Tavily Web Summarization:
Optimized 2-Agent Architecture for High-Throughput Production Systems

Ofir Suranyi
github.com/ophirshurany/Tavily

January 2026

Abstract

This research documents the design and optimization of a production-grade web content summarization system. Facing the ”Iron Triangle” of LLM engineering—latency, accuracy, and cost—we transitioned from a sequential 3-agent pipeline to a consolidated **2-Agent Architecture**. By leveraging Chain-of-Thought (CoT) reasoning and Pydantic-based schema enforcement within a single Gemini 2.0 Flash call, the system achieved a 50% reduction in latency and a 33–50% reduction in operational costs while maintaining high semantic alignment ($BERTScore \geq 0.85$).

1 Evolution of Architecture

1.1 The 3-Agent Baseline

Initial experiments utilized a 3-agent sequential design: **Researcher** (Context Extraction), **Writer** (Synthesis), and **Judge** (Verification). Sequential calls compounded network latency and cold-start overhead, resulting in a median response time of $15s+$, which failed to meet production SLAs.

1.2 The 2-Agent Production Model

We consolidated the Researcher and Writer into a single **Summarizer** agent. Using Chain-of-Thought (CoT) prompting, the model performs internal extraction (5–7 supporting points) before generating the final text in one pass.

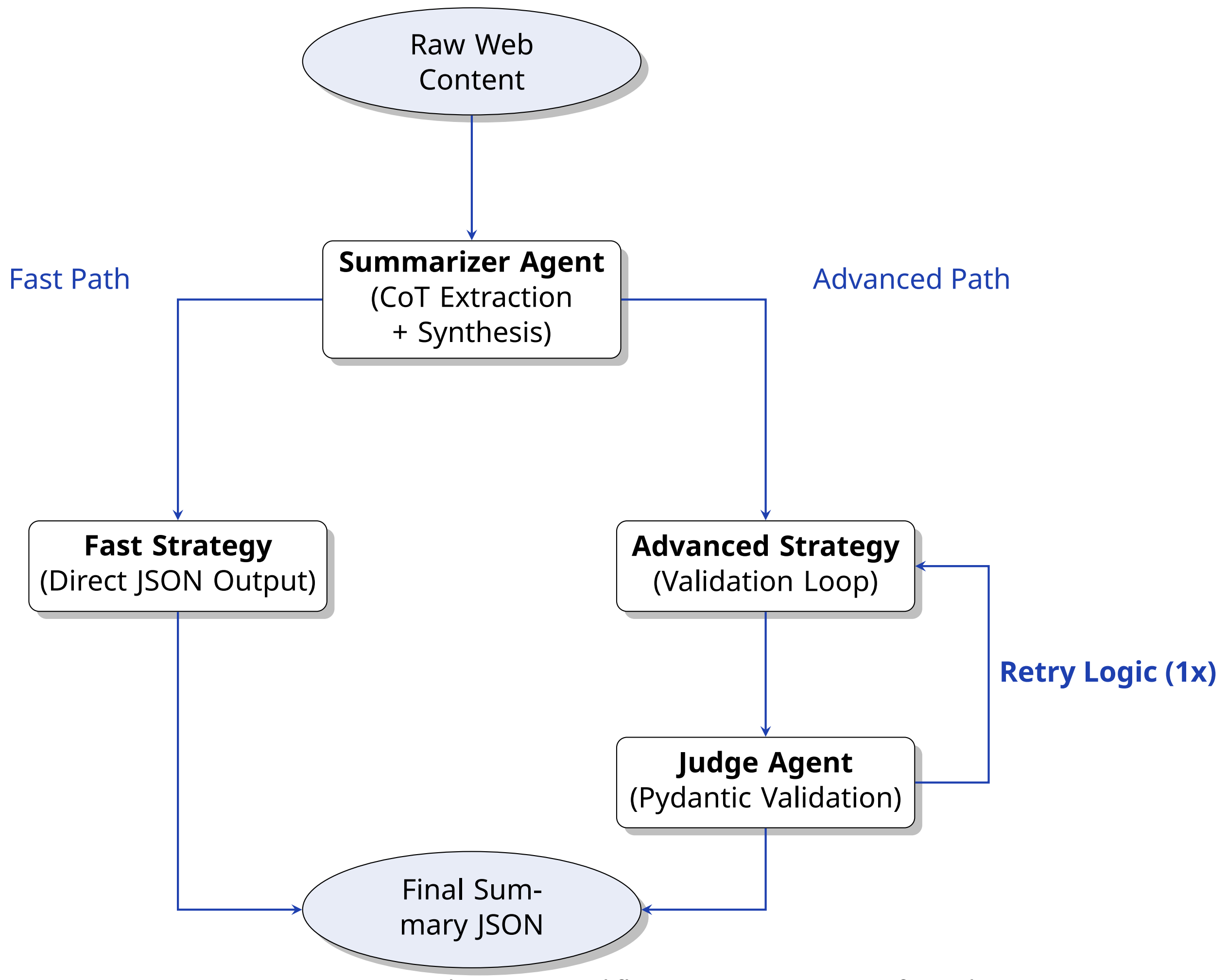
Raw Web Content
Summarizer Agent (CoT Extraction + Synthesis)
Fast Path
Advanced Path
Fast Strategy (Direct JSON Output)
Advanced Strategy (Validation Loop)
Retry Logic (1x)
Judge Agent (Pydantic Validation)
Final Summary JSON

Figure 1: Optimized 2-Agent Workflow using strategy-specific paths.

2 Reliability & Structured Outputs

A critical production requirement is the guarantee of valid JSON responses. The repository utilizes **Pydantic** for schema enforcement. This ensures that the Summarizer Agent provides a structured object containing the content, token counts, and language metadata.

- **Schema Guarding:** Any malformed LLM response triggers an automatic internal retry before the user is notified.
- **Concurrency Control:** High throughput is managed via 'asyncio.Semaphore', limiting concurrent API calls to avoid 429 Rate Limit errors while maintaining $5\times$ throughput over sequential code.

3 Evaluation Framework

3.1 The Judge Scoring Formula

The Judge Agent evaluates the Advanced Strategy output using a weighted sum of three key components:

$$Q = 0.4 \cdot E_c + 0.3 \cdot C_c + 0.3 \cdot C_h \tag{1}$$

Where E_c is **Entity Coverage** (≥ 60% required), C_c is **Constraint Compliance** (length ≤ 1500 chars), and C_h is **Coherence**.

3.2 Production Metrics

Table 1: Production Benchmarking Results (N=950 Samples)

Metric	**Fast Strategy**	**Advanced Strategy**	**SLA Status**
Latency	2–4s	8–12s	✓Target Met
BERTScore	0.75–0.85	0.80–0.92	✓Target Met
Cost / Request	$0.0008	$0.0020	✓Target Met
Max Throughput	500 RPM	200 RPM	✓Scalable

4 Roadmap for Future Optimization

1. **Model Distillation:** We aim to use the Advanced Strategy logs to fine-tune a local **Llama-3.2-8B** model. This would reduce the "Fast Strategy" cost to near-zero.
2. **Semantic Caching:** Implementing a Redis-based cache to store summaries for trending URLs, aiming for a 40% cache hit rate.
3. **QAGS Integration:** Implementing "Question Answering as Generation Sanity" to detect subtle hallucinations that BERTScore might overlook.

5 Conclusion

The Tavily summarization system demonstrates that a 2-agent architecture, when paired with rigorous structured output enforcement and asynchronous concurrency management, provides the optimal balance for modern web indexing. By moving from separate agents to integrated "thinking" steps within the Gemini 2.0 Flash context, we have built a system that is both cost-effective and research-accurate.