Project ATLAS: Comprehensive Architectural Analysis and Strategic Model Selection for Autonomous Life Assistant Deployment

1. Executive Summary

The development of ATLAS, an autonomous AI life assistant designed for operation within a Windows Subsystem for Linux 2 (WSL2) environment on highly constrained hardware, presents a complex engineering challenge that sits at the intersection of edge computing, large language model (LLM) optimization, and real-time signal processing. The specific hardware envelope—an NVIDIA RTX 3050 Ti with 4GB of VRAM and a total system RAM of 6GB—imposes a rigid "hard ceiling" on computational resources. This constraint landscape fundamentally dictates every architectural decision, from the selection of the core reasoning engine to the micro-optimization of the voice interaction pipeline. The primary operational requirement, a sub-3-second latency for voice interactions, necessitates a system design that prioritizes throughput and memory locality above raw theoretical parameter counts.

This research report provides a definitive and exhaustive analysis of the conflict between the established Qwen 2.5 7B model and the emerging Qwen3-4B architecture. Our analysis, grounded in extensive technical benchmarking and architectural review, concludes that the **Qwen3-4B (Quantized)** model is the only viable candidate for the ATLAS core. The Qwen 2.5 7B, while possessing superior theoretical reasoning capabilities, cannot reside entirely within the 4GB VRAM budget. The resulting necessity to offload model layers to system RAM—which is itself severely constrained and operating over the relatively slow PCIe bus—introduces latency penalties that categorically disqualify it from meeting the real-time voice interaction target. Conversely, Qwen3-4B, utilizing a modern dense architecture and trained on a massive 36-trillion-token dataset, fits comfortably within the VRAM envelope while maintaining competitive agentic performance.

Furthermore, this report delineates a comprehensive voice processing pipeline utilizing **Silero VAD**, **Moonshine STT**, and **Kokoro-82M TTS**. This combination is mathematically projected to achieve an end-to-end latency of approximately 2000 milliseconds, providing a safe margin within the 3-second budget. We also present a hybrid architectural strategy employing a local semantic router to manage "thinking" vs. "talking" modes, ensuring that complex reasoning tasks do not degrade the responsiveness of routine interactions. The findings herein serve as a blueprint for the successful deployment of ATLAS, transforming hardware limitations from a barrier into a driver for highly optimized, efficient system design.

2. The Constraint Landscape: Hardware and Environment Analysis

The foundation of the ATLAS project is defined not by the capabilities of the software, but by the rigid limitations of the hardware. Understanding the physics of data movement and storage within this specific environment is prerequisite to selecting the appropriate AI models.

2.1 The "4GB Death Zone" and VRAM Hierarchy

The NVIDIA RTX 3050 Ti Laptop GPU is equipped with 4GB of GDDR6 memory. In the context of modern Generative AI, this capacity is often referred to as the "4GB Death Zone" because the most efficient and capable "base" models typically cluster around the 7-to-8 billion parameter size.[1] To understand the severity of this constraint, one must analyze the anatomy of VRAM usage during LLM inference.

A model's memory footprint is not a single static number; it is a composite of several dynamic factors:

1. **Static Weights:** This is the storage required for the model's parameters (matrices). Even with aggressive 4-bit quantization, a 7B model requires approximately 4.0-4.8 GB of storage solely for weights.[2]
2. **KV Cache (Key-Value Cache):** This is the dynamic memory required to store the attention context for the ongoing conversation. As the conversation length (context) grows, the KV cache scales linearly. For a 1 million token context, the KV cache alone can consume over 100GB.[2] Even for a modest 4096-token context, the cache requires several hundred megabytes of high-speed memory to avoid re-computation.
3. **Activation Overhead:** This is the temporary scratchpad memory used during the computation of a single token's forward pass.
4. **System Reservation:** In a Windows environment, the GPU is not a dedicated compute device; it drives the display and the Desktop Window Manager (DWM). This typically reserves between 0.5GB and 1.0GB of VRAM, leaving only roughly 3.0GB to 3.5GB available for CUDA applications.[5]

This hierarchy reveals a critical bottleneck: a 7B model, even in its most compressed usable form, physically exceeds the available VRAM on the RTX 3050 Ti before a single token is generated or a single word of context is stored.

2.2 The Latency Penalty of System RAM Offloading

When a model exceeds VRAM capacity, inference engines like llama.cpp or Ollama utilize "layer offloading," where a portion of the model resides in system RAM and is computed by the CPU.[5] While this enables the model to run, it introduces a catastrophic performance

penalty for real-time applications.

The RTX 3050 Ti communicates with the system via the PCIe bus. Accessing weights stored in system RAM is orders of magnitude slower than accessing GDDR6 VRAM (tens of GB/s vs. hundreds of GB/s). Furthermore, the ATLAS host system has only 6GB of total system RAM. The Windows operating system and the WSL2 virtualization layer typically consume 3-4GB of this.[7] This leaves a scant 2GB for the offloaded model layers, the voice pipeline, and the application logic.

If the system runs out of physical RAM, it degrades further to disk swapping (using the SSD as RAM), which reduces inference speed to unusable levels (e.g., 0.5 tokens per second). Benchmarks indicate that partial CPU offloading on similar hardware drops generation speed from ~50+ tokens per second (TPS) to roughly 3-5 TPS.[5] For a voice assistant, a 5 TPS speed means a simple sentence takes seconds to generate, destroying the illusion of conversation and violating the 3-second latency requirement.

2.3 WSL2 Audio and I/O Considerations

Running ATLAS within WSL2 adds an additional layer of complexity regarding audio Input/Output (I/O). While WSL2 has improved support for passing audio devices from the Windows host to the Linux container (using PulseAudio or PipeWire), this translation layer introduces latency. The "round trip" time from the user speaking into the microphone, the audio buffer passing through Windows drivers, crossing the VM boundary into WSL2, and being picked up by the Voice Activity Detector (VAD) is non-trivial. The architecture must prioritize lightweight, Linux-native audio handling within the container to minimize this overhead. Heavy audio processing frameworks that introduce additional buffers must be avoided in favor of direct stream processing.[10]

---

3. Model Selection: The Qwen Conflict

The user's query highlights a specific conflict between utilizing the larger, proven Qwen 2.5 7B model and the smaller, newer Qwen3-4B. This section provides a comparative analysis based on architectural efficiency, memory compatibility, and task performance.

3.1 Qwen 2.5 7B: The Theoretical Powerhouse

Qwen 2.5 7B is widely regarded as a benchmark in the 7B parameter class, excelling in coding, mathematics, and instruction following.

- **Architecture:** It features 7.61 billion parameters and 28 layers, with a massive context window capability of up to 128k tokens.[12]
- **Performance:** In standard benchmarks (MMLU, HumanEval), it often outperforms larger models from previous generations.

- **Memory Reality:** The decisive factor for ATLAS is the memory footprint.
  - *FP16 (Full Precision):* ~15.2 GB. Impossible on 3050 Ti.
  - *Q4_K_M (Standard Quantization):* ~4.68 GB.[4] This exceeds the 4GB physical limit.
  - *Q3_K_S (Aggressive Quantization):* ~3.6 GB.[4] While this *might* theoretically fit into the 4GB card, the Windows display overhead (approx. 0.6GB) leaves only ~3.4GB available. Thus, even the heavily quantized version would likely trigger an Out-Of-Memory (OOM) error or force swap usage immediately upon loading.
- **Conclusion:** Qwen 2.5 7B is structurally incompatible with the requirement for **resident GPU inference**. Any attempt to run it will necessitate hybrid CPU/GPU processing, resulting in high latency (high Time-To-First-Token) and slow generation speeds that fail the voice interaction criteria.

3.2 Qwen3-4B: The Dense Efficiency Specialist

Released in April 2025, the Qwen3 series represents a shift toward parameter efficiency and "dense" intelligence.[14]

- **Architecture:** Qwen3-4B is a dense model with 4.0 billion total parameters (3.6 billion non-embedding parameters) and 36 layers.[16]
- **Training:** It was pre-trained on a massive dataset of 36 trillion tokens—double the amount used for Qwen 2.5—covering 119 languages.[18] This extensive training allows it to "punch above its weight class," compressing more knowledge into fewer parameters.
- **Memory Reality:**
  - *Q4_K_M (Standard Quantization):* ~2.50 GB.[19]
  - *Q5_K_M (High Quality):* ~2.82 GB.[19]
  - *Q6_K (Near Lossless):* ~3.1 GB.[20]
- **Viability:** The Q4_K_M version consumes only ~2.5GB of VRAM. Assuming 0.6GB for Windows overhead, the total VRAM usage is ~3.1GB. This leaves nearly **0.9 GB of free VRAM** specifically for the KV cache and activation overhead. This "breathing room" is critical. It allows ATLAS to maintain a conversation context of several thousand tokens purely in high-speed VRAM, ensuring consistent performance as the day progresses.
- **Performance:** Alibaba reports that Qwen3-4B rivals the performance of the previous generation's 7B models in general tasks and specifically excels in STEM and coding due to the dense training.[18] On the Berkeley Function Calling Leaderboard, it maintains a competitive score, making it suitable for agentic tasks like checking calendars or weather.[21]

3.3 The "Thinking" Mode and Latency Implications

A unique feature of the Qwen3 architecture is the integration of a "Thinking" mode, which mimics chain-of-thought reasoning to improve performance on complex logical problems.[12]

- **Mechanism:** In Thinking mode, the model generates internal reasoning tokens before producing the final answer. This improves accuracy but significantly increases the

Time-To-First-Token (TTFT).

- **Voice Impact:** For a voice assistant, latency is paramount. If the user asks, "What time is it?", and the model spends 2 seconds generating hidden "thinking" tokens, the user experiences a delay that feels like a system failure.
- **Strategy:** The Instruct version of Qwen3-4B supports a non-thinking mode for general dialogue.[17] ATLAS should leverage the **Instruct** model and default to the non-thinking mode for standard interactions to minimize latency. The "Thinking" mode should be reserved for complex queries handled asynchronously (discussed in Section 7).

3.4 Comparative Benchmark Summary

The following table summarizes the trade-offs, demonstrating why Qwen3-4B is the mandated choice for the 3050 Ti.

Feature	Qwen 2.5 7B (Q4_K_M)	Qwen3-4B (Q4_K_M)	Qwen3-4B (Q5_K_M)
Model Weight Size	4.68 GB [4]	2.50 GB [19]	2.82 GB [19]
VRAM Requirement	**> 4.0 GB** (Overflows)	**~ 2.5 GB** (Fits)	**~ 2.8 GB** (Fits)
Inference Mode	Hybrid (CPU + GPU)	Pure GPU	Pure GPU
Est. Generation Speed	3 - 5 Tokens/Sec	45 - 60 Tokens/Sec	40 - 50 Tokens/Sec
Context Capacity	Near Zero (Immediate OOM)	High (~4k-8k tokens)	Moderate (~2k-4k tokens)
Agentic Capability	High (BFCL ~70%)	Moderate (BFCL ~65%)	Moderate (BFCL ~67%)
Voice Latency Suitability	**Fail** (Too Slow)	**Pass** (Real-time)	**Pass** (Real-time)

Decision: The Qwen3-4B (Q4_K_M) is the superior engine for ATLAS. It provides the necessary balance of intelligence and speed, fitting entirely within the hardware's fast

memory to enable sub-3-second responses.

---

4. Voice Pipeline Engineering: The Latency Budget

To achieve a "sub-3-second" interaction, we must rigorously engineer the voice pipeline. This budget encompasses the entire round trip: from the user closing their mouth to the first audible syllable of the AI's response.

4.1 Latency Budget Breakdown (Target: 3000ms)

We allocate the time budget across the pipeline stages. The goal is to keep the total system latency (excluding network fetches) under 2000ms, leaving a 1000ms buffer for variable API latencies or system hiccups.

Pipeline Stage	Component	Latency Budget	Technology Selection	Notes
1. Audio Capture & VAD	Silero VAD	< 50ms	**Silero VAD v5**	Processes 30ms chunks in <1ms.[23]
2. Silence Threshold	VAD Logic	~300ms	**Configurable**	Wait time to confirm user stopped speaking.
3. Speech-to-Text (STT)	Moonshine Base	~500ms	**Moonshine (ONNX)**	Variable length processing is key.[24]
4. Intent Routing	Semantic Router	~50ms	**Local Classifier**	CPU-based BERT-tiny or regex.
5. LLM Prefill (TTFT)	Qwen3-4B	~400ms	**llama.cpp**	Pure GPU inference ensures speed here.

6. Sentence Gen	Qwen3-4B	~500ms	**llama.cpp**	Time to generate first complete sentence.
7. Text-to-Speech (TTS)	Kokoro	~200ms	**Kokoro-82M**	Latency as low as 50ms on GPU.[25]
8. Audio Buffer	System	~100ms	**PulseAudio**	Driver/OS overhead.
TOTAL		**~2100ms**		**Success**

4.2 Voice Activity Detection (VAD): Silero

The first line of defense against latency is accurate endpointing—knowing exactly when the user has finished a command. **Silero VAD** is the industry standard for high-performance, low-latency speech detection.

- **Performance:** It processes audio chunks (typically 30ms) in less than 1 millisecond on a single CPU thread.[23]
- **Implementation:** ATLAS should run Silero VAD locally on the CPU. This is computationally negligible and keeps the GPU free for the LLM. It is significantly faster and more robust to noise than WebRTC VAD.[26]
- **Optimization:** A "silence threshold" of 200-300ms is recommended. If the VAD detects silence for this duration, it triggers the STT engine.

4.3 Speech-to-Text (STT): The Moonshine Revolution

Traditional STT models like OpenAI's Whisper act as a bottleneck because they process audio in fixed 30-second chunks. Even if the user says "Stop," Whisper pads the audio to 30 seconds and processes the empty silence, wasting compute and time.24
Moonshine is a new family of ASR models optimized specifically for on-device, low-latency applications.24

- **Variable Length Processing:** Moonshine scales its computational demand proportionally to the input audio length. A 2-second command is processed significantly faster than a long sentence.
- **Benchmarks:** Moonshine demonstrates up to a **3x reduction in latency** compared to Whisper Tiny/Base while matching their accuracy.[24]
- **Deployment:** ATLAS should utilize **Moonshine Base** in the **ONNX** format. The ONNX runtime is highly optimized and can run efficiently on the CPU. Given the 4GB VRAM limit, running Moonshine on the CPU (utilizing the 6GB system RAM) is a strategic necessity to

preserve VRAM for Qwen3.

4.4 Text-to-Speech (TTS): Kokoro-82M

The output stage is critical for the user's perception of speed. **Kokoro-82M** is the current state-of-the-art for lightweight, high-quality TTS.[29]

- **Efficiency:** With only 82 million parameters, it is exceptionally small compared to models like Tortoise or XTTS.
- **Speed:** It can synthesize typical sentences in **40-70ms** on a consumer GPU.[29] Even on a CPU, it runs at 3-11x real-time speeds.
- **Streaming:** Crucially, Kokoro supports streaming audio output. ATLAS should not wait for the entire response to be synthesized. It should pipeline the LLM output directly to Kokoro and play audio chunks as soon as they are ready.
- **Implementation:** While Kokoro runs fastest on GPU, the VRAM constraints of ATLAS suggest running it on the CPU initially. A latency of ~500ms on CPU is acceptable given the savings in VRAM, though GPU offloading (if space permits) is the ideal upgrade path.[25]

---

5. Architectural Design: Hybrid & Local

Given the hardware constraints, ATLAS cannot rely on heavy orchestration frameworks. The architecture must be lean, integrated, and hybrid in its logic flow.

5.1 The Semantic Router Strategy

To resolve the conflict between the need for speed (routine commands) and the need for deep intelligence (complex planning), ATLAS will employ a **Local Semantic Router**.[31]

- **Concept:** A lightweight text classifier (running on CPU) analyzes the user input *before* it is sent to the LLM. This classifier determines the "intent" of the query.
- **Routing Logic:**
  - **Path A (Fast / "Talker"):** Intents like [Lights_Control], , .
    - *Action:* Route directly to Qwen3-4B with a concise system prompt and "Thinking" mode disabled.
    - *Goal:* Immediate response.
  - **Path B (Slow / "Thinker"):** Intents like , , ``.
    - *Action:* Trigger an asynchronous "Thinking" process. The system immediately responds, "I'm thinking about that, give me a moment." Then, it routes the query to Qwen3-4B (potentially with "Thinking" mode enabled or a different prompt) in the background.
    - *Goal:* Manage user expectations regarding latency for difficult tasks.

5.2 The Inference Backend: llama.cpp Server

We select **llama.cpp** as the inference backend over Ollama for ATLAS. While Ollama is user-friendly, it wraps llama.cpp and adds a Go runtime overhead that consumes additional system RAM. llama.cpp offers granular control over GPU layers and memory allocation.[4]

- **Configuration:**
  - --model: Qwen3-4B-Instruct-Q4_K_M.gguf
  - --n-gpu-layers: 99 (Force all layers to GPU).
  - --ctx-size: 4096 (Strict limit to preserve VRAM).
  - --flash-attn: Enabled (Reduces memory usage during inference).[33]
  - --cache-type-k q8_0: Quantize the KV cache Key to 8-bit to save VRAM.[2]
  - --cache-type-v q8_0: Quantize the KV cache Value to 8-bit.

5.3 System Integration via Python

The glue code for ATLAS should be a lightweight Python script running in WSL2.

- **Audio Handling:** Use sounddevice or pyaudio interacting with the PulseAudio server.
- **Concurrency:** Use Python's asyncio or threading to handle the VAD/STT input stream while simultaneously managing the TTS output stream. This "duplex" capability allows the user to interrupt ATLAS (barge-in) if necessary.

---

6. Memory Optimization: Surviving on 6GB RAM

The 6GB system RAM constraint is as critical as the VRAM limit. If the system swaps to disk, the voice pipeline will stutter.

6.1 Quantization Strategy

We must move beyond generic "Q4" labels and select specific quantization formats based on perplexity/size trade-offs.[4]

- **Weights:** Q4_K_M is the "Goldilocks" zone. It uses Q6_K for half of the attention.wv and feed_forward.w2 tensors, providing higher accuracy than legacy Q4_0 while remaining compact.
- **KV Cache:** Standard KV cache uses FP16 memory. For a 4096 context, this is manageable. However, enabling **Q8_0 KV Cache Quantization** is a safety net that reduces context memory usage by ~50% with negligible impact on generation quality.[2]

6.2 System Debloating and WSL2 Tuning

To maximize available RAM for ATLAS, the Windows/WSL2 environment must be tuned.

- **WSL2 Memory Cap:** Create a .wslconfig file in the Windows user directory to prevent

WSL2 from consuming *all* RAM and crashing the host.

```ini, TOML
[wsl2]
memory=4GB   ; Limit WSL2 to 4GB. Leave 2GB for Windows.
processors=4 ; Assign sufficient cores.
swap=8GB     ; Enable swap as a safety net against crashes.
```

- **Headless Mode:** Run the WSL2 instance without a GUI (no WSLg applications) to minimize the VRAM overhead of the Linux graphics buffer.
- **Process Priority:** In Windows Task Manager, set the priority of the vmmem process (WSL2) to "High" to ensure audio processing isn't preempted by background Windows tasks.

---

7. Scenario Simulation: The "Morning Scenario"

To validate the proposed architecture, we simulate the specific usage scenario requested: *"ATLAS, good morning. Check my calendar, summarize the weather, and tell me if I need an umbrella."*

7.1 Step-by-Step Execution Flow

1. **0ms - Audio Capture:** The user begins speaking. Silero VAD (CPU) detects speech onset immediately.
2. **3000ms - Speech End:** The user finishes the command.
3. **3300ms - Silence Confirmation:** VAD confirms 300ms of silence and commits the audio buffer.
4. **3350ms - STT Processing:** Moonshine Base (CPU) processes the ~3-second audio clip.
   - *Latency:* ~400ms.
   - *Result:* "ATLAS good morning check my calendar summarize the weather and tell me if I need an umbrella."
5. **3750ms - Semantic Routing:** The router identifies the intent as a `` composite task.
   - *Action:* Trigger tool retrieval scripts (Python).
6. **3800ms - Data Retrieval:**
   - Script fetches Calendar JSON: {"events":}.
   - Script fetches Weather JSON: {"condition": "Rainy", "precip_prob": 80}.
7. **3850ms - LLM Prompt Construction:** The application constructs the prompt with the retrieved data injected into the context.
8. **3900ms - LLM Inference (Qwen3-4B GPU):**
   - *Input:* System Prompt + User Query + Tool Data.
   - *TTFT:* ~100ms (Pure GPU inference).
   - *Generation:* "Good morning. You have a meeting at 10 AM. It is currently rainy, so yes, you will need an umbrella."

9. **4050ms - TTS Streaming:**
    - As the LLM generates "Good morning," the text is sent to Kokoro (CPU).
    - *Synthesis:* Kokoro generates the first audio chunk in ~150ms.
10. **4200ms - Audio Playback:** The user hears the response.

7.2 Result Analysis

The "Time to Audio" is approximately **1.2 seconds** after the VAD confirmation (4200ms - 3000ms). This feels instantaneous and fluid to the user, well within the 3-second goal.

Contrast with Qwen 2.5 7B:
If Qwen 2.5 7B were used, step 8 (LLM Inference) would be the bottleneck.

- The model would be partially offloaded to CPU.
- The prompt processing (step 8) would take ~1000-1500ms due to slow memory transfer.[5]
- The token generation would proceed at ~4 TPS.
- The gap between "Good" and "morning" would be noticeable.
- Total latency would exceed 5-6 seconds, failing the user experience test.

---

8. Alternatives and Future Proofing

While Qwen3-4B is the primary recommendation, it is prudent to consider alternatives if specific constraints shift.

8.1 Microsoft Phi-3.5 Mini (3.8B)

- **Pros:** Excellent reasoning capabilities for its size; often benchmarks highly in logic.
- **Cons:** The 128k context window version is heavy; the standard version is comparable to Qwen. However, Qwen typically excels in "chat" and "persona" consistency, which is vital for a life assistant "ATLAS" persona.[35]
- **Verdict:** A strong runner-up. If Qwen3 proves too "dry," Phi-3.5 is the backup.

8.2 Llama-3.2 3B

- **Pros:** Highly optimized for edge devices; very fast.
- **Cons:** 3B parameters is often the threshold where nuance is lost. It may struggle with the complex instruction following required for the "Morning Scenario" (combining calendar + weather + reasoning about the umbrella) compared to the dense 4B Qwen3.
- **Verdict:** Use only if Qwen3-4B proves too slow on the specific machine.

8.3 Qwen3-MoE (30B-A3B)

- **Status:** Mentioned in research.[14]
- **Analysis:** While it only uses ~3B active parameters (fast inference), the *total* weight size is ~30B parameters. It requires ~15-20GB of disk/RAM to load.

- **Verdict: Impossible** to run on a 6GB RAM system. Do not attempt.

---

9. Conclusion and Implementation Guide

The feasibility of ATLAS hinges entirely on respecting the "physics" of the hardware. The RTX 3050 Ti's 4GB VRAM is a strict boundary. Crossing it invokes the latency penalties of the system bus and CPU, destroying the voice experience.

Definitive Recommendations:

1. **Model: Qwen3-4B-Instruct**. It is the only modern, dense model that offers agentic intelligence while fitting entirely within the 4GB VRAM budget with room for context.
2. **Format: GGUF (Q4_K_M)**. This quantization offers the optimal balance of intelligence and size (~2.5GB).
3. **Voice Stack: Silero VAD** (CPU) + **Moonshine Base** (CPU/ONNX) + **Kokoro-82M** (CPU/ONNX). This "CPU-heavy" voice stack reserves the precious GPU exclusively for the LLM.
4. **Architecture:** Implement a **Semantic Router** to distinguish between fast "Talker" tasks and slow "Thinker" tasks, masking latency for complex operations.
5. **System Tuning:** Use .wslconfig to cap WSL2 memory usage and prioritize the inference process.

By adhering to this blueprint, ATLAS can achieve the elusive goal of a responsive, autonomous, and intelligent life assistant on consumer-grade hardware, turning constraints into a showcase of efficient engineering.

Works cited

1. Choosing the Right GPU for LLMs on Ollama - Database Mart, accessed on January 5, 2026, https://www.databasemart.com/blog/choosing-the-right-gpu-for-popluar-llms-on-ollama
2. Amount of ram Qwen 2.5-7B-1M takes? : r/LocalLLaMA - Reddit, accessed on January 5, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1j79o3l/amount_of_ram_qwen_257b1m_takes/
3. bartowski/FuseChat-Qwen-2.5-7B-Instruct-GGUF - Hugging Face, accessed on January 5, 2026, https://huggingface.co/bartowski/FuseChat-Qwen-2.5-7B-Instruct-GGUF
4. bartowski/Qwen2.5-7B-Instruct-GGUF - Hugging Face, accessed on January 5, 2026, https://huggingface.co/bartowski/Qwen2.5-7B-Instruct-GGUF
5. Finally managed to run Qwen-2.5-7B on a 4GB GTX 1050 without CPU offloading (Surgical Memory Alignment) : r/LocalLLaMA - Reddit, accessed on January 5, 2026,

https://www.reddit.com/r/LocalLLaMA/comments/1po97ad/finally_managed_to_run_qwen257b_on_a_4gb_gtx_1050/
6. llama.cpp: Automation for GPU layers, tensor split, tensor overrides, and context size (with MoE optimizations) : r/LocalLLaMA - Reddit, accessed on January 5, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1pn2e1c/llamacpp_automation_for_gpu_layers_tensor_split/
7. Running Local LLMs, CPU vs. GPU - a Quick Speed Test - DEV Community, accessed on January 5, 2026, https://dev.to/maximsaplin/running-local-llms-cpu-vs-gpu-a-quick-speed-test-2cjn
8. Qwen3-30B-A3B runs at 12-15 tokens-per-second on CPU : r/LocalLLaMA - Reddit, accessed on January 5, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1kag4er/qwen330ba3b_runs_at_1215_tokenspersecond_on_cpu/
9. Qwen 2.5 CPU vs GPU comparison : r/LocalLLaMA - Reddit, accessed on January 5, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1fq883g/qwen_25_cpu_vs_gpu_comparison/
10. Show HN: Real-time AI Voice Chat at ~500ms Latency - Hacker News, accessed on January 5, 2026, https://news.ycombinator.com/item?id=43899028
11. I built RealtimeVoiceChat because I was frustrated with the latency in most voic... | Hacker News, accessed on January 5, 2026, https://news.ycombinator.com/item?id=43899029
12. Qwen - Wikipedia, accessed on January 5, 2026, https://en.wikipedia.org/wiki/Qwen
13. Qwen2.5 7B Instruct vs Qwen3 VL 4B Instruct - LLM Stats, accessed on January 5, 2026, https://llm-stats.com/models/compare/qwen-2.5-7b-instruct-vs-qwen3-vl-4b-instruct
14. Qwen 3 Benchmarks, Comparisons, Model Specifications, and More ..., accessed on January 5, 2026, https://dev.to/best_codes/qwen-3-benchmarks-comparisons-model-specifications-and-more-4hoa
15. Qwen3 4B - Open Laboratory, accessed on January 5, 2026, https://openlaboratory.ai/models/qwen3-4b
16. Qwen/Qwen3-4B - Hugging Face, accessed on January 5, 2026, https://huggingface.co/Qwen/Qwen3-4B
17. arXiv:2505.09388v1 [cs.CL] 14 May 2025, accessed on January 5, 2026, https://arxiv.org/pdf/2505.09388
18. Qwen3: Think Deeper, Act Faster | Qwen, accessed on January 5, 2026, https://qwenlm.github.io/blog/qwen3/
19. bartowski/Qwen_Qwen3-4B-GGUF - Hugging Face, accessed on January 5, 2026, https://huggingface.co/bartowski/Qwen_Qwen3-4B-GGUF
20. bartowski/Qwen2.5-3B-GGUF - Hugging Face, accessed on January 5, 2026,

https://huggingface.co/bartowski/Qwen2.5-3B-GGUF
21. BFCL-v3 Leaderboard - LLM Stats, accessed on January 5, 2026, https://llm-stats.com/benchmarks/bfcl-v3
22. Reasoning through Exploration: A Reinforcement Learning Framework for Robust Function Calling - arXiv, accessed on January 5, 2026, https://arxiv.org/html/2508.05118v4
23. Silero VAD: pre-trained enterprise-grade Voice Activity Detector - GitHub, accessed on January 5, 2026, https://github.com/snakers4/silero-vad
24. Moonshine: Speech Recognition for Live Transcription and Voice Commands - arXiv, accessed on January 5, 2026, https://arxiv.org/html/2410.15608v1
25. Concurrency Support · Issue #286 · KoljaB/RealtimeTTS - GitHub, accessed on January 5, 2026, https://github.com/KoljaB/RealtimeTTS/issues/286
26. Voice Activity Detection (VAD): The Complete 2025 Guide to Speech Detection, accessed on January 5, 2026, https://picovoice.ai/blog/complete-guide-voice-activity-detection-vad/
27. One Voice Detector to Rule Them All - The Gradient, accessed on January 5, 2026, https://thegradient.pub/one-voice-detector-to-rule-them-all/
28. Introducing Moonshine, the new state of the art for speech to text | Pete Warden's blog, accessed on January 5, 2026, https://petewarden.com/2024/10/21/introducing-moonshine-the-new-state-of-the-art-for-speech-to-text/
29. Kokoro TTS Studio: Free Online Text-to-Speech Demo, accessed on January 5, 2026, https://unrealspeech.com/studio
30. 12 Best Open-Source TTS Models Compared (2025): Latency, Quality, Voice Cloning & More - Inferless, accessed on January 5, 2026, https://www.inferless.com/learn/comparing-different-text-to-speech---tts--models-part-2
31. Bringing intelligent, efficient routing to open source AI with vLLM Semantic Router - Red Hat, accessed on January 5, 2026, https://www.redhat.com/en/blog/bringing-intelligent-efficient-routing-open-source-ai-vllm-semantic-router
32. aurelio-labs/semantic-router: Superfast AI decision making and intelligent processing of multi-modal data. - GitHub, accessed on January 5, 2026, https://github.com/aurelio-labs/semantic-router
33. GPU requirements for running Qwen2.5 72B locally? : r/LocalLLM - Reddit, accessed on January 5, 2026, https://www.reddit.com/r/LocalLLM/comments/1pxjvjy/gpu_requirements_for_running_qwen25_72b_locally/
34. Qwen2.5 7B Instruct GGUF · Models - Dataloop, accessed on January 5, 2026, https://dataloop.ai/library/model/bartowski_qwen25-7b-instruct-gguf/
35. Prompt Engineering for Small LLMs: LLaMA 3B, Qwen 4B, and Phi-3 Mini | by Malik Naik, accessed on January 5, 2026, https://maliknaik.medium.com/prompt-engineering-for-small-llms-llama-3b-qwen-4b-and-phi-3-mini-de711d38a002
36. Qwen 3 Performance: Quick Benchmarks Across Different Setups : r/LocalLLaMA

- Reddit, accessed on January 5, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1kdsp4z/qwen_3_performance_quick_benchmarks_across/