本地大模型 Agent Harness 控制变量实验报告

Qwen 3.6 / Qwen 3.8 · Hermes Agent · DeepSeek Harness (DSH)

性能、可靠性、工具行为与自治协作的客观比较

实验日期：2026-08-18 至 2026-08-20

版本：External-share / Privacy-redacted

隐私说明 本报告将真实实验载荷统一匿名为“Repo-R（私有大型 Python 仓库）”。不包含项目名称、业务目标、领域对象、源文本、私有路径或产品路线。保留的仅是理解 agent/harness 差异所必需的工作负载类型、规模、测试结果和系统指标。

1. 执行摘要

本实验不是为了评估某个私有项目本身，而是用一个真实、历史包袱较重、测试规模超过千项的 Python 仓库作为稳定载荷，逐步只改变一个变量，观察本地模型、推理配置、agent harness 与自治编排方式的差异。最终形成了从“单 agent 问答”到“架构 agent + 实现 agent + 确定性控制器 + 独立复核”的完整实验链。

最重要结论 不存在一个工具在所有任务上全面胜出。Hermes 在交互式代码实现、短闭环修复和进度可见性上更适合作为执行器；DSH 在只读架构审计、边界论证和独立复核上更适合作为 architect/auditor。二者使用同一模型时，优势来自 harness 行为和程序化隔离，而不是模型级独立性。

- 模型层：Qwen 3.8 的过程可见性和推理显式性更强，但在相同硬件上 wall time 并没有稳定优于 Qwen 3.6；reasoning=medium 会显著增加生成 token 和等待时间。
- 推理层：单卡 27B FP8 的主要瓶颈是 decode，不是 CPU、工具或常规网络。生成期 GPU 99% 且触及 300W 功耗墙，典型 decode 约 90–115 tok/s，部分受控探测约 145 tok/s。
- 流式层：stream interval 从 1 改为 4 后，5 次重复实验的平均 wall time 仅从 42.073s 变为 42.110s（+0.09%），但 chunk 数从 1360 降到 351（-74.2%）。因此较大的 interval 可明显降低传输/渲染事件数量，而几乎不影响完成时间。
- harness 层：同模型三项直接对照中，Hermes 在“错误前提恢复”的只读调查上快约 3.17×，在单文件 exact-write 回归修复上快约 2.56×；DSH 在双 wrapper preflight-guard 回归上快约 2.37×。任务形态比“工具品牌”更能解释速度差异。
- 状态层：同一 Qwen 3.8 medium 的重复健康检查，在 skills/memory 被触发后由 218.6s 降到 52.1s（约 4.2×）。这说明 agent 的持久技能、记忆、profile 与缓存状态可以比模型版本本身产生更大的生产力差异。
- 自治层：最稳定的组合不是让单 agent 全权执行，而是 DSH 做只读架构决策，Hermes 在 disposable worktree 中执行，controller 做硬边界/测试/字节一致性验证，再由 DSH 独立 post-audit。真实恢复任务与首个清理批次均完成闭环。

1.1 推荐的工具分工

任务类型	首选	理由	注意事项
交互式代码实现 / 小范围修复	Hermes	动作快、工具反馈清晰、适合在隔离 worktree 中自主实施	需由 controller 限制可写边界并复验
架构理解 / consumer graph / 风险审计	DSH	更适合作为只读、证据驱动的 architect/auditor	广范围调查可能明显慢于 Hermes
高置信度自治软件工程	DSH → Hermes → Controller → DSH	把“判断、实现、确定性校验、独立复核”拆开	同模型不等于独立模型，应理解为程序化独立
纯模型版本 A/B	重置 harness 状态后再测	避免 skills/memory/profile/cache 把模型差异淹没	必须冻结 prompt、repo commit、服务配置、reasoning 与工具集

2. 实验设计与控制变量方法

2.1 实验载荷（已匿名）

Repo-R 是一个真实大型 Python 软件仓库，具有长期迭代、脚本化工作流、持久化约束、架构文档与超过千项测试。实验任务只暴露为通用软件工程问题：主机健康检查、只读架构追踪、性能瓶颈诊断、错误前提恢复、最小回归修复、执行前置守卫修复和自治代码清理。报告不保留其产品目的。

载荷特征	实验时观察到的量级
仓库规模	约 1.1k tracked files；数百个 source/script/test/doc 文件
回归规模	完整测试套件约 1,067–1,077 tests（随清理批次变化）
代码任务	1–2 个源文件的最小修复，以及低风险 bounded cleanup
分析任务	需要在当前 authority 与历史/兼容代码之间做区分
正确性信号	git diff、完整 pytest、文档一致性检查、固定 artifact 的 byte-identity、read-only 状态不变

2.2 控制变量原则

1. 同一硬件主机、同一 GPU 和同一 OpenAI-compatible 本地端点作为推理基础。

2. 比较模型版本时保持任务定义相同或高度同构；当 reasoning、skills/memory 或服务状态改变时单独标记，避免伪装成纯模型 A/B。

3. 比较 Hermes 与 DSH 时固定同一底层模型（后期为 Qwen 3.8 27B FP8）、同一 Repo-R commit/测试、同一任务目标，并使用隔离 worktree。

4. 读任务必须保持 git status 不变；写任务限定为最小 diff，并由 benchmark-supplied regression test 验证。

5. 所有关键运行以 /usr/bin/time、tool-call 统计、server log/API probe、git diff/status 和测试结果组成证据链。

2.3 可比性分级

等级	含义	本报告示例
A：严格同任务/同模型/不同 harness	最适合直接判断 harness 差异	Hermes vs DSH 三项 Repo-R 对照
B：同任务/模型配置变化	可判断性能与行为趋势	Qwen 3.6 → 3.8；reasoning=medium
C：同模型但 agent 状态变化	用于评估生产环境“越用越熟”的收益，不应归因于模型	Qwen 3.8 medium repeated runs + skills/memory
D：微基准	只判断单一推理/传输参数	stream interval 1 vs 4

3. 实验硬件与软件环境

3.1 主机硬件

组件	记录值	解释
操作系统	Ubuntu 24.04；kernel 7.0.0-28-generic	实验期健康检查中直接观察
CPU	32 个可见计算核心；实验负载下 CPU 约 5%	具体 CPU 型号未在保留日志中记录，因此不推测
内存	约 186 GiB；典型可用 170–171 GiB；swap 基本未用	明显不是 agent 延迟瓶颈
GPU	1× NVIDIA RTX PRO 6000 Blackwell	单卡运行 27B FP8 模型

组件	记录值	解释
	Max-Q	
GPU 显存	97,887 MiB（约 96 GiB）；典型使用约 53.4 GiB	与 mem_fraction_static=0.55 一致
GPU 驱动 / CUDA	Driver 580.173.02 / CUDA 13.0	来自只读健康检查
GPU 功耗	最高约 300W；生成时经常 299–300.8W	decode 时触及功耗墙
磁盘	实验期约 62% 使用率，约 108–109GB 空闲（根分区记录）；另一次 /data 记录约 500GB 级容量	空间不是实验瓶颈

3.2 推理服务器

配置项	实验值
Serving engine	SGLang 0.5.17
Endpoint	127.0.0.1:8000，OpenAI-compatible /v1
Tensor parallel	tp=1
模型（早期）	Qwen/Qwen3.6-27B-FP8
模型（后期）	Qwen/Qwen3.8-27B-FP8
Context length	131,072 tokens（max request input ≈131,066）
Static memory fraction	0.55
KV pool	约 151,801 tokens（一次观测）
Chunked prefill	2,048 tokens；max_prefill_tokens 16,384
Max running requests	48
Speculative decoding	NEXTN / EAGLE；4 steps；5 draft tokens；topk=1
Observed spec acceptance	avg_spec_accept_length ≈3.26；流式实测 ≈3.37 token/chunk（一次受控阶段）
Tool-call parser	qwen3_coder
CUDA graph	enabled（server log 中持续为 True）
Mamba / SSM	radix cache extra_buffer；SSM float32；max_mamba_cache_size=25；track_interval=256
Prefix-cache related	disable_chunked_prefix_cache=true（实验阶段记录值）
Long-context probe	约 80,068 input tokens → HTTP 200；约 18.7s；sentinel 成功恢复

3.3 辅助服务与本地工具环境

服务/工具	实验值	对端到端时延的意义
Firecrawl	localhost:3002；一次本地调用约 92ms	不是模型 agent 的主要瓶颈
SearXNG	localhost:8888；典型约 1.2s	外部检索可见但通常远小于长 decode
Open WebUI	localhost:3000	交互界面，不参与核心 harness 对照
SGLang process lifecycle	早期由 SSH/PTY 启动，缺少稳健 supervisor 与持久日志	属于可靠性风险，而非模型质量差异
本地 shell/file tools	多数单次 <100ms	与几十至数百秒 LLM wall time 相比可忽略

3.4 Agent / harness 版本

组件	实验中记录的版本/状态	说明
Hermes（早期直接 benchmark）	Hermes Agent v0.16.0 (2026.6.5), upstream 36ae9584	Python 3.11.15；OpenAI SDK 2.24.0
Hermes（后期自治实验）	v0.20.4	作为 disposable worktree 内的 implementation agent；启用 --yolo

组件	实验中记录的版本/状态	说明
DeepSeek Harness (DSH)	@deepseek-ai/dsh 0.1.0-rc.7	作为 architect / forensic auditor；reasoning=medium
DSH protocol compatibility	supportsDeveloperRole=false（本地兼容补丁）	用于适配当时 Qwen/SGLang 对 developer role 的拒绝行为
Reasoning setting	后期固定 medium，并从 DSH request header 验证	避免“配置声称 medium、实际未生效”
Hermes profiles	default / tv_science / 其他 profile 曾并存；后期自治使用 tv_science 派生临时 profile	profile 状态是重要实验变量

配置教训 实验早期曾发现 harness 把 model.max_tokens=16384 误当作 context window，而服务端实际是 131072；这会让自动压缩失败并造成长会话膨胀。后续比较因此必须把“模型上下文长度”和“单次 completion 上限”分开显式记录。

4. 完整实验流程（按控制变量演进）

阶段	唯一主要变化	核心任务	目的
P0	Qwen 3.6 baseline	天气/主机健康/仓库静态分析/性能诊断/长期状态	建立功能与性能基线
P1	模型切换到 Qwen 3.8	重复同类任务	观察模型版本带来的行为与耗时变化
P2	Qwen 3.8 + reasoning=medium	重复健康、代码分析、性能诊断	量化显式 reasoning 的成本与收益
P3	保持 3.8 medium，重复运行	相同 Task 2/3	观察 skills/memory/profile 熟化效应
P4	只改 streaming interval	固定输出，5× interval=1 vs 5× interval=4	隔离传输粒度的性能影响
P5	固定模型，比较 Hermes vs DSH	读分析 + 两类代码回归	直接比较 harness 行为
P6	角色拆分自治	DSH architect → Hermes implement → deterministic controller → DSH audit	验证组合式自治是否比单 agent 更稳健

4.1 P0：Qwen 3.6 baseline

基线覆盖了六类任务，其中保留了 Task 1、2、3、5、6 的完整统计。Task 4 的聊天端展示限制使其结果不适合作为结构化对照，因此没有纳入速度汇总。

任务	类型	Wall time	Tool calls	结果
1	外部信息检索 / weather	53.219s	13	PASS
2	本机只读健康检查	44.898s	14	PASS
3	Repo-R 静态架构分析	125.648s	22	PASS
5	agent 响应瓶颈诊断	163.0s	26	PASS
6	长期主机运行条件审计	207.5s	55	PARTIAL（需求完成但发现高风险运维项）

这一阶段首次明确：单卡 27B FP8 的主要时延来自 decode；CPU、RAM、本地工具调用均有较大余量。与此同时还发现了端口期望与实际服务端口不一致、模型服务未守护、Hermes context-length 配置误判等环境风险。

4.2 P1：Qwen 3.8 替换模型，其他基础设施保持不变

任务	Qwen 3.6 baseline	Qwen 3.8 initial	变化
Task 1	53.2s	120.4s	3.8 更慢；过程说明更丰富
Task 2	44.9s	66.2s	3.8 更慢
Task 3	125.6s	392.4s	3.8 明显更慢且进行了更广泛调查
Task 5	163.0s	380.1s	3.8 更慢
Task 6	207.5s	405.5s	3.8 更慢但最终为 PASS

原始 wall time 不支持“3.8 比 3.6 更快”的结论。相反，早期 3.8 在多个任务上更慢。主观观察是 3.8 更愿意展示自己正在做什么，并倾向更广泛地并行收集证据；这会提高过程透明度和覆盖面，也会增加总调用量和时长。

4.3 P2：Qwen 3.8 + reasoning=medium

medium 推理配置在 API 返回中真实产生 reasoning tokens。一个 400-token 受控输出中，219 tokens（约 55%）属于 reasoning；因此“更慢”至少有一部分是可解释的推理预算，而不是 harness 阻塞。

任务	3.8 initial	3.8 medium	观察
Task 2	66.2s	249.3s	medium 阶段调查更深入；不宜只按速度排名
Task 3	392.4s	168.7s	medium 反而更快，说明会话/工具路径差异很大
Task 5	380.1s	386.7s	基本同量级；decode + reasoning 仍主导

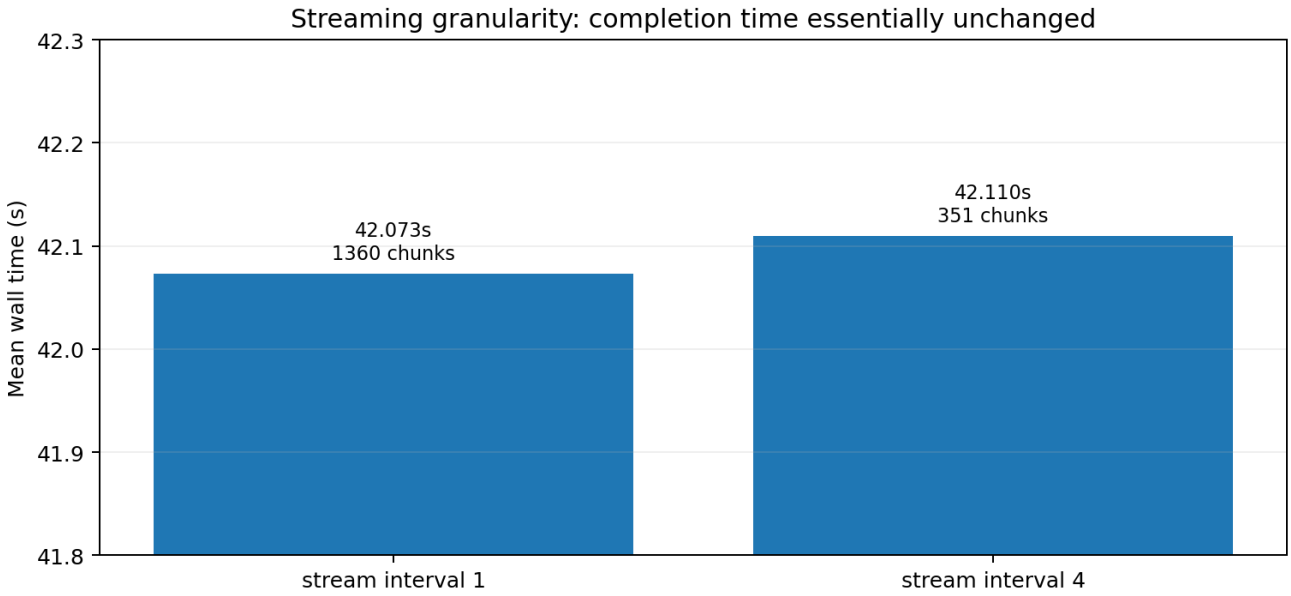
4.4 P3：同为 3.8 medium，但 skills/memory 被复用

这一阶段最能说明为什么“生产 agent 性能”不能只按模型版本解释。Task 2 的重复实验从 218.633s 降到 52.116s，约 4.20×；用户侧同时观察到 Hermes skills 被触发，且长期运行使 skills/memory 更新。Task 3 的两轮则保持在 161–169s。

重复任务	Round 1	Round 2	相对变化
Task 2	218.633s	52.116s	约 4.20× 加速
Task 3	161.312s	169.066s	基本稳定（Round 2 +4.8%）

解释边界 Task 2 的 4.2× 加速是“agent 系统熟化”证据，不是纯模型能力提升证据。若要做模型论文式 A/B，应清空/冻结 skills、memory、session、profile 与 prefix cache；若要评估真实生产效率，则这些状态恰恰应该被保留。

4.5 P4：stream interval 微基准



每种 interval 重复 5 次，输出字符数固定为 19,402。interval=1 平均 wall 42.073s、TTFT 52.4ms、1,360 chunks；interval=4 平均 wall 42.110s、TTFT 53.0ms、351 chunks。wall time 差仅 +0.086%，chunk 数减少 74.2%。

微基准结论 在这组输出长度和本地网络环境中，stream interval=4 是更高效的传输粒度：几乎不牺牲首 token 或总完成时间，却显著减少事件数量。interval=1 的优势主要是更细粒度的 UI 动画，而不是吞吐。

5. 模型与推理服务器性能分析

5.1 Decode 是主瓶颈

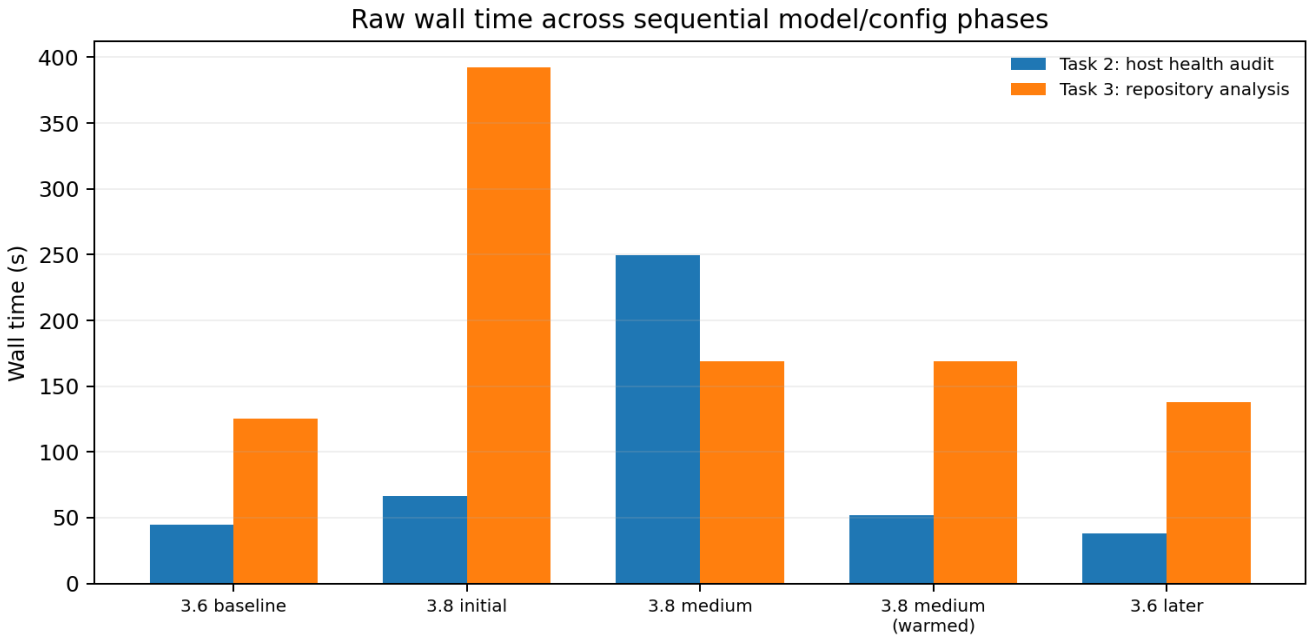
指标	实测范围/样本	结论
Decode throughput	典型 ~90–115 tok/s；部分单探测约 145 tok/s	决定多轮 agent 的主要 wall time
GPU utilization	生成阶段接近 99%	GPU 计算资源充分利用
GPU power	299–300.8W，接近 300W 功耗上限	存在明显功率限制
SM clock	一次生成期约 2242/3090 MHz（~72% 峰值）	与功耗墙一致
CPU	约 5%，32 cores 可见	不是当前瓶颈
RAM	约 170 GiB free	不是当前瓶颈

5.2 Prefill 与缓存

冷 prefill 常见约 5.2k–8.4k tok/s，远快于 decode。前缀命中时提升巨大：受控重放中出现 5.4k 前缀约 15×、26.9k 约 55×、54.9k 约 47.6×的加速。但另一次 80k 前缀测试几乎没有二次加速，说明长前缀缓存行为受当时 SGLang/Mamba/chunked-prefix 配置影响，不应简单假设“cache 永远有效”。

5.3 Reasoning 的成本

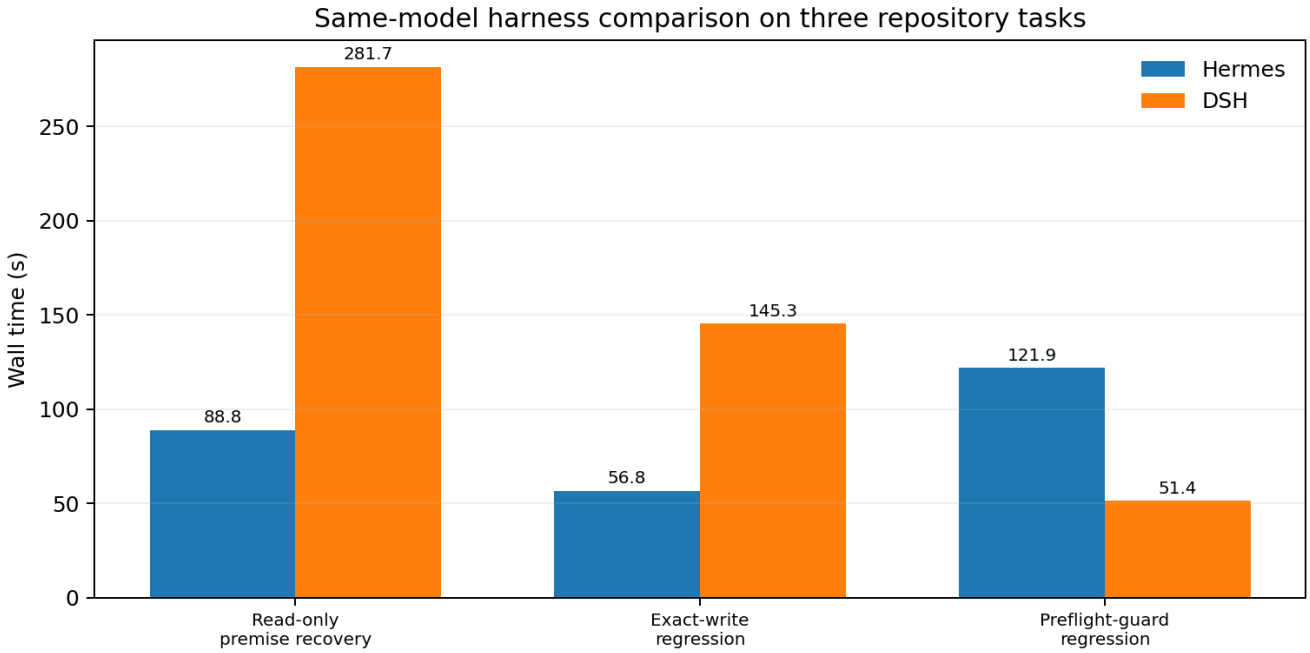
reasoning=medium 的价值是让模型在复杂诊断和代码任务上更显式地分配推理预算；成本则是生成 token 变多，而 decode 恰好是系统主瓶颈。在一次受控 400-token 输出里，reasoning 占 219 tokens（约 55%）。因此 medium 很适合架构审计和根因分析，但不应默认用于所有快速健康检查。



图中的跨阶段 wall time 是原始观测，不是“纯模型排名”。后期 skills/memory、profile、缓存与服务状态都发生过可记录变化，因此只能用于展示真实生产系统中的波动范围。

6. Hermes 与 DSH：同模型直接对照

后期直接对照固定为同一 Qwen 3.8 27B FP8 本地模型和相同 Repo-R 任务。两者都是“同模型不同 harness”，因此这是本报告最有价值的工具比较。



任务	Hermes	DSH	速度结果	正确性结果
错误前提恢复：只读追踪当前真实执行链	88.76s	281.70s	Hermes ~3.17× faster	二者均识别 supplied premise 不成立并恢复到当前路径
Exact-write invariant regression：单文件一行根因修复	56.77s	145.35s	Hermes ~2.56× faster	二者得到相同一行修复；DSH 还跑 full suite 1076 tests
Preflight-guard regression：两 wrapper 的对称错误	121.92s	51.35s	DSH ~2.37× faster	二者均发现不是单点，而是两个 wrapper 的 guard 被交换

6.1 Hermes 的客观优势

- 执行型任务速度更稳定：单文件最小修复显著快于 DSH，并且能快速进入 patch/test 循环。
- 交互过程可见性更好：会明确显示 read/search/terminal/patch 等动作，适合人类实时监督。
- 在 disposable worktree + YOLO 模式下，适合作为“受边界约束的实现 agent”。
- skills/memory 能对重复运维任务产生巨大提速，说明它作为长期驻留执行器有明显生产价值。

6.2 Hermes 的客观弱点

- 状态依赖明显：同一任务在不同 profile/session/skills 状态下耗时差可以达到数倍。
- 早期 v0.16.0 出现过 context-length 配置误判、过时 .env TERMINAL_CWD 警告，以及一次 Path.expanduser() 未捕获 RuntimeError 导致工具批中断。
- “会做很多”不等于“架构边界一定正确”；自动写操作必须外置 hard guard。
- Hermes v0.16.0 与后期 v0.20.4 之间没有单独做版本-only A/B，因此不能从本实验声称 v0.20.4 在速度/质量上定量优于 v0.16.0。

6.3 DSH 的客观优势

- 适合 forensic / architecture 模式：倾向明确给出 observed fact、inference、authority、legacy 的区分。
- 在双 wrapper preflight 回归中非常快，并正确识别“同一回归在两个 active wrapper 对称出现”，说明结构性模式识别很强。
- reasoning effort 可通过保存的 request/header 独立验证，便于做审计型实验。
- 作为 read-only post-auditor 时能检查实际 diff、测试和 artifact，而不是只接受执行 agent 的自述。

6.4 DSH 的客观弱点

- 广范围只读 repo 调查明显可能更慢：错误前提恢复任务耗时约为 Hermes 的 3.17×。
- 单文件 exact-write 回归也比 Hermes 慢 2.56×，表明其更偏“证明充分”而非最短闭环。
- 当时需要一处 protocol compatibility patch（禁用 developer role）才能稳定适配该 Qwen/SGLang 组合。
- 与 Hermes 使用同一模型，因此它作为 auditor 提供的是程序流程独立，而不是模型认知独立。

6.5 内存占用

任务	Hermes max RSS	DSH max RSS	解释
错误前提恢复	160,932 KB	191,552 KB	DSH 更高约 19%
Exact-write regression	171,892 KB	189,820 KB	DSH 更高约 10%
Preflight-guard regression	181,108 KB	181,888 KB	基本相同

二者 agent 进程自身的 RSS 都在约 160–192MB 量级，与 GPU 模型服务占用相比可以忽略；性能差异主要不是 host RAM。

7. Harness 状态与调用形态：为什么同一工具也会差很多

7.1 Hermes v0.16.0 的两种调用形态

同一只读代码调查曾记录到两个非常不同的 Hermes wall time：一个运行约 99.50s，另一个本地直接运行约 924.48s。后者明确记录为 Hermes v0.16.0。由于两次运行的 session/profile/memory/cache 条件不能证明完全相同，本报告不把“9.3×”解释为工具版本差异；它更像是一个警告：agent harness 的会话状态、压缩、工具初始化和 profile 会主导端到端时延。

7.2 Skills / memory 是生产特性，也是 benchmark 混杂因素

重复任务发生 4.2× 提速时，用户同时观察到 skills 被触发，而且长期任务执行让 skills/memory 更新。对于真实生产，这是优势；对于模型 A/B，这是污染变量。两种评价体系应分开：

评价目标	应如何处理 skills/memory
纯模型能力比较	固定/清空/禁用，确保每次从等价状态开始
真实长期 agent 生产力	允许累积，并把“熟化速度、失效风险、可迁移性”作为被测指标
自治可靠性	允许 agent state，但关键 gate 必须由外部 controller 重新验证

8. 从单 Agent 到双角色自治：最终实验

8.1 最终角色设计

组合 DSH（Architect / Forensic Auditor，read-only）→ Hermes（Implementation Agent，disposable worktree）→ Deterministic Controller（测试、diff、artifact、Git boundary）→ DSH（Post-audit）。

这不是为了追求“两个 agent 投票”，而是把不同类型的错误放到不同责任边界。Architect 决定什么应该做；executor 只在授权范围实施；controller 不使用语言模型判断，而是用确定性规则检查；最后 auditor 重新读取真实 diff 与结果。

8.2 Recovery 闭环验证

Gate	结果
DSH pre-audit	PASS；确认补丁修复真实根因且未打开额外行为边界
Hermes verification	完成；没有额外 source commit
Controller independent execution	PASS
关键 artifact identity	修复前已生成的有效 artifact 与修复后结果 byte-identical
DSH post-audit	PASS
完整测试	1,077 tests + documentation check 通过
Git acceptance	只在全部 gate PASS 后 fast-forward 到 autonomous main

这一闭环证明：对真实软件工程任务，“LLM 判断 + LLM 实现”还不够。确定性 controller 才是让自治结果可接受的关键。

8.3 首个 repo reduction 批次暴露的控制器问题

DSH 提出一个低风险历史代码清理批次，Hermes 正确执行；controller 随后因为简单 grep 把“历史 lineage ID 字符串”误判为“live executable consumer”而 fail-closed。进一步审计证明这些只是 opaque lineage/schema reference，最终分类为 SAFE_LINEAGE_ONLY，完整测试 1,067 passed。

这一失败很有价值 它说明 deterministic controller 也会写错规则。正确做法不是放弃硬控制，而是把规则升级为“executable/import、runtime call、persisted schema、replay、opaque lineage、test-only compatibility、documentation”分类型判断。语言模型和规则系统都需要可审计的失败模式。

8.4 Controller 工程化经验

- 必须是 resumable state machine，而不是一次性 shell pipeline；agent call 已完成后，脚本重启不能重新花模型调用。
- 每个 gate 有持久状态：architecture complete → patch applied → executor complete → controller verified → post-audit → accepted。
- worktree 必须隔离；共享 venv 只提供依赖，源码用 PYTHONPATH=<current worktree>/src，避免 editable install 指向错误 checkout。
- Git patch identity、branch/head、dirty status、原始 repo 不变都应作为 hard invariants。
- 长阶段必须有终端/Telegram gate-level 进度，否则“正在跑”与“静默退出”无法区分。

9. 综合比较：各版本与工具的客观优劣

比较对象	优势	代价/风险	适合场景
Qwen 3.6 27B FP8	在多项同类任务上 wall time 较低；后期 Task 2 仅 38.1s	过程透明度/推理显式性较弱；模型版本较旧	快速日常诊断、低延迟任务
Qwen 3.8 27B FP8	过程说明更丰富；复杂 repo 分析覆盖更广；tool calling/长上下文可用	早期同任务明显更慢；更容易主动扩大调查范围	需要更强可解释性、复杂取证
Qwen 3.8 + medium	推理显式、复杂根因分析更稳健	reasoning token 直接增加 decode 成本；不适合所有快任务	架构审计、代码根因、独立复核
Hermes v0.16→0.20.x 系列	执行速度、交互可见性、skills/memory 生产收益	状态依赖强；早期配置/压缩/路径问题需注意	长期驻留 executor
DSH rc7	forensic reasoning、架构边界、独立审计强；某些结构性 coding task 非常快	广范围调查可能慢；当时需要 developer-role 兼容补丁	architect / auditor
Hybrid autonomous stack	把架构、实现、确定性验证、复核拆开；真实任务已闭环	controller 本身也要版本化、测试和修复；同模型不是认知独立	高置信度 autonomous software engineering

9.1 不应该得出的结论

- 不能仅凭本实验宣布“Qwen 3.8 总体优于/劣于 Qwen 3.6”；端到端 agent 时间受 reasoning、skills/memory、调查深度和缓存影响太大。
- 不能用三项 Hermes/DSH 任务做统计意义上的通用排行榜；它们足以说明 task-shape dependence，但样本量仍小。
- 不能把 DSH post-audit 当成“第二模型意见”；底层模型相同，因此它是独立流程，不是独立模型。
- 不能把测试全绿等同于架构正确；首个 cleanup 的 textual-reference false positive 说明规则与语义需要共同审计。

9.2 当前最合理的生产组合

1. 默认模型：Qwen 3.8 27B FP8；复杂架构/审计使用 reasoning=medium，快速重复运维任务可降低 reasoning。

2. Hermes：主 implementation/execution agent，允许使用长期 skills/memory，但只在 disposable worktree 中写。

3. DSH：主 architect 和 post-auditor，强制 read-only，reasoning=medium，并验证实际 request header。

4. Controller：负责所有不可委托给 LLM 的 hard facts——Git identity、patch hash、allowed paths、完整测试、文档检查、artifact byte identity、branch fast-forward、resume gates。

5. 若未来需要真正独立审计，应引入不同模型家族作为第二 auditor，而不仅是不同 harness。

10. 推荐的下一轮可重复 benchmark 规范

10.1 每次运行必须冻结的元数据

类别	必须记录
Hardware	GPU model/VRAM/power cap/driver/CUDA；CPU visible cores；RAM；disk free
Inference	SGLang version；model exact ID；context length；tp；mem fraction；speculative config；chunked prefill；tool parser
Harness	Hermes/DSH exact version/commit；profile；skills/memory policy；reasoning effort；stream setting
Workload	anonymized task ID；repo commit；worktree ID；test fixture hash；read-only/write policy
Timing	wall/user/sys/maxrss；TTFT；server decode/prefill；tool calls；retries
Correctness	requirements met；diff scope；focused/full tests；docs check；artifact identity
State	cache warm/cold；session fresh/resumed；skills/memory fresh/warm；concurrent sessions

10.2 建议的 A/B 矩阵

实验	固定项	唯一变量	核心指标
Model A/B	harness/profile/tasks/ cache/reasoning	Qwen 3.6 vs 3.8	质量、wall、tool calls、decode tokens
Reasoning A/B	model/harness/task	low/medium	正确性增益 / 额外 token / wall
Harness A/B	model/task/reasoning/repo	Hermes vs DSH	正确性、wall、diff scope、tool efficiency
State A/B	model/task/harness	fresh vs warm skills/memory	重复任务提速、错误率
Streaming A/B	固定 prompt/output	interval 1 vs 4	TTFT、wall、chunks、UI overhead
Autonomy A/B	同任务和模型	single-agent vs architect→executor→controller→auditor	错误逃逸率、人工介入次数、总 wall/model calls

附录 A：关键原始数字（隐私脱敏）

A.1 Sequential model/config runs

阶段	Task 2	Task 3	Task 5	Task 6
Qwen 3.6 baseline	44.898s	125.648s	163.0s	207.5s
Qwen 3.8 initial	66.220s	392.421s	380.1s	405.5s
Qwen 3.8 medium	249.287s	168.722s	386.68s	—
Qwen 3.8 medium repeat 1	218.633s	161.312s	—	—
Qwen 3.8 medium repeat 2	52.116s	169.066s	—	—
Qwen 3.6 later check	38.141s	137.843s	—	—

A.2 Streaming microbenchmark（5-run means）

参数	interval=1	interval=4	变化
wall time	42.0734s	42.1098s	+0.0865%
TTFT	52.4ms	53.0ms	近似不变
stream chunks	1360	351	-74.2%
output chars	19,402	19,402	完全相同
chunks/s	32.324	8.334	事件频率约降 74%

A.3 Hermes vs DSH direct runs

任务	Hermes wall	Hermes RSS	DSH wall	DSH RSS
Read-only premise recovery	88.76s	160,932 KB	281.70s	191,552 KB
Exact-write regression	56.77s	171,892 KB	145.35s	189,820 KB
Preflight-guard regression	121.92s	181,108 KB	51.35s	181,888 KB

附录 B：数据质量与限制

- 部分早期 benchmark 的 token 统计不可从 harness 内部自省，因此记录为 N/A/unknown；本报告没有用估算 token 做跨模型排名。
- Qwen 3.6 与 3.8 的 sequential runs 并非每轮都处于完全相同的 session/skills/cache 状态，因此 wall time 只做趋势展示。
- Hermes v0.16.0 与 v0.20.4 没有单独版本-only A/B；后者只在自治闭环中被验证。
- SGLang 某些阶段 /metrics 未启用，因此 speculative acceptance、cached_prompt_tokens 等指标来自 server log 或受控 API 重放，而不是统一的 Prometheus 序列。
- 所有 Repo-R 内容均已抽象；报告中的任务描述用于说明工具行为，不足以恢复项目目的或业务语义。

附录 C：一句话决策表

如果你要……	选择
快速完成受限代码修改	Hermes + disposable worktree + controller
证明一个架构边界是否真的安全	DSH read-only medium
最大化长期重复任务效率	Hermes + warm skills/memory
做严格模型 A/B	fresh state + frozen harness/profile + fixed repo/task
做高置信度自治	DSH architect → Hermes executor → deterministic controller → DSH auditor
减少 streaming UI/transport 事件	stream interval=4（本实验输出规模下）

报告结束。本文档是外部分享版：保留实验可复现性和工具比较结论，同时主动移除了所有与私有项目目的、领域内容和路线有关的信息。