA-256 hash and an L2-normalized word/3-gram vector in SQLite for every prompt, instantly returning the cached response hit when cosine similarity is >= 0.95.

Benefit: Sub-10 millisecond response times and 100% token savings on semantically identical queries.

Cost & Trade-off: If the cosine threshold is set too low (< 0.92), the cache misses subtle semantic distinctions (e.g. 'delete test DB' vs. 'delete prod DB') and serves stale or incorrect responses.

When to disable & thresholds: Disable on tasks involving real-time data (stock quotes, live logs, system telemetry) or when each response must be freshly generated. | Thresholds: Cosine Threshold: < 0.92 = incorrect answers served from cache! | > 0.98 = very few hits (Default: 0.95). TTL: Default 24h. Max Entries: Default 500.

90. Parallel Map-Reduce Subtasking / Fan-Out & Fan-In (backend/map_reduce_executor.py)

Problem: Sequential processing of numerous documents or sub-questions leads to linear runtimes and extended

wait periods.

Mechanics: Splits list tasks across up to N parallel Asyncio workers (semaphore-throttled) and synthesizes partial outputs during the reduce stage.

Benefit: Multiplies throughput on document, research, and batch code analyses without overloading the event loop.

Cost & Trade-off: Generates N-fold concurrent API load (risking HTTP 429 rate limits from providers) and introduces ~20–35% token overhead during downstream reduce synthesis.

When to disable & thresholds: Disable when facing strict API rate limits or when subtasks depend sequentially on one another. | Thresholds: Max Workers: > 8 = rate-limit risk (Default: 4). Chunk Timeout: Default 30s. Trigger Count: Default 3 items.

91. Speculative Fast Routing & Latency Hedging (backend/speculative_hedger.py)

Problem: Sporadic slow cloud requests (tail latency spikes) block overall agent execution.

Mechanics: Automatically initiates a parallel backup request on a fast fallback model once hedge_delay_seconds (e.g. 2.5s) elapses, accepting whichever response finishes first.

Benefit: Eliminates stalls caused by sluggish API endpoints and stabilizes 99th-percentile response times.

Cost & Trade-off: If the primary model answers shortly after the hedge triggers, both requests are billed (up to 2x token costs on slow prompts).

When to disable & thresholds: Disable on tight API budgets or when running deep reasoning models that routinely require > 10 s thinking time. | Thresholds: Hedge Delay: < 1.5s = near-guaranteed double billing! | > 8s = minimal latency protection (Default: 2.5s).

92. Dynamic Context Compaction & Hierarchical Summarization (backend/context_compactor.py)

Problem: Long agent sessions bloat the context window, slowing prompt prefill and dramatically driving up per-turn token costs.

Mechanics: Extracts key factual points (goals, file paths, decisions, errors) into a compact [HIERARCHICAL_MEMORY_STATE] block once token limits are crossed, keeping only recent turns in full text.

Benefit: Saves 65–80% of context tokens in long sessions without losing critical project state.

Cost & Trade-off: The compaction run incurs a one-time cost of ~1,000 tokens; exact code snippets or fine details from early messages may be lost in the summary.

When to disable & thresholds: Disable if an agent must reference verbatim details from very early messages in the same session. | Thresholds: Token Threshold: < 3000 = discards details too early | > 24000 = high prompt costs (Default: 6000). Retained Recent Turns: Default 4.

93. SQLite WAL Connection Pool & Async Write Batching (backend/async_write_batcher.py)

Problem: Hundreds of individual SQLite INSERT statements during concurrent agent runs cause disk I/O spikes and 'database is locked' errors.

Mechanics: Collects telemetry and audit events inside a thread-safe in-memory ring buffer and writes them in batches via executemany() under WAL mode.

Benefit: Reduces disk write cycles by roughly 80% and keeps the main event loop unblocked.

Cost & Trade-off: In the event of an abrupt power outage or kill -9 before the next flush interval, in-memory buffered records from the last 1–5 seconds are lost.

When to disable & thresholds: Disable if every single log entry must be written synchronously to disk with zero latency. | Thresholds: Batch Size: < 5 = frequent disk writes | > 100 = greater data loss risk on crash (Default: 25). Flush Interval: Default 2.0s.

14. Modern Memory & RAG Architecture

94. Hybrid Search with BM25 FTS5 & Reciprocal Rank Fusion (backend/hybrid_search_engine.py)

Problem: Pure vector search locates semantically related passages but frequently misses exact error codes, function

signatures, or serial numbers.

Mechanics: Combines SQLite FTS5 full-text ranking (BM25) and cosine vector similarity via Reciprocal Rank Fusion (RRF).

Benefit: Unifies exact keyword matches and semantic relevance into a single high-precision result list.

Cost & Trade-off: Querying dual indices increases search latency by ~15–40 ms; overly large top-k values (> 10) flood agent prompts with thousands of surplus tokens.

When to disable & thresholds: Seldom disable — at most lower top-k to 3–5 if token overhead per query must be kept to a minimum. | Thresholds: RRF Constant K: Default 60. BM25 Weight: 0.0 = vector only | 0.5 = balanced | 1.0 = keyword only (Default: 0.50). Top-K: Default 5.

95. Local Knowledge Graph / Triple Store (backend/knowledge_graph_store.py)

Problem: Flat text chunks miss structured dependencies (such as which entity manages which system or which component invokes which service).

Mechanics: Extracts and stores subject-predicate-object triples with confidence scores in SQLite, enabling multi-hop graph traversals.

Benefit: Agents identify system and organizational dependencies across multiple tiers.

Cost & Trade-off: Automated triple extraction incurs ~500 extra tokens during ingestion. Traversal depths > 2 cause neighbouring node counts to grow exponentially.

When to disable & thresholds: Disable auto-extraction in short-lived chat scenarios without persistent knowledge capture; cap traversal at 2 hops. | Thresholds: Max Hops: 1 = direct neighbors | 2 = optimal | >= 3 = node explosion! (Default: 2). Min Confidence: Default 0.70.

96. Agentic Corrective RAG / CRAG (backend/corrective_rag.py)

Problem: Traditional RAG forwards irrelevant retrieved passages to the language model without filtering, provoking hallucinations and incorrect answers.

Mechanics: Evaluates every retrieved document chunk against a relevance threshold (0.45) and triggers an automatic query rewrite with synonym expansion when matches score poorly.

Benefit: Filters out knowledge noise before answer generation and elevates retrieval precision on ambiguous prompts.

Cost & Trade-off: Each query reformulation consumes 1 additional LLM call (+1.5 to +4 s latency). Setting the relevance threshold too high (> 0.70) causes CRAG to run unnecessary retrieval loops.

When to disable & thresholds: Disable if the knowledge base is compact and curated, or if sub-2-second response times are required. | Thresholds: Relevance Threshold: < 0.30 = admits poor sources | > 0.70 = endless search loops (Default: 0.45). Max Rewrites: Default 2.

97. 2-Tier Core Memory Modeled after MemGPT/Letta (backend/core_memory_manager.py)

Problem: Important user preferences and project goals are lost as soon as they fall out of the ephemeral chat window.

Mechanics: Permanently anchors editable pinned working memory blocks (<CORE_WORKING_MEMORY>) in the system prompt, separated from large archival storage.

Benefit: Guarantees permanent core memory maintained by the agent itself across any number of sessions.

Cost & Trade-off: Every pinned block permanently consumes prompt tokens on every single LLM call (10 blocks × 800 characters ≈ 2,000 tokens baseline load per message).

When to disable & thresholds: Disable or reduce block count to 3–5 when using small models with narrow context windows or tight token budgets. | Thresholds: Max Blocks: > 15 = high token baseline consumption (Default: 10). Max Chars/Block: < 300 = truncates rules | > 1500 = clogs prompt (Default: 800).

98. Automated Citation & Source Pinning (backend/citation_pinner.py)

Problem: Without direct source citations, users cannot verify which document an agent's factual statement originates from.

Mechanics: Maps response sentences to retrieved source documents via token overlap, inserts inline citations ([Source: ID]), and verifies the minimum citation ratio.

Benefit: Makes every statement audit-proof and traceable, immediately exposing unsubstantiated claims.

Cost & Trade-off: Increases output token consumption by ~10–15%; an overly high minimum coverage (> 80%) leads to false warnings on creative or summary queries.

When to disable & thresholds: Disable during open brainstorming, code generation, or casual daily chat without document context. | Thresholds: Min Coverage: < 30% = unsubstantiated statements | > 85% = false warnings on summaries (Default: 60%).

15. Reliable Execution & Determinism (Durable Execution)

99. Structured Outputs & JSON Schema Guarantee (backend/structured_output_enforcer.py)

Problem: Language models sometimes return invalid JSON (markdown code blocks, missing brackets, or incorrect data types), causing downstream code pipelines to crash.

Mechanics: Automatically repairs syntax errors in raw strings and enforces all required fields and data types of the provided target schema.

Benefit: Guarantees 100% machine-readable, type-safe JSON payloads between agents and backend interfaces.

Cost & Trade-off: Each auto-repair loop adds +1 to +3 seconds of latency and extra tokens. Strict schema validation also rejects responses due to harmless extra fields.

When to disable & thresholds: Disable strict validation if API endpoints or scripts can handle flexible or partial JSON fields. | Thresholds: Schema Validation: Strict vs. Tolerant (Default: Strict). Max Repairs: 1–2 = fixes 98% of all syntax errors | > 3 = wastes tokens (Default: 2).

100. Durable Execution Checkpointing / Pause & Resume (backend/durable_checkpoint_manager.py)

Problem: If a 10-step research or build job aborted at step 8 due to a reboot, all 7 previous steps previously had to be repeated at high cost.

Mechanics: Transactionally saves complete variable and progress state to SQLite after each completed step.

Benefit: Interrupted long-running tasks resume precisely from the last valid checkpoint without duplicating work or tokens.

Cost & Trade-off: Checkpointing at every single step (N = 1) produces noticeable SQLite write load. Auto-resume on boot can unintentionally restart stale tasks from the previous day.

When to disable & thresholds: Disable auto-resume if interrupted tasks should be manually inspected and approved after a reboot. | Thresholds: Every N Steps: 1 = fully crash-proof, higher disk I/O | 3–5 = spares SSD (Default: 1). Retention: Default 14 days.

101. Idempotency Keys & Tool Call Deduplication (backend/idempotency_guard.py)

Problem: During network retries or agent loops, the same email, invoice, or webhook action risks being triggered multiple times.

Mechanics: Assigns a SHA-256 idempotency key in the SQLite ledger for each mutating tool call and blocks identical subsequent calls within the TTL.

Benefit: Reliably prevents duplicate external side effects, modeled after modern payment gateways (Stripe pattern).

Cost & Trade-off: If the user intentionally wants to execute the exact same command a second time within the lockout period (default 24 h), the guard blocks it.

When to disable & thresholds: Lower TTL hours to 1 h or briefly disable if identical actions must be executed repeatedly during testing. | Thresholds: TTL (Hours): < 1h = short duplicate protection | > 48h = blocks intentional repetitions (Default: 24h).

102. Dry-Run / Shadow Mode for Agent Jobs (backend/dry_run_simulator.py)

Problem: Teams often want to test new or complex agent jobs safely against production data before live execution.

Mechanics: Intercepts all mutating file, database, and API commands in shadow mode, logging a detailed mutation preview plan instead.

Benefit: Enables safe dry runs and risk audits before final approval.

Cost & Trade-off: WHILE ACTIVE, NO ACTUAL FILES ARE WRITTEN AND NO ACTUAL APIS ARE EXECUTED! If you forget this switch, the system appears externally 'dead'.

When to disable & thresholds: Keep strictly DISABLED in normal production! Enable only for dry runs and security audits. | Thresholds: Master switch default OFF (false). Activate only for isolated test runs.

103. Dead-Letter Queue (DLQ) with Auto-Replay Manager (backend/dead_letter_queue.py)

Problem: Failed webhooks or asynchronous background jobs are lost without a trace during brief third-party outages without a fallback buffer.

Mechanics: Saves each failed job along with original payload and error cause in the DLQ table and retries it with exponential backoff.

Benefit: No job is lost; failed tasks can be replayed individually or in batches at any time.

Cost & Trade-off: For permanently failing jobs (e.g., invalid API key), retry attempts waste resources; delayed replays may trigger actions later than intended.

When to disable & thresholds: Disable auto-replay if failed jobs should only be reviewed manually and approved individually. | Thresholds: Max Retries: 1 = aborts on brief outages | > 5 = accumulates dead jobs (Default: 3). Backoff Base: Default 10s (10s -> 20s -> 40s).

16. Security, Data Privacy & Enterprise Guardrails

104. Bidirectional PII Scrubber & Re-Hydration (backend/pii_scrubber.py)

Problem: Transmitting plaintext emails, phone numbers, or IBANs to external cloud LLMs risks GDPR violations and unintentional data leaks.

Mechanics: Replaces sensitive personal data locally with placeholders ([EMAIL_1], [IBAN_1]) before the API call and restores them to their actual values in the incoming response.

Benefit: Cloud models never see real personal data, while the local user still receives complete documents and answers.

Cost & Trade-off: If the LLM slightly alters the placeholder in the response text, re-hydration fails. Additionally, regex masking can occasionally corrupt code strings.

When to disable & thresholds: Disable if using exclusively local models (Ollama) or if the LLM strictly needs to analyze real email domains/names. | Thresholds: Granular switches for emails, phone numbers, IBANs, and automatic re-hydration (all enabled by default).

105. Egress Firewall & Domain Allowlisting for Agents (backend/egress_firewall.py)

Problem: Via hidden prompt injections on websites, agents could be instructed to send internal data via HTTP requests to third-party attacker servers.

Mechanics: Checks every outgoing target URL against the configured domain allowlist and blocks unauthorized hosts as well as raw IPv4 addresses.

Benefit: Prevents data exfiltration at the network level, even with manipulated inputs.

Cost & Trade-off: As long as allowlist_only is active, any new website, API, or documentation not explicitly in the domain list is hard-blocked during web research.

When to disable & thresholds: Disable (or add required domains) if the research agent should research freely on the open web. | Thresholds: Allowed Domains: Comma-separated whitelist. Raw IP blocking prevents direct IP exfiltration.

106. Granular Agent RBAC & File-Scope Jails (backend/agent_rbac_jail.py)

Problem: If every subagent has full write access to the entire project directory, a frontend agent could accidentally

overwrite backend kernel files or .env keys.

Mechanics: Binds each agent role to a fixed tool list and an allowed directory jail (e.g., static/ for frontend agents).

Benefit: Enforces the principle of least privilege for every individual process in the operating system.

Cost & Trade-off: Prevents an administrator agent from legitimately maintaining configuration files outside the project directory or directly inspecting .env templates.

When to disable & thresholds: Loosen only briefly for dedicated system maintenance tasks; .env protection should always remain active during normal operations. | Thresholds: Switches for path jail, .env lock, and system directory lock (/etc, /root, /proc).

107. Deterministic Output Linter & Policy Checker (backend/output_policy_linter.py)

Problem: Language models occasionally lapse into artificial AI clichés or generate risky shell snippets despite system prompts.

Mechanics: Scans generated text in real time using regular expressions, removes sycophantic openers/AI slop, and defuses destructive commands.

Benefit: Ensures linguistic precision and technical safety without an additional LLM verification call.

Cost & Trade-off: The destructive command filter can also block harmless cleanup scripts or intended database migrations if commands such as DROP or rm -rf appear in them.

When to disable & thresholds: Briefly disable the destructive blocker when intentionally creating server cleanup scripts or SQL reset migrations. | Thresholds: Switches for removing AI clichés, blocking destructive commands, and auto-cleanup.

108. Tamper-Proof Execution Receipts / SHA-256 Receipts (backend/execution_receipt_ledger.py)

Problem: For compliance audits (EU AI Act / ISO 42001), plain text logs are insufficient because they can be altered retroactively.

Mechanics: Chains every task billing record (model, tokens, USD cost, executed tools) with the SHA-256 hash of the predecessor entry in a blockchain-like chain.

Benefit: Any retroactive tampering with cost or decision logs is immediately detected during integrity checks.

Cost & Trade-off: Causes minimal compute and storage overhead (~2 ms and ~2 KB SQLite storage per task); the hash chain grows continuously over months.

When to disable & thresholds: Can remain permanently active in production; disable only on extremely memory-constrained micro-systems. | Thresholds: Hash chaining (SHA-256) + detailed cost breakdown per run.

17. Self-Optimization, Evaluation & Swarm Intelligence

109. Automated Prompt Optimizer based on DSPy/TextGrad Principles (backend/prompt_optimizer.py)

Problem: Static system prompts repeat the same errors continuously until a human manually refines them.

Mechanics: Analyzes real error logs of a role, detects systematic weaknesses, and generates targeted guardrail rules for the system prompt.

Benefit: Agents learn continuously from failed runs and harden their instructions autonomously.

Cost & Trade-off: If auto-apply is active, a flawed inferred rule can permanently distort agent behavior (prompt drift) and bloat the system prompt.

When to disable & thresholds: Keep auto-apply disabled in production so that proposed prompt patches are only applied after human review. | Thresholds: Error Threshold: 1 = overreacts to isolated errors | > 8 = learns very sluggishly (Default: 3). Max Rules: > 10 = bloats system prompt (Default: 5).

110. Gold-Standard Regression Test Suite / Eval Runner (backend/eval_regression_runner.py)

Problem: A seemingly minor change to a system prompt or model can silently degrade existing security or formatting rules.

Mechanics: Executes automated gold-standard test cases before adopting new settings, verifying pass rates and

forbidden patterns.

Benefit: Prevents quality regressions in production through an objective quality gate.

Cost & Trade-off: Each complete eval run executes all configured test prompts against the LLM API (consuming ~5,000–25,000 tokens and 15–60 s runtime depending on test scope).

When to disable & thresholds: Disable deploy blocking when conducting rapid experimental prompt tests without wait times. | Thresholds: Min Pass Rate: < 75% = permits degradations | 100% = blocks on the slightest deviations (Default: 90%).

111. Multi-Armed Bandit Model Router (backend/bandit_model_router.py)

Problem: Fixed model assignments cannot respond if a cloud provider suddenly slows down or exhibits elevated error rates.

Mechanics: Continuously scores each model based on real success rate (60%) and speed (40%), dynamically selecting the best candidate.

Benefit: Maximizes response quality and speed automatically using real runtime telemetry.

Cost & Trade-off: Through the exploration rate (epsilon, e.g., 0.15 = 15%), the router deliberately explores alternative models periodically, which may occasionally be slower.

When to disable & thresholds: Disable (or set epsilon to 0.00) if a 100% fixed, predictable model must be enforced for every run. | Thresholds: Epsilon: 0.00 = strictly uses best model | > 0.30 = samples weaker models too often (Default: 0.15). Success/Latency Weight: Default 0.60 / 0.40.

112. Dynamic Ephemeral Team Spawning (backend/ephemeral_team_spawner.py)

Problem: A single generalist agent quickly reaches its limits on large cross-cutting tasks (architecture + UI + security + QA).

Mechanics: Recruits a temporary specialist task force matched to the mission briefing and disbands it once the mission is complete.

Benefit: Concentrates specialized knowledge on demand without permanent resource overhead.

Cost & Trade-off: A 4-member ad-hoc team multiplies token consumption (3x to 5x) and requires additional coordination overhead between team members.

When to disable & thresholds: Disable for simple standalone tasks or tight token budgets; limit max team size to 3–4 in day-to-day operations. | Thresholds: Max Team Size: 2–4 = ideal | > 5 = high communication overhead & 5x token consumption (Default: 4).

113. Live Resource Protector & Host Backpressure (backend/resource_backpressure.py)

Problem: Concurrent containers, browser audits, and local processes can overload host RAM or CPU during load spikes (OOM kill).

Mechanics: Monitors CPU, RAM, and disk metrics in real time and automatically throttles low-priority background tasks when load exceeds 85%.

Benefit: Reliably protects the server from overload crashes and keeps critical core processes responsive at all times.

Cost & Trade-off: If thresholds are set too low (CPU < 70% or RAM < 70%), background tasks are throttled during routine, brief load spikes.

When to disable & thresholds: Do not disable entirely — instead raise CPU/RAM thresholds to 90% on powerful hardware. | Thresholds: Max CPU/RAM: < 70% = throttles during normal workloads | > 92% = OOM risk (Default: 85%). Min Free Disk: Default 2.0 GB.

18. Pillar 3: Business & ERP (alpha) & Multi-Tenant Suite

Module / View	Functional Scope & Technical Features (v0.5.1)
Addresses & CRM + PocketBase Sync	Central contact, accounts receivable (10000+), and accounts payable master records (70000+), 360° contact drawer, multi-channel timeline, calendar/follow-ups, as well as LinkedIn & media sync via PocketBase (ports 3005/8090/8092).

Module / View	Functional Scope & Technical Features (v0.5.1)
Articles & Services (ART)	Master data catalog from ART-1001 for services, hourly rates, flat rates, SaaS subscriptions, and physical products with inventory tracking.
Transactions & Contracts	End-to-end document chain: Quote (ANG-YYYY-NNNN) → Order Confirmation (AB-YYYY-NNNN) including weighted deal pipeline, revenue forecast, and contract deadline radar (30/14/7-day alerts).
Accounting & DATEV	Accounts receivable and payable invoices compliant with Section 14 UStG, GiroCode payment QR, financial analytics, automated dunning engine, and EXTF/CSV export for DATEV.
Corporate & Tenant Isolation	Multiple internal corporate entities (company_id) with physically isolated SQLite databases in WAL mode, individual numbering sequences, letterheads, and template editor.

19. Architecture, Security & Deployment Features (v0.5.0 – v0.5.1)

Feature / Subsystem	Implementation & Operational Benefits (v0.5.1)
Modular Router Architecture	Extraction into dedicated FastAPI routers under archon/routers/ (main.py reduced from 16,254 to 1,875 lines, -88%) and central configuration layer archon/core/config.py.
Security Hardening (Archon Hardened)	Remediation of 25 critical and 68 high audit findings; Starlette threadpool relief, hardened UI DOM paths, and Class-A Docker admin proxy for hermes_updater.
RBAC User Management (17.10)	Granular roles (admin, operator, viewer) via backend/auth_service.py with 100,000 rounds of PBKDF2-HMAC-SHA256 password hashing and single-user/multi-user mode.
2-Phase System Update (17.11)	Immediate display of local version/HEAD via ?local_only=true (~0.25 s) plus asynchronous GitHub comparison, pre-update SQLite online snapshots, and 1-click rollback.
AnythingLLM Proxy & Chroma Stack	Seamless embedding of AnythingLLM into the Central Library via /anythingllm-proxy including /assets/* fast path and local ChromaDB vector index.
Portable 1-Click Installer	Cross-platform initial setup via setup.bat, setup.ps1, and install.sh with docker-compose.yml, .env.example, INSTALL_AGENT.md, and 9-language GitHub documentation.
4-Eyes Quality Gate & Code Sandbox	Automated secondary review in Kanban prior to ticket closure, alongside an interactive code sandbox in Showroom (4.4.1) for live preview of generated artifacts.

© Michael Keller

20. Glossary & Alphabetical Index

Alphabetical index of terms for all core modules, architecture components, and features (v0.5.1) with exact page numbers.

Feature / Term	Brief Description	Page(s)
3-Tier Cost Routing	Automatic model selection in LOW / MEDIUM / HIGH according to task complexity and budget.	p. 3, 8, 15
4-Eyes Quality Gate	Independent review of completed Kanban tasks by a reviewer agent.	p. 3, 13, 29
Anti-AI-Slop & STE100	Style and wording filter to avoid unnecessary filler phrases and save output tokens.	p. 8, 9
AnythingLLM Proxy	Local embedding of AnythingLLM + ChromaDB via /anythingllm-proxy in the Central Library.	p. 3, 29
Approval Gate (HITL)	Human review step before external emails, accounting entries, or critical file operations.	p. 3, 4, 16, 26
Archon Hardened (v0.5.1)	Security and stability release with modular router architecture and resolved audit findings.	p. 3, 29
Audit Log (SHA-256 Hash Chain)	Audit-proof logging chain of all system and agent actions.	p. 11, 16, 26, 27
Bandit Model Router	Adaptive router (multi-armed bandit) for dynamic model selection based on hit rate and cost.	p. 27, 28
Bot Screen (Web Navigator)	Visual execution monitor for browser interactions via Playwright.	p. 13, 21
ChromaDB	Local vector database for semantic document retrieval in Pillar 2.	p. 23, 29
Context Compaction & LRU-K	Memory swapping and compression of long chat contexts upon reaching RAM limits.	p. 8, 9, 22
Corporate & Multi-Tenancy	Physical database separation of multiple companies via company_id.	p. 28, 29
Corrective RAG (CRAG)	Self-verifying knowledge retrieval with relevance scoring prior to response generation.	p. 23, 24
DAG Scheduler	Two-tier execution planning for dependent subtasks with parallel fan-out / fan-In.	p. 12, 22
DATEV Export & GiroCode	EXTF/CSV interface for accounting data and EPC payment QR codes on invoices.	p. 3, 28, 29
Dead-Letter Queue (DLQ)	Error queue with automatic retry management for failed jobs.	p. 25, 26
Decision Engine (55 Frameworks)	Strategy module for structured boardroom deliberations across 9 methodology categories.	p. 3, 4, 18
DSPy / TextGrad Optimizer	Automated optimization of system prompts based on real evaluation metrics.	p. 27
Durable Execution & Time Travel	Interruptible long-running tasks with SQLite checkpoints and step-by-step execution replay.	p. 3, 19, 25, 26
Egress Firewall	Outbound network and domain control for agent tool calls.	p. 26
Ephemeral Code Sandbox	Ephemeral, isolated container runtime for secure Python and script execution.	p. 3, 21, 29
ERP Document Chain (ANG -> AB -> RE)	Linked sales and accounting documents including pipeline forecast and dunning engine.	p. 3, 6, 16, 28

Feature / Term	Brief Description	Page(s)
Event Mesh & Webhook Triggers	Event-driven processing of incoming webhooks and directory changes.	p. 13, 19
Evey / Hermes Plugins	Modular system extensions for storage, delegation, verification, and interfaces.	p. 3, 6, 18, 19
Hermes Master Orchestrator	Central core engine for goal decomposition, delegation, and synthesis in the daemon on port 8420.	p. 10, 12, 18, 21
Hybrid Search (FTS5 + RRF)	Combination of BM25 full-text search and vector search via Reciprocal Rank Fusion.	p. 3, 23
Idempotency Keys & Dry Run	Protection against duplicate tool execution alongside simulation mode without write access.	p. 25
JEV Router (0-Token Intent)	Deterministic pre-filter with zero AI cost for unambiguous short commands.	p. 8, 9
Knowledge Graph	Local triple store for structured entity relationships alongside vector data.	p. 15, 23, 24
MCP (Model Context Protocol)	Standardized driver architecture for secure tool integration.	p. 5, 13
OmniRoute AI Gateway	Multi-provider gateway including free-tier routing, latency hedging, and eval arena.	p. 3, 5, 19
PBKDF2 / Argon2id RBAC	Role and password protection in the System Control Center (Tab 17.10).	p. 3, 26, 29
PII Scrubber & Re-Hydration	Bidirectional anonymization of sensitive personal data before external LLM calls.	p. 26
Pre-Flight Cost Estimator	Advance estimation of expected token costs including task budget abort on overruns.	p. 8, 10
Pre-Mortem & Consensus Gate	Simulated failure stress-testing and multi-agent voting prior to critical approvals.	p. 5, 19, 21, 22
Research Lab	Multi-stage web and source research with evidence triangulation.	p. 3, 6, 19
Router Architecture (archon/routers)	Modular backend partitioning into lean FastAPI routers (-88% file size in main.py).	p. 3, 29
Semantic Caching	24h caching of semantically similar prompts to save tokens.	p. 8, 18, 22
SLA Contracts (Contract Enforcement)	Formal handoff and quality assurances between specialized agents.	p. 21, 22
Sleep-Cycle Consolidation	Nightly deduplication and long-term consolidation of agent memory.	p. 14, 21
SQLite WAL Connection Pool	Write-safe multi-threaded database access via Write-Ahead Logging.	p. 14, 19, 22, 23
System Control Center (11 Tabs)	Central administration console including user/RBAC management and 2-phase system update.	p. 3, 7, 8, 29
Token Guard & Loop Killer	Real-time protection against infinite loops (Jaccard similarity check) and token overruns.	p. 3, 6, 8, 22
Watchdog & Deadlock Supervisor	30-second monitoring of hung tasks, containers, and automated self-healing.	p. 5, 10, 11, 19

© Michael Keller