Strategic Architecture and Implementation Plan for ATLAS: A Hybrid Voice-First AI Assistant

1. Executive Summary and System Philosophy

The conception of ATLAS—a voice-first AI life assistant utilizing a hybrid Large Language Model (LLM) architecture—represents a sophisticated exercise in constrained optimization. The objective is not merely to construct a functional command-and-control interface, but to synthesize a digital entity that embodies the "Lethal Gentleman" persona (R21): an agent of understated authority, absolute competence, and seamless execution. This persona, reminiscent of archetypes like Alfred Pennyworth, demands a system interaction model where latency is masked by deliberation and failure is framed as discretion.

The engineering challenge is defined by a distinct "Trilemma of Constraints":

1. **Latency:** The "Voice Loop" (Speech-to-Text $\rightarrow$ Intelligence $\rightarrow$ Text-to-Speech) must complete within 3 seconds to maintain the illusion of conversation.
2. **Hardware:** The physical substrate is restricted to a localized WSL2 environment with only 4GB of VRAM and 16GB of system RAM, necessitating aggressive model quantization and resource management.
3. **Economics:** The operational expenditure (OpEx) for cloud intelligence is capped at a modest $5-10 per month, requiring a routing architecture that treats high-intelligence tokens as a scarce resource.

To resolve this trilemma, this report proposes and details a **Tiered Semantic Cascade Architecture**. This architecture rejects the monolithic model approach in favor of a specialized hierarchy: a "Reflex Tier" for instantaneous device control, a "Local Intelligence Tier" (Qwen2.5-3B) for routine interaction, and a "Cloud Intelligence Tier" (Anthropic Haiku 4.5/Agent SDK) for complex reasoning. The intelligence of the system lies not just in the models themselves, but in the *router*—the decision engine that assigns each user query to the most cost-effective tier capable of satisfying the intent.

The following analysis draws upon extensive 2025-2026 research into LLM routing strategies [1], quantization benchmarks [3], and voice interface taxonomies [5] to provide a blueprint for ATLAS. It moves beyond high-level abstraction into specific implementation patterns, code logic, and economic modeling required to bring the Lethal Gentleman to life within the specified constraints.

---

2. Query Classification Approaches

The "Brain" of ATLAS is not the LLM that generates the answer, but the Classifier that decides *who* answers. Given the requirement for <3s end-to-end latency, the routing decision must be made in milliseconds. If the router is too slow, it consumes the time budget required for the actual intelligence; if it is inaccurate, it either degrades the user experience (by sending complex queries to a dumb local model) or bankrupts the project (by sending simple queries to the cloud).

2.1 Comparative Analysis of Classification Strategies

We have analyzed five distinct approaches to query classification, evaluating them against the ATLAS constraints of minimal latency and low computational overhead.

2.1.1 Rule-Based Heuristics (The Reflex Layer)

The most primitive yet essential layer of any voice assistant is heuristic routing. This involves mapping specific keyword patterns or regular expressions (Regex) directly to executable functions.

- **Mechanism:** The system scans the Automatic Speech Recognition (ASR) transcript for rigid patterns such as "Turn on" or "Set volume to [X]%".
- **Latency:** Effectively zero (<1ms). This approach bypasses all neural inference.
- **Role in ATLAS:** This serves as the "spinal reflex" of the Lethal Gentleman. When a user issues a direct command, immediate obedience is required. There is no need for "reasoning" about whether to turn on a light; to ponder such a request would break the persona of the efficient servant.
- **Limitations:** It has zero semantic flexibility. A user saying "It's a bit dark in here" would fail a regex check for "Turn on lights," necessitating a smarter fallback.

2.1.2 Embedding-Based Classification (The Primary Router)

This represents the current state-of-the-art for low-latency routing.[1] The core concept is **Semantic Routing**: converting the user's query into a high-dimensional vector and comparing its angle (cosine similarity) against a database of pre-defined "Intent Vectors."

- **Mechanism:** An embedding model, such as all-MiniLM-L6-v2 or the more modern bge-small-en, processes the text. The resulting vector is compared against a local index of "Route Prototypes." For example, the "Cloud Reasoning" route might be defined by the centroid of vectors for "Write a python script," "Analyze this file," and "Explain quantum physics."
- **Benchmarks:**
  - **Latency:** all-MiniLM-L6-v2 is exceptionally fast. On a standard CPU (avoiding VRAM usage), it can encode a query in ~14-20ms.[6] While some Python implementations show higher latency (~122ms) due to unoptimized frameworks, utilizing optimized libraries like ONNX Runtime or candle (Rust bindings) brings this down to negligible

levels.[6]
    - **Accuracy:** Research indicates that embedding-based routers can achieve 90-95% of the accuracy of LLM-based routers for domain-specific tasks.[1] The semantic-router library demonstrates that this approach can reduce token consumption by 48.5% by effectively filtering queries.[9]
- **Implementation Strategy:** For ATLAS, we define a "High Water Mark" threshold. If a query's similarity to the "Complex Task" cluster is >0.82, it routes to Cloud. If >0.6 to "Conversation", it routes to Local.

2.1.3 Local LLM as Classifier (Generative Routing)

This strategy involves prompting the local Qwen2.5-3B model to classify the query before generating a response. The prompt would look like: *"Classify the following user input as 'SIMPLE', 'COMPLEX', or 'AGENT': [Input]"*.

- **Analysis:** While Qwen2.5-3B is a capable instruction follower [10], this approach is **architecturally flawed** for ATLAS.
    - **Latency Penalty:** Even with a fast Time-To-First-Token (TTFT) of ~200ms, the system must generate the classification, parse it, and *then* potentially initiate a call to the Cloud. This serializes the latency: $T_{total} = T_{ASR} + T_{Local} + T_{Network} + T_{Cloud}$. This significantly risks the 3s budget.
    - **Context:** [1] notes that while routers using LLMs (like "LLM Router") have high accuracy, they incur computational costs that defeat the purpose of optimization for a single-user low-power system.

2.1.4 Cascade and Speculative Approaches

The Cascade approach organizes classifiers in a sequence of increasing cost and accuracy.

1. **Stage 1 (Regex):** Is this a command? (Cost: 0, Latency: 0).
2. **Stage 2 (Vector):** Is this semantically similar to known complex tasks? (Cost: Low, Latency: 20ms).
3. **Stage 3 (Confidence/Perplexity):** This is the "Speculative" element. If Stage 2 is ambiguous (e.g., similarity score 0.5), the query is sent to the Local Qwen model. However, we monitor the **perplexity** (uncertainty) of the model's generation. If Qwen's perplexity for the first 5 tokens is high (indicating it is "confused"), the generation is aborted immediately, and the query is seamlessly handed off to the Cloud.[11]

- **Benefits:** This creates a safety net. If the Semantic Router fails to catch a subtle nuance, the Local LLM's own confusion acts as the final trigger for Cloud intervention. This aligns perfectly with the "Lethal Gentleman" persona—if the assistant hesitates (high perplexity), he defers to a higher authority (the Cloud) rather than blundering forward.

2.1.5 Haiku Micro-Classifier

Using the Anthropic Haiku API solely to classify the query.

- **Assessment:** This is economically viable ($1/M tokens is cheap) but latently disastrous. It introduces a mandatory ~500ms network round-trip for *every single interaction*, even asking for the time.[13] This violates the latency constraint for simple queries and should be rejected.

2.2 Recommended Architecture: The Hybrid Semantic Cascade

Based on the evidence, ATLAS should implement a **Hybrid Semantic Cascade**.

Stage	Mechanism	Target Latency	Logic
1. Reflex	Regex / Keyword	< 1 ms	Matches volume, stop, lights, timer. Action: Device Control.
2. Router	all-MiniLM-L6-v2 (CPU)	~20 ms	Vector sim vs. ``. Action: Route to Tier.
3. Safety	Local Qwen Perplexity	~250 ms	If Local generation perplexity > Threshold, Abort & Escalate to Cloud.

Code Pattern for Semantic Router (Python):

Python

```
import numpy as np
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

class AtlasRouter:
    def __init__(self):
        # Load on CPU to save VRAM for Qwen
        self.model = SentenceTransformer('all-MiniLM-L6-v2', device='cpu')
```

```
        # Define Canonical Prototypes for each Tier
        self.routes = {
            "TIER_3_AGENT": [
                "plan a trip", "analyze this file", "book a flight",
                "research the history of", "execute code"
            ],
            "TIER_2_CLOUD": [
                "draft an email", "summarize this article", "explain the nuance",
                "creative writing", "medical advice"
            ],
            "TIER_1_LOCAL": [
                "what time is it", "turn on the lights", "hello",
                "play music", "weather forecast"
            ]
        }

        # Pre-compute embeddings for prototypes
        self.route_vectors = {
            k: self.model.encode(v) for k, v in self.routes.items()
        }

    def route(self, query: str) -> str:
        # 1. Reflex Check (Regex would go here)

        # 2. Semantic Check
        query_vec = self.model.encode([query])

        best_score = -1
        selected_tier = "TIER_1_LOCAL" # Default to safe/cheap

        for tier, prototypes in self.route_vectors.items():
            # Calculate max similarity with any prototype in this tier
            similarities = cosine_similarity(query_vec, prototypes)
            max_sim = np.max(similarities)

            # Apply Tier-specific thresholds
            # We need higher confidence to burn money (Tier 2/3)
            threshold = 0.85 if tier!= "TIER_1_LOCAL" else 0.65

            if max_sim > threshold and max_sim > best_score:
                best_score = max_sim
                selected_tier = tier
```

```
    return selected_tier
```

This pattern ensures that the system defaults to the Local Tier (Free) unless there is strong semantic evidence that the query requires higher intelligence, directly addressing the budget constraint.

---

3. Query Taxonomy for Personal AI

To train and configure the router effectively, we must map the "Intent Universe" of a personal assistant. A binary "Simple/Complex" distinction is insufficient for the nuance of the Lethal Gentleman persona.

3.1 Existing Taxonomies

Academic datasets provide a baseline for intent distribution.

- **CLINC150:** This dataset identifies 150 specific intents across 10 domains (e.g., Banking, Travel, Utility).[14] Crucially, it introduces the concept of **Out-of-Scope (OOS)** queries—queries that the system should *not* attempt to answer. For ATLAS, OOS might technically be "In-Scope" for the Cloud Tier, representing the long tail of general knowledge.
- **SNIPS:** Focuses heavily on device control and media (e.g., PlayMusic, GetWeather, BookRestaurant).[16] These intents map almost exclusively to the Local Tier.

3.2 ATLAS-Specific Taxonomy

We propose a taxonomy organized by **Cognitive Load** and **Risk**, which dictates the routing tier.

Category A: The "Valet" (Tier 1 - Local Qwen)

- **Cognitive Load:** Low (Retrieval, Deterministic).
- **Risk:** Low (Misunderstanding is annoying, not critical).
- **Intents:**
  - **Phatic/Social:** "Good morning, Alfred." "Thank you." (Maintaining the persona bond).
  - **Device Control:** "Dim the lights." "Volume 50%."
  - **Basic Info:** "What time is it in Tokyo?" "Weather report."
  - **Contextual Memory (Recent):** "What did I just say?"
- **Persona Note:** Responses here must be crisp. "Done, sir." "Right away."

Category B: The "Consigliere" (Tier 2 - Cloud Haiku 4.5)

- **Cognitive Load:** Medium/High (Generative, Nuanced).
- **Risk:** Moderate (Requires accuracy and tone).

- **Intents:**
  - **Drafting:** "Write a polite decline to this invitation."
  - **Summarization:** "Summarize this email thread."
  - **Advisory:** "What are the pros and cons of this decision?"
  - **Complex Knowledge:** "Explain the mechanism of a rotary engine."
- **Persona Note:** Responses here are more verbose and thoughtful. "If I may suggest, sir..."

Category C: The "Operator" (Tier 3 - Agent SDK)

- **Cognitive Load:** Extreme (Multi-step, Tool Use).
- **Risk:** High (Financial/Scheduling implications).
- **Intents:**
  - **Planning:** "Plan a 3-day trip to Chicago under $1000."
  - **Execution:** "Book a table for two at a steakhouse tonight."
  - **Data Analysis:** "Analyze my spending spreadsheet."
- **Persona Note:** Requires "Buying Time." "I shall arrange the details immediately. Please allow me a moment."

3.3 Distribution Analysis

Empirical data on voice assistant usage suggests a **Pareto Distribution (80/20 Rule)** [5]:

- **~70-75%** of queries are Category A (Simple/Valet).
- **~20%** are Category B (Consigliere).
- **~5%** are Category C (Operator).

This distribution is the economic saviour of the project. If 75% of queries are handled by the free Local Qwen model, the $5-10 budget can be concentrated entirely on the 25% of queries that actually generate value. This validates the hybrid architecture: a pure cloud model would waste money on simple timers, while a pure local model would fail the 25% of complex tasks.

---

4. Cost Modeling and Budget Strategies

The user has set a target of **$5-10/month**. We must rigorously model the token economics to ensure feasibility.

4.1 Token Economics

We utilize the pricing for **Claude 3.5 Haiku** (referred to as Haiku 4.5 in the prompt context) and the **Claude Agent SDK**.

- **Tier 1 (Qwen):** $0.00 / token.
- **Tier 2 (Haiku 4.5 API):** $1.00 per Million Input tokens / $5.00 per Million Output tokens.[18]
- **Tier 3 (Agent SDK - Max Plan):** The prompt states "Claude Agent SDK via Max Plan (free

but ~6s TTFT)". This is a critical distinction. It implies that **agentic capabilities are financially free** (likely covered by a separate flat subscription the user already has, or a specific beta plan) but **latency expensive**.

The Cost Equation:

$$Cost_{monthly} = (N_{Tier2} \times (T_{in} \times P_{in} + T_{out} \times P_{out}))$$
Assumptions:

- Average Voice Query Length: 30 tokens (Input).
- Average System Prompt (Persona): 200 tokens (Input - sent every time).
- Average Response: 150 tokens (Output).
- **Total per Query:** ~230 Input / 150 Output.

Cost per Tier 2 Transaction:

- Input: $230 \text{ tokens} \times \$1.00/1M = \$0.00023$
- Output: $150 \text{ tokens} \times \$5.00/1M = \$0.00075$
- **Total:** $\approx \$0.001$ (One tenth of a cent).

4.2 Budget Allocation Scenarios

With a $10 budget, ATLAS can afford **~10,000 Tier 2 queries per month**.

- This equates to **~330 Cloud queries per day**.
- Given the 75/25 distribution derived in Section 3.3, if ATLAS handles 330 Cloud queries, it implies a total volume of $330 \times 4 = 1320$ interactions per day.
- **Conclusion:** This capacity is virtually unlimited for a single personal user. The user is unlikely to speak to the assistant 1000+ times a day.

However, if the user engages in "Long Context" tasks (e.g., pasting a 50-page PDF into the context via a companion app), costs spike.

- 100k token context window = $0.10 per query.
- Budget allows for only ~100 such queries per month (3 per day).

4.3 Cost Tracking Implementation (The Budget Watchdog)

Despite the generous capacity for short queries, a "runaway script" or accidental loop could drain the budget. We require a **Token Bucket Rate Limiter**.[19]

Implementation:
A CostTracker class records usage.

- **Soft Limit:** At $8.00 spend, the system switches to "Thrifty Mode."
- **Thrifty Mode:** The Semantic Router thresholds are raised. Ambiguous queries that *might* have gone to Cloud are forced to Local.

- **Hard Limit:** At $10.00, Tier 2 is disabled. The Persona responds: *"I am afraid my connection to the external archives is temporarily severed. I must rely on my own wits."*

ROI Analysis:

- **Qwen:** Infinite ROI (Free).
- **Haiku:** High ROI (Excellent intelligence for $0.001/query).
- **Agent SDK:** High Latency Cost. Its "Free" financial cost makes it attractive, but the 6s wait time destroys the conversational flow. It should strictly be reserved for asynchronous tasks ("Do this and tell me when done") rather than synchronous conversation.

---

5. Adaptive Routing (Learning Over Time)

A static router is brittle. If the user consistently asks "Calculate the trajectory" and the router sends it to Local (which fails), the system must learn to upgrade that query type to Cloud.

5.1 The Feedback Loop

In a voice interface, explicit feedback (buttons) is absent. We must rely on **Implicit Feedback Signals**:

1. **Reformulation (Negative):** If the user repeats a query within 15 seconds with slight modification, the previous route likely failed.
    - *Action:* Penalty to the previous tier's weight for that vector cluster.
2. **Interruption (Negative):** If the user stops the TTS mid-sentence ("Stop," "No"), the response was likely poor or verbose.
    - *Action:* Penalty.
3. **Positive Phatic (Positive):** "Thanks," "Good job," "Perfect."
    - *Action:* Reward.

5.2 Contextual Bandit Implementation

We model routing as a **Contextual Bandit Problem**.[20] The "Context" is the query vector; the "Arms" are the Tiers.

Algorithm: Linear Upper Confidence Bound (LinUCB)
LinUCB is ideal because it generalizes. If the system learns that "Coding questions" (vector cluster A) perform better on Cloud, it will route future, unseen coding questions to Cloud, even if the specific words differ.

- **Mechanism:**
    - Maintain a weight matrix θ for each arm (Local, Cloud).
    - For a new query vector x, calculate expected payoff: $p = x^T \theta$.
    - Add an "Exploration Term" $\alpha \sqrt{x^T A^{-1} x}$ to account for uncertainty.

- Select the arm with the highest $p + \text{exploration}$.

This allows ATLAS to "explore" (try Local for a complex task occasionally) but generally "exploit" (stick to what works).

5.3 Memory Requirements

The LinUCB matrices are small ($d \times d$ where d is embedding dimension). For MiniLM ($d=384$), the matrix is roughly $150k$ floats (≈ 600 KB). This easily fits in memory and can be updated online.

---

6. Failure Handling and Graceful Degradation

The "Lethal Gentleman" persona requires that ATLAS never appears confused or broken. Errors must be absorbed into the persona.

6.1 Failure Scenarios

Scenario	Technical Event	Persona Response (R21)
Cloud Timeout	API takes > 3s	*"The external lines are congested, sir. I shall attempt to recall the information from my own memory."* (Fallback to Qwen)
API Error	500/403 Error	*"I am unable to access that specific archive at the moment. Perhaps we can approach this differently?"*
Misunderstanding	User corrects ATLAS	*"My apologies. I seem to have missed the nuance. Let me reconsider."* (Force Route to Cloud)
Agent Latency	SDK task starts	*"I shall attend to this matter immediately. It may require some time to coordinate the details."* (Acknowledges

		delay)

6.2 The "Poker Face" Protocol (Latency Masking)

The 3s latency budget is tight for Cloud queries. We use **Filler Phrases** to mask the processing time.

- **Immediate Ack:** When the router selects Cloud, the Local TTS *immediately* says: *"Let me see..."* or *"One moment..."*
- **Psychology:** This "buys" 1.5 seconds of audio time. The user perceives the system as responding instantly, even if the actual answer takes 3 seconds to generate. This keeps the *perceived* latency low.

6.3 Circuit Breaker Design

To prevent cascading failures (e.g., if Anthropic is down), we implement a **Circuit Breaker**.[22]

- If 3 consecutive Cloud requests fail/timeout, the breaker moves to **Open State**.
- **Open State:** The Router *automatically* downgrades all Tier 2 queries to Tier 1 (Local) for 5 minutes.
- **User Notification:** *"Sir, I am currently operating with limited connectivity. Some advanced analysis may be unavailable."*

---

7. Implementation Architecture

This section translates the strategy into a concrete software stack optimized for the WSL2/4GB VRAM environment.

7.1 Hardware & Environment Constraints

The 4GB VRAM limit is the primary bottleneck. A standard float16 (FP16) 3B model requires ~6GB VRAM. It will not load.

- **Solution:** We MUST use **4-bit Quantization (Q4_K_M)** or similar (GPTQ/AWQ).[3]
- **Impact:** A 4-bit Qwen2.5-3B model occupies ~2.2 GB VRAM. This leaves ~1.8 GB for the Context Window (KV Cache).
- **Context Limit:** This roughly allows for a 4096-token context window on the GPU. This is sufficient for conversation.

WSL2 Configuration:
The Linux kernel in WSL2 often fights Windows for RAM. We must clamp it.23

- Create C:\Users\[User]\.wslconfig:
  Ini, TOML

```
[wsl2]
memory=12GB   # Limit WSL to 12GB, leave 4GB for Windows/Voice I/O
swap=4GB
```

7.2 Component Design (Hexagonal Architecture)

The system is composed of decoupled services communicating via a fast local bus (e.g., ZeroMQ or HTTP/REST).

1. **Input Adapter (The Ear):**
   - **Microphone Stream:** PyAudio.
   - **VAD (Voice Activity Detection):** silero-vad (extremely lightweight, runs on CPU).
   - **ASR:** faster-whisper (small model, quantized int8). Runs on CPU to save VRAM. Latency ~200-400ms.
2. **The Core (The Brain):**
   - **Orchestrator:** Python application.
   - **Router:** AtlasRouter (Semantic + Reflex).
   - **State Manager:** Redis (stores conversation history).
3. **Model Services (The Intelligence):**
   - **Local Service: Ollama** running qwen2.5:3b-quant. Exposed on port 11434.
   - **Cloud Service:** Wrapper around anthropic Python client.
   - **Agent Service:** Async worker for the Claude Agent SDK.
4. **Output Adapter (The Voice):**
   - **TTS: Piper TTS** (High quality, runs faster than real-time on CPU). This is crucial to avoid VRAM contention.

7.3 Detailed Data Flow

1. **User:** "ATLAS, summarize this email."
2. **ASR:** Transcribes audio (300ms).
3. **Reflex:** Regex check (Failed) -> **Router:** Embeds query (20ms).
4. **Router Logic:** Vector matches "Summarization" prototype -> Score 0.88 -> Select **Tier 2 (Cloud)**.
5. **Masking:** Orchestrator sends "One moment..." to TTS immediately.
6. **Cloud Call:** Request sent to Haiku 4.5 (500ms TTFT).
7. **Streaming:** Tokens arrive. TTS streams audio buffer.
8. **Completion:** Cost updated. Bandit weights updated based on successful completion.

7.4 Configuration Structure

Configuration should be externalized (YAML/JSON) to allow tuning without code changes.

YAML

```yaml
system:
  persona: "lethal_gentleman"
  latency_budget_ms: 3000

router:
  model: "all-MiniLM-L6-v2"
  thresholds:
    cloud: 0.82
    agent: 0.88
  reflex_patterns:
    - "turn on.*"
    - "stop"

budget:
  monthly_limit_usd: 10.00
  daily_soft_limit: 0.33
  current_spend: 0.00

hardware:
  gpu_layers: 35 # Offload all Qwen layers to GPU
  context_size: 4096
```

---

8. Testing and Validation

To certify the system for deployment, a structured validation regime is required.

8.1 Benchmark Dataset (The "Golden Set")

We create a custom test set of 100 queries reflecting the Taxonomy:

- 50 Category A (Valet) - Expected: Local.
- 40 Category B (Consigliere) - Expected: Cloud.
- 10 Category C (Operator) - Expected: Agent.

8.2 Metrics to Track

1. **Routing Accuracy:** $ \frac{\text{Correctly Routed}}{\text{Total Queries}} $. Target > 90%.
2. **Latency (P95):** The time from "Voice End" to "System Audio Start." Target < 3000ms.

3. **Leakage Rate:** The % of Simple queries that accidentally go to Cloud (wasting money). Target < 5%.
4. **Miss Rate:** The % of Complex queries that accidentally go Local (stupid answers). Target < 5%.

8.3 A/B Testing (Serial)

Since the user is the only subject, we perform **Time-Series A/B Testing**.

- **Week 1 (Baseline):** Semantic Threshold 0.75.
- **Week 2 (Conservative):** Raise Threshold to 0.85.
- **Metric:** Compare "Reformulation Rate" (User annoyance) vs. "Daily Cost".
- **Optimization:** Find the threshold where Cost is minimized before Reformulation Rate spikes.

---

9. Research Sources

This report synthesizes data from the following key sources:

- **Routing & Classification:** [1] *Survey on Routing Strategies*[2] *Semantic Router Benchmarks*.
- **Hardware & Models:** [10] *Qwen2.5 Performance*[3] *GPU Requirements*[13] *Haiku Pricing*.
- **Voice & Taxonomy:** [14] *CLINC150*[5] *Voice Search Stats*.
- **Adaptive Systems:** [20] *Contextual Bandits*[22] *Circuit Breakers*.
- **Persona:** [25] *Alfred Pennyworth Quotes*.

---

10. Conclusion

The ATLAS system, as defined by the constraints of the user, is a feasible and highly capable architectural design. By rejecting the premise that "smarter is always better" and instead embracing **"smarter routing is better,"** we can deliver a high-end AI experience on consumer hardware and a coffee-shop budget.

The key to success lies in the **Semantic Cascade**. By filtering 75% of traffic through the zero-cost Local Tier, we liberate the budget to apply high-quality Cloud intelligence where it truly matters. The "Lethal Gentleman" persona serves not just as a character, but as a functional UI element—masking latency, managing expectations, and creating a cohesive user experience that feels far more expensive than it actually is.

Final Recommendation: Proceed with the implementation of the Python-based Orchestrator using all-MiniLM-L6-v2 for routing and qwen2.5:3b-quant on Ollama as the foundation. The resulting system will be robust, responsive, and undeniably gentlemanly.

Report End
Author: ATLAS Systems Architect
Date: January 2026

Detailed Technical Appendix: Implementation Code Patterns

A.1 The Cost Tracker (Singleton Pattern)

Python

```python
import time
from threading import Lock

class CostTracker:
    _instance = None
    _lock = Lock()

    PRICE_IN = 1.0 / 1_000_000 # $1 per M
    PRICE_OUT = 5.0 / 1_000_000 # $5 per M

    def __new__(cls):
        if cls._instance is None:
            with cls._lock:
                if cls._instance is None:
                    cls._instance = super(CostTracker, cls).__new__(cls)
                    cls._instance.daily_spend = 0.0
                    cls._instance.monthly_spend = 0.0
        return cls._instance

    def track_transaction(self, input_tokens, output_tokens):
        cost = (input_tokens * self.PRICE_IN) + (output_tokens * self.PRICE_OUT)
        with self._lock:
            self.daily_spend += cost
            self.monthly_spend += cost

    def can_afford_cloud(self):
        # Soft limit check
```

```
        return self.daily_spend < 0.33
```

A.2 The Circuit Breaker Decorator

Python

```
import functools
import time

class CircuitBreakerOpen(Exception):
    pass

def circuit_breaker(max_failures=3, reset_timeout=300):
    def decorator(func):
        failures = 0
        last_failure_time = 0
        state = "CLOSED" # CLOSED, OPEN, HALF_OPEN

        @functools.wraps(func)
        def wrapper(*args, **kwargs):
            nonlocal failures, last_failure_time, state

            if state == "OPEN":
                if time.time() - last_failure_time > reset_timeout:
                    state = "HALF_OPEN"
                else:
                    raise CircuitBreakerOpen("Circuit is OPEN")

            try:
                result = func(*args, **kwargs)
                if state == "HALF_OPEN":
                    state = "CLOSED"
                    failures = 0
                return result
            except Exception as e:
                failures += 1
                last_failure_time = time.time()
                if failures >= max_failures:
                    state = "OPEN"
                raise e
```

```
    return wrapper
  return decorator
```

A.3 The Adaptive Router Update (Bandit Logic)

Python

```
def update_weights(self, query_vec, arm_selected, user_feedback):
    # user_feedback: +1 (Positive), -1 (Negative/Reformulation)

    learning_rate = 0.01

    if arm_selected == "TIER_1_LOCAL":
        # Simple Gradient Descent update
        # If feedback is negative, we want the score for this vector to be LOWER next time
        # So we move weights in opposite direction of query_vec
        self.local_weights += learning_rate * user_feedback * query_vec

    elif arm_selected == "TIER_2_CLOUD":
        # If Cloud was good, boost it. If Cloud was bad, reduce it.
        self.cloud_weights += learning_rate * user_feedback * query_vec

    # Normalize weights to prevent explosion
    self.local_weights /= np.linalg.norm(self.local_weights)
    self.cloud_weights /= np.linalg.norm(self.cloud_weights)
```

Works cited

1. Doing More with Less: A Survey on Routing Strategies for Resource Optimisation in Large Language Model-Based Systems - arXiv, accessed on January 6, 2026, https://arxiv.org/html/2502.00409v3
2. vLLM Semantic Router: Improving efficiency in AI reasoning | Red Hat Developer, accessed on January 6, 2026, https://developers.redhat.com/articles/2025/09/11/vllm-semantic-router-improving-efficiency-ai-reasoning
3. GPU System Requirements Guide for Qwen LLM Models (All Variants), accessed on January 6, 2026, https://apxml.com/posts/gpu-system-requirements-qwen-models
4. Quantizing Qwen VL Models Using BitsAndBytes (bnb) for Efficient Inference with Small GPU | by Ranjith B | Medium, accessed on January 6, 2026, https://medium.com/@krshranjith/quantizing-qwen-models-using-bitsandbytes-

bnb-for-efficient-inference-b937286b9bbf
5. 72 Voice Search Statistics for 2026 - SeoProfy, accessed on January 6, 2026, https://seoprofy.com/blog/voice-search-statistics/
6. Candle Inference ~8.5x Slower Than PyTorch on CPU · Issue #2877 - GitHub, accessed on January 6, 2026, https://github.com/huggingface/candle/issues/2877
7. Supercharge Your Transformers with Model2Vec: Shrink by 50x, Run 500x Faster - Medium, accessed on January 6, 2026, https://medium.com/@hrishikesh19202/supercharge-your-transformers-with-model2vec-shrink-by-50x-run-500x-faster-c640c6bc1a42
8. Intent Classification in <1ms: How We Built a Lightning-Fast Classifier with Embeddings | by Durgesh Rathod | Medium, accessed on January 6, 2026, https://medium.com/@durgeshrathod.777/intent-classification-in-1ms-how-we-built-a-lightning-fast-classifier-with-embeddings-db76bfb6d964
9. When to Reason: Semantic Router for vLLM - arXiv, accessed on January 6, 2026, https://arxiv.org/html/2510.08731v1
10. Qwen2.5-3B: Specifications and GPU VRAM Requirements - ApX Machine Learning, accessed on January 6, 2026, https://apxml.com/models/qwen2-5-3b
11. Rational Tuning of LLM Cascades via Probabilistic Modeling - arXiv, accessed on January 6, 2026, https://arxiv.org/html/2501.09345v1
12. Learning to Route LLMs with Confidence Tokens - arXiv, accessed on January 6, 2026, https://arxiv.org/html/2410.13284v2
13. Claude 3.5 Haiku: Pricing, Context Window, Benchmarks, and More - LLM Stats, accessed on January 6, 2026, https://llm-stats.com/models/claude-3-5-haiku-20241022
14. Few-Shot-Intent-Detection - GitHub, accessed on January 6, 2026, https://github.com/jianguoz/Few-Shot-Intent-Detection
15. shriadke/Intent_Classification_using_CLINC150_Dataset: A short intent classification project with CLINC 150 dataset using Machine Learning - GitHub, accessed on January 6, 2026, https://github.com/shriadke/Intent_Classification_using_CLINC150_Dataset
16. The intent labels and the number of ut- terances in each label in Snips. - ResearchGate, accessed on January 6, 2026, https://www.researchgate.net/figure/The-intent-labels-and-the-number-of-ut-terances-in-each-label-in-Snips_tbl1_335589601
17. 51 Voice Search Statistics 2026: New Global Trends - DemandSage, accessed on January 6, 2026, https://www.demandsage.com/voice-search-statistics/
18. Models overview - Claude Docs, accessed on January 6, 2026, https://platform.claude.com/docs/en/about-claude/models/overview
19. Token Bucket Throttling | CodeSignal Learn, accessed on January 6, 2026, https://codesignal.com/learn/courses/throttling-api-requests-3/lessons/token-bucket-throttling
20. Adaptive LLM Routing Under Budget Constraints - ACL Anthology, accessed on January 6, 2026, https://aclanthology.org/2025.findings-emnlp.1301.pdf
21. Multi-armed bandit implementation - Peter Roelants, accessed on January 6, 2026, https://peterroelants.github.io/posts/multi-armed-bandit-implementation/

22. Circuit Breaker Pattern - Azure Architecture Center | Microsoft Learn, accessed on January 6, 2026, https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker
23. How to configure memory limits in WSL2 - Willem's Fizzy Logic, accessed on January 6, 2026, https://fizzylogic.nl/2023/01/05/how-to-configure-memory-limits-in-wsl2
24. Qwen2.5 Speed Benchmark - Qwen - Read the Docs, accessed on January 6, 2026, https://qwen.readthedocs.io/en/v2.5/benchmark/speed_benchmark.html
25. Top Ten Alfred Pennyworth Quotes | Movies-Films-MotionPictures, accessed on January 6, 2026, https://moviesfilmsmotionpictures.com/2012/10/11/top-ten-alfred-pennyworth-quotes/