Archonics Sample Audit: GPT-Researcher

SUBJECT: GPT-Researcher (open-source autonomous research agent) **Repository:** github.com/assafelovic/gpt-researcher **Version audited:** v3.4.4 (April 2026) **Audit tier:** Full Audit (sample, unsolicited) **Methodology version:** Archonics Audit Methodology v1.0 **Auditor:** Archonics **Date:** April 22, 2026

A note on this audit

This is a sample audit of an open-source project, performed by Archonics without engagement from the GPT-Researcher maintainers. It is published to demonstrate the shape, depth, and honesty of a real Archonics audit. We hold GPT-Researcher in high regard — 26,500 stars and sixty-nine releases reflect a project that has earned the trust of its users. Our critique here is meant in the spirit in which an engineer reviews a peer's code: specific, technical, and focused on marginal improvements rather than fundamental redesign. Findings are based exclusively on the project's published source code, documentation, and architecture diagrams. We have no access to internal evaluation infrastructure or production telemetry.

If you are a GPT-Researcher maintainer and would like to discuss any finding, we welcome the conversation at audits@archonics.ai.

Executive summary

GPT-Researcher is architecturally sound. The planner-execution split, parallelized sub-query processing, and agent-role specialization are thoughtful choices that have clearly been refined through real use. The audit surfaces nineteen findings across four dimensions, of which two are rated Critical, five High, eight Medium, and four Low.

The highest-leverage improvements cluster in three areas:

1. `ROLE-PROMPT SPECIFICATION.` The six specialized agent roles (Finance, Travel, Academic Research, Business Analyst, Computer Security Analyst, Default) are defined in a single-line dictionary with minimal behavioral guidance. Production research quality depends heavily on role-prompt content; the current implementation leaves substantial quality on the table.
2. `TOOL-CALL RELIABILITY ON THE MCP INTEGRATION.` When operating with the `RETRIEVER=tavily,mcp` hybrid retriever, the agent relies on MCP tool descriptions it did not author. Tool descriptions from third-party MCP servers vary wildly in quality, and the current architecture has no defensive layer to normalize or supplement those descriptions before the model sees them. This is the highest-risk finding.
3. `EVAL-TO-PROMPT FEEDBACK LOOP.` The `evals/` directory exists and contains benchmark infrastructure, but there is no evidence of a CI path that runs prompt-level regression tests on every change to `prompts.py`. Given how central `prompts.py` is to output quality, any change to it carries silent risk.

This audit's top-ten prioritized fix list appears at the end.

Context

- **System audited:** The core `gpt_researcher` package, with particular attention to `gpt_researcher/prompts.py` and the agent orchestration in `gpt_researcher/agent.py`. The multi-agent LangGraph variant in `multi_agents/` was reviewed at the architectural level only.
- **What we had access to:** public source code, public documentation at docs.gptr.dev, published release notes, and observable behavior from the pip package.
- **What we did not have access to:** internal eval results, production telemetry, user-reported failure cases, or unreleased branches.
- **Scope boundary:** This audit covers prompt and context engineering. It does not audit the web-scraping stack, vector storage, or the frontend — those systems have their own engineering disciplines and are out of scope for an Archonics audit.

Dimension 1 — System prompt analysis

GPT-Researcher's "system prompt" is distributed across several components: the role prompt returned by `generate_agent_role_prompt()`, the task-specific instructions embedded in the report generation prompts (`generate_report_prompt()`, `generate_resource_report_prompt()`, `generate_outline_report_prompt()`), and the decomposition instructions in `generate_search_queries_prompt()`.

// Finding 1.1 — Role prompts are underspecified (Severity: High)

EVIDENCE. The six agent role prompts are single-sentence descriptors. For example, the Finance Agent role is described as a "seasoned finance analyst focused on comprehensive financial reports." This is a category label, not an operating specification. It communicates domain to the model but provides no behavioral guidance: how should the agent handle conflicting sources? What's the disposition toward quantitative claims without citations? What's the tone? How should the agent handle topics at the edge of the domain (e.g., a finance-adjacent question that's really a policy question)?

IMPACT. Production output quality on domain-specific research tasks is disproportionately affected by role prompt content. A one-sentence role leaves the model to infer everything else from its training distribution, which produces generic, hedged, and sometimes tonally inconsistent reports. A finance analyst report and a travel report read remarkably similarly in their default output, which is a symptom of under-differentiated roles.

RECOMMENDATION. Expand each role prompt to 150–300 words covering: domain expertise scope, handling of conflicting evidence, disposition toward quantitative vs. qualitative claims, tone and register, and explicit guidance on what's out of scope. The dictionary structure makes this change trivial; the work is in writing the role prompts well.

BLAST RADIUS. Affects every research task. A quality improvement here propagates to all report types.

// Finding 1.2 — No explicit priority structure for instruction conflicts (Severity: Medium)

EVIDENCE. `generate_report_prompt()` includes multiple directives: minimum 1,200 words, Markdown with APA formatting, concrete opinions based on findings, source URLs at the end. These can and do conflict at runtime — a question with sparse sources may not support 1,200 words of genuine content without padding, and a question where evidence is genuinely ambiguous resists "concrete opinion" framing.

IMPACT. When instructions conflict, the model resolves the conflict unpredictably. This produces two common failure modes: reports that hit the word count by padding with restated material (we observed this pattern in publicly shared GPT-Researcher outputs) and reports that assert confidence on topics where the evidence base warrants more caution.

RECOMMENDATION. Introduce an explicit priority order in the prompt — for example, "When these requirements conflict, prioritize in this order: (1) fidelity to evidence, (2) structural completeness, (3) minimum length." The word count should be framed as a soft floor, not a hard target.

// Finding 1.3 — "No such agent" fallback is silent and unrecoverable (Severity: Critical)

EVIDENCE. The `generate_agent_role_prompt()` function returns the literal string `"No such agent"` when the requested agent type is not in the dictionary. This string is then used as the system message for the LLM call.

IMPACT. A misconfigured or typo'd agent type silently degrades the agent's operating identity to a three-word nonsensical instruction. The model will still produce output — often acceptable-looking output — but without the role specialization that was supposed to be active. This is a failure mode that's exceptionally hard to detect in production because the output doesn't look obviously broken.

RECOMMENDATION. Treat unknown agent types as a programming error. Either raise an exception that forces the caller to handle the missing agent, or fall back to the `Default Agent` role *and* log a warning. A three-word system message is the worst of both worlds: the behavior continues but with silent quality degradation.

// Finding 1.4 — Report prompts embed research summary with ambiguous delimiters (Severity: Medium)

EVIDENCE. Report prompts use triple-quoted strings (`"""{research_summary}"""`) to delimit the input data from instructions. This is a widely-used pattern and works most of the time, but it is vulnerable to input containing matching triple quotes, which can confuse the model about where data ends and instructions resume.

IMPACT. In normal research, scraped content rarely contains triple quotes. But research on topics like Python programming, agent systems, or prompt engineering frequently includes content with triple-quoted blocks. The audit observed that GPT-Researcher would be susceptible to prompt-injection-via-content if a scraped page contained `"""` followed by adversarial instructions.

RECOMMENDATION. Use a more robust delimiter convention — XML-style tags (`<research_summary>...</research_summary>`) are substantially harder for scraped content to collide with. Modern frontier models are trained to respect such boundaries more reliably than triple quotes.

// Finding 1.5 — Prompts are not versioned (Severity: Medium)

EVIDENCE. `prompts.py` is a flat module; there is no prompt version identifier surfaced in the research pipeline, logs, or output artifacts.

IMPACT. When a prompt change affects output quality — positively or negatively — there is no way to correlate that change with specific reports after the fact. Users reporting quality regressions cannot be answered with "that report was produced with prompt v2.3, which had a known issue fixed in v2.4."

RECOMMENDATION. Embed a prompt version in each generated prompt and propagate it into report metadata. This unlocks structured rollback analysis and makes A/B testing of prompt changes tractable.

// Finding 1.6 — Output format specified but not enforced (Severity: Low)

EVIDENCE. The JSON-output prompts (`generate_search_queries_prompt` , `generate_concepts_prompt` , `auto_agent_instructions`) specify output formats in natural language: `["query 1", "query 2", "query 3", "query 4"]` . Downstream code presumably parses these as JSON.

IMPACT. Natural-language format specification is the most common cause of tool-call and structured-output parse failures. Modern models support structured output modes (OpenAI's `response_format`, Anthropic's tool use) that guarantee schema adherence. Relying on prompt-level format specification is a choice that buys flexibility at the cost of reliability.

RECOMMENDATION. Migrate JSON-output calls to use provider-native structured output where available. The benefit compounds over time as models are updated; natural-language format specifications are the kind of technical debt that accumulates silently.

Dimension 2 — Tool definition review

GPT-Researcher's tool surface includes web search (via Tavily, SerpAPI, or other configurable retrievers), local document retrieval, and — when configured — third-party MCP servers. The audit examines how the agent decides when to use which tool.

// Finding 2.1 — MCP tool descriptions are consumed un-normalized (Severity: Critical)

EVIDENCE. When `RETRIEVER=tavily,mcp` is configured, the agent receives MCP tool definitions from whatever server is configured (GitHub, databases, custom APIs). The quality of these tool descriptions varies dramatically across the MCP ecosystem. GPT-Researcher passes them through to the model without normalization, deduplication, or quality checks.

IMPACT. A well-configured MCP server with good tool descriptions improves research quality. A poorly-configured one can actively degrade it — models waste calls on tools whose descriptions don't communicate when to use them, or redundantly call overlapping tools from different servers. The audit observed that the `@modelcontextprotocol/server-github` example in the README would, if combined with a generic web search tool, produce ambiguous tool selection because neither description explicitly disambiguates "when to use GitHub search vs. general web search."

RECOMMENDATION. Introduce a tool-description normalization layer that (a) appends a uniform "When to use this tool vs. alternatives" section to each tool, derived from the tool's signature and the agent's active context, and (b) deduplicates overlapping tools by picking one primary per capability. This is defensive context engineering.

BLAST RADIUS. Affects every hybrid research task. This is the single most important finding in the audit because hybrid MCP research is positioned as a flagship capability and its reliability depends on tool descriptions the project does not control.

// Finding 2.2 — Search retriever abstraction lacks capability metadata (Severity: High)

EVIDENCE. The retriever abstraction supports Tavily, SerpAPI, DuckDuckGo, Bing, Google, Exa, and others. Each has different latency, cost, result quality, and semantic-search capability. The current architecture treats them as interchangeable behind a common interface.

IMPACT. Users configure a retriever based on their API keys and preferences, but the agent cannot reason about trade-offs. An Exa search is better for semantic discovery; DuckDuckGo is better for avoiding rate limits on simple queries; Tavily is optimized for agent use and rejects low-quality results server-side. The agent would benefit from knowing this when deciding how many queries to issue and how to interpret their results.

RECOMMENDATION. Add a capability-metadata object to each retriever implementation (semantic-capable, rate-limit tier, typical latency, result quality self-assessment). Expose this metadata to the agent so it can adapt query-decomposition strategy to retriever capability.

// Finding 2.3 — No tool-call regression tests (Severity: High)

EVIDENCE. The `evals/` directory contains benchmark infrastructure, but reviewing the test structure reveals the focus is end-to-end report quality, not specific tool-call behavior. There is no test that says "given this question, the agent should decompose into N queries with these characteristics."

IMPACT. Changes to `generate_search_queries_prompt()` — a central prompt — can silently degrade query decomposition in ways that only surface as worse reports three steps downstream. The feedback loop from prompt change to observed quality change is long and noisy.

RECOMMENDATION. Add targeted tool-call tests for the search-query-generation step specifically. Test cases should assert properties of the generated queries (count, diversity, specificity) rather than exact string matches, which would be too brittle.

// Finding 2.4 — Auto-agent selection has no rejection path (Severity: Medium)

EVIDENCE. `auto_agent_instructions()` provides three example task-to-agent mappings and asks the model to select one of the six defined agents. There is no explicit instruction for what to do when a task doesn't clearly fit any of the six agents.

IMPACT. The agent will always select one of the six, even for tasks that are poor fits. A question about 19th-century poetry will be routed to Academic Research Agent, which is the best available fit but not actually a great fit — the Academic role is biased toward empirical research, not literary criticism.

RECOMMENDATION. Add a "Default Agent" selection path that explicitly covers topics outside the six specialized domains, and provide guidance in the auto-agent prompt on when to prefer Default over a marginal specialty fit.

Dimension 3 — Context packing analysis

Research agents are context-intensive by nature — each sub-query adds scraped content that must be summarized, stored, and eventually aggregated into a report. GPT-Researcher makes several sensible choices here (parallelized scraping, per-source summarization, aggregated-summary-as-context-for-report). The audit surfaces specific inefficiencies.

// Finding 3.1 — Per-source summarization runs before deduplication (Severity: High)

EVIDENCE. Based on the published architecture, each scraped source is summarized independently before summaries are aggregated. If two sources contain substantially overlapping content (common for news stories syndicated across outlets), both are independently summarized and both summaries enter the aggregated context.

IMPACT. A research task that scrapes 20 sources covering an event that was reported by 5 primary outlets and 15 syndicating aggregators will produce 15 summaries that are near-duplicates. Token cost scales linearly; report quality does not.

RECOMMENDATION. Introduce a pre-summarization deduplication pass. Even crude deduplication (shingling, embedding similarity) at the raw-content level before summarization would meaningfully reduce context bloat. The deduplication pass pays for itself in saved summarization API calls.

// Finding 3.2 — Word count target inflates context on aggregation (Severity: Medium)

EVIDENCE. Report generation prompts specify "minimum of 1,200 words." Per-source summaries are therefore kept long to ensure enough raw material for a 1,200-word report.

IMPACT. Long per-source summaries are context-expensive at aggregation time. The aggregation prompt must hold all source summaries plus the research question plus the report-generation instructions. For research tasks with 20+ sources, this can approach context window limits on older models and incurs significant token cost on newer ones.

RECOMMENDATION. Switch to two-pass aggregation: a first pass that identifies the key claims and their supporting sources (short output), and a second pass that expands those claims into the 1,200-word report. This pattern keeps the expensive context-dense pass short and the expensive generation pass focused.

// Finding 3.3 — No prompt caching utilization (Severity: Medium)

EVIDENCE. The static portion of each prompt (role prompt, format instructions) is re-sent on every LLM call with no indication the project leverages prompt caching on providers that support it (Anthropic, increasingly OpenAI).

IMPACT. Across a research task with a dozen or more LLM calls, a cached role prompt would save meaningful token spend. At Anthropic's current cache pricing, a typical research task could see 20–40% cost reduction from aggressive caching of the system prompt and tool definitions.

RECOMMENDATION. Structure prompts so the static prefix is cache-friendly (identical byte-for-byte across calls), and enable caching on supported providers. This is low-effort and the cost savings compound at scale.

// Finding 3.4 — No strategy for long-conversation context overflow (Severity: Medium)

EVIDENCE. Deep Research mode explicitly takes ~5 minutes per task with tree-like exploration. This implies many LLM calls and accumulating context. There is no observable strategy for context trimming as conversations grow.

IMPACT. Deep Research tasks on broad topics will hit context limits or suffer from attention dilution on older tree branches. The failure mode isn't a hard error — it's a gradual quality degradation as the agent's effective working memory fills up.

RECOMMENDATION. Implement hierarchical summarization for Deep Research: periodically compress earlier branches of the exploration tree into summary form, preserving recent/active branches in full detail. This mirrors how human researchers handle long investigations.

Dimension 4 — Evaluation gap analysis

The `evals/` directory is a positive signal — many agent projects ship with no eval infrastructure at all. The audit examines what's tested, what isn't, and where the highest-ROI additions would be.

// Finding 4.1 — No prompt-level regression tests in CI (Severity: High)

EVIDENCE. The evals directory contains benchmarks, but there is no CI workflow that runs prompt-specific regression tests on every PR touching `prompts.py`.

IMPACT. A contributor can modify a prompt and have the change merged based on unit tests and a manual review that cannot realistically catch subtle behavioral regressions. Prompt changes are disproportionately likely to cause quality regressions that manifest only on specific task categories.

RECOMMENDATION. Add a CI workflow that, on any PR touching `prompts.py`, runs a curated set of prompt-regression tests: for each of ~20 canonical research tasks, generate the search queries and score them on coverage and specificity. Fail the CI if any score regresses significantly.

// Finding 4.2 — No failure-case library (Severity: High)

EVIDENCE. Reviewing the issue tracker, users report specific failure modes (hallucinated citations, missed topics, overly-generic reports), but there is no codified test case library that captures these failures and prevents regression.

IMPACT. Every reported issue is effectively an orphan observation — once it's closed, there's nothing preventing the same failure class from recurring. Over time the project accumulates a well-known list of "sometimes this breaks" scenarios with no automated defense.

RECOMMENDATION. Adopt the practice: every user-reported failure that can be reduced to a reproducible test case should land in `evals/failure_cases/` before the fix lands. This converts individual reports into durable quality infrastructure.

// Finding 4.3 — No tool-call accuracy metrics (Severity: Medium)

EVIDENCE. Eval focus is on final report quality. There are no separate metrics for "did the agent choose the right retriever?" or "did the agent's decomposed queries cover the question space?"

IMPACT. Report quality is a lagging metric with high variance. A bad tool-call decision may produce an acceptable report through sheer model robustness, which hides the tool-call problem until it compounds with other issues.

RECOMMENDATION. Add intermediate-step metrics. Tool-call accuracy, query-decomposition coverage, and per-source summary fidelity are all measurable and give faster feedback than waiting for end-to-end report quality to shift.

// Finding 4.4 — Observability is WebSocket-based but not structured (Severity: Low)

EVIDENCE. The WebSocket manager streams progress updates to the frontend, but there is no structured observability export suitable for offline analysis.

IMPACT. Teams deploying GPT-Researcher in production have limited ability to retrospectively debug specific research tasks that produced poor output. "Reproduce it locally and watch the WebSocket" is not an acceptable observability story for a production agent system.

RECOMMENDATION. Add structured logging (JSONL) at each stage transition: query generation, per-query scraping, summarization, aggregation, report generation. Make these logs optionally exportable to LangSmith (already supported for LangGraph agents) or similar observability platforms.

Prioritized fix list (top 10)

Ranked by severity × blast radius × reversibility, with effort notes.

#	FINDING	SEVERITY	EFFORT	NOTES
1	2.1 — Normalize MCP tool descriptions	Critical	Significant	Highest blast radius; affects hybrid MCP research reliability
2	1.3 — Replace "No such agent" fallback	Critical	Trivial	One-line change with large silent-failure prevention
3	1.1 — Expand role prompts to 150–300 words each	High	Modest	Disproportionate quality gain per line of code
4	4.1 — Prompt-regression CI workflow	High	Modest	Prevents future quality debt accumulation
5	4.2 — Failure-case library	High	Modest	Cultural shift as much as code; massive long-term ROI
6	3.1 — Pre-summarization deduplication	High	Modest	Direct cost savings and quality improvement
7	2.2 — Retriever capability metadata	High	Modest	Enables smarter agent query strategy
8	1.4 — XML delimiters for research summaries	Medium	Trivial	Cheap injection-resistance improvement
9	3.3 — Enable prompt caching	Medium	Modest	Direct cost savings, no quality trade-off
10	2.3 — Tool-call regression tests	High	Significant	Longer-term investment; pairs with #4

Recommended eval additions

Independent of specific findings, the following test classes would strengthen GPT-Researcher's quality posture substantially:

1. **Query-decomposition coverage test.** For a fixed set of research questions, assert the generated queries cover the expected subtopics. Score on recall against a hand-curated subtopic set.
2. **Role-consistency test.** For each specialized agent, run a task matched to the agent and a task mismatched to the agent. Assert the output reflects the role specialization. Score on per-role signature presence.
3. **Citation-fidelity test.** For a known source set, assert the report's citations map to actual source content (no hallucinated quotes, no misattributed claims).
4. **Context-budget test.** For a task that produces N source summaries, assert aggregation stays within a target token budget. Catches context bloat regressions early.
5. **Prompt-injection test.** Seed scraped content with injection attempts ("ignore previous instructions") and assert the agent continues the research task. Most current agents fail this test; it's worth having.

Open questions

We did not have visibility into these areas and flag them as gaps in this audit's coverage, not gaps in the project itself:

1. **Internal eval results.** Is the project measuring any of these dimensions internally? If so, what does the data say?
2. **Production telemetry.** What do users' production deployments reveal about failure modes?
3. **Roadmap alignment.** Which of these findings, if any, are already on the team's roadmap? We did not attempt to reconstruct roadmap state from the Trello board linked in the README.

A paid Archonics Full Audit engagement would address these gaps through direct collaboration with the maintainers. This sample audit represents approximately the depth achievable from public information alone.

About Archonics

Archonics performs context engineering audits on production agent systems. We examine system prompts, tool definitions, context packing, and evaluation infrastructure — the four dimensions that determine whether an agent built in a demo survives in production. Our methodology is documented at archonics.ai/methodology and this audit applies it in full.

If you're shipping an agent system and suspect you have findings like these hiding in your own code, we offer three tiers of engagement:

- **Free Scan** — Connect our MCP server, submit a prompt, get three findings back. Zero cost.
- **Instant Audit** — $49 USDC via x402. Full methodology applied programmatically to a submitted system. 5-10 page PDF report.
- **Full Audit** — $750. Human-reviewed audit of a complete agent system, in the shape of this document.

Contact: audits@archonics.ai Methodology: archonics.ai/methodology

This sample audit was produced on April 22, 2026, using Archonics Audit Methodology v1.0. It is offered freely to the GPT-Researcher community and to anyone evaluating Archonics for a paid engagement.