Local LLM & Voice Integration Architecture

1. Background

Hearthlink supports fully local AI inference and voice interaction, enabling modules (Alice, Mimic, Alden, Sentry) to operate offline or with minimal external dependencies. This spec defines a dual-model LLM strategy, workspace integration, voice I/O pipelines, and dynamic resource profiling to optimize performance and UX.

2. Requirements

Must have

- Embedded lightweight LLM (1–2 B quant) always loaded for fast basic tasks (micro-LLM)
- Optional heavyweight LLM (7–13 B quant) on GPU or remote fallback
- Secure, isolated per-agent memory stores (short, long-term, shared)
- Voice capture, transcription, TTS pipelines with global default **ON**, toggle available in System Settings
- System-settings UI to configure model paths, voice engines, micro-LLM resource caps, and heavy-LLM thresholds

Should have

- Automatic system resource detection (CPU cores, RAM, GPU VRAM)
- Intelligent model recommendation based on workload and availability
- On-demand download/installation of model binaries and voice packages
- Audit logging of LLM loads and voice interactions in Vault

Could have

- Adaptive model swapping: escalate from micro to heavy LLM when confidence low
- Real-time voice noise reduction and echo cancellation profiles

Won’t have

- GPU acceleration for micro-LLM (CPU-only inference)

3. Architecture. Architecture

```
@startuml
package "LLM Manager" {
  [Resource Profiler]
  [Model Loader] --> [LightModel]
  [Model Loader] --> [HeavyModel]
  [Workspace API]
}
```

```
package "Voice Service" {
  [Mic Capture] --> [ASR Engine]
  [TTS Engine] --> [Speaker Output]
}
LLM Manager --> Modules
Voice Service --> Modules
Modules --> LLM Manager : LLM requests
Modules --> Voice Service : Voice I/O
@enduml
```

3.1 Resource Profiler

- Detect cores, RAM, GPU & VRAM on startup
- Expose `/v1/sys/resources` API
- Recommend model variants: light by default, heavy if GPU ≥6 GB

3.2 Model Loader

- Manage model directories under `~/.hearthlink/models/`: `micro/` for small LLMs, `heavy/` for large LLMs
- Download & verify model artifacts via checksums
- Load quantized GGML for CPU (micro-LLM) or ONNX for GPU (heavy-LLM)
- Default load behavior: on startup, automatically load micro-LLM into memory for quick tasks
- API endpoints:
- `GET /v1/llm/models`
- `POST /v1/llm/models/load` (specify `modelType`: micro or heavy)
- `DELETE /v1/llm/models/unload`
- **NEW** `POST /v1/llm/micro/infer` for micro-LLM basic tasks (max 2 threads, <2 GB RAM)

4. Memory Isolation & Routing. Memory Isolation & Routing

- **Short-Term Memory:** in-process cache per session, cleared on end
- **Long-Term Memory:** SQLite/JSON store per module under `~/.hearthlink/memory/{agent}`
- **Shared Memory:** document store for authorized cross-agent access
- Agents tag messages with `ttl` and `scope` flags; Core mirrors to central store if `scope=shared`

5. Voice Integration

```
+----------------------------------------------+
| System Settings → Voice Tab                  |
| [Voice On/Off Toggle]   [Engine ▼]            |
| [Mic Sensitivity Slider]  [Noise Suppression] |
| [TTS Voice ▼]   [Speech Rate Slider]         |
| [Test Mic]    [Play Sample]    [Save/Cancel]  |
+----------------------------------------------+
```

Component	Function	API/Data Call
VoiceToggle	Global ASR/TTS enable	`PUT /system/settings/voice/enabled`
EngineDropdown	Select ASR/TTS engine (e.g. VOSK, Coqui)	`PUT /system/settings/voice/engine`
MicSensitivitySlider	Adjust microphone gain	`PUT /system/settings/voice/micSensitivity`
NoiseSuppressionToggle	Enable echo/noise cancellation	`PUT /system/settings/voice/noiseCancel`
TTSVoiceDropdown	Choose synthetic voice profile	`PUT /system/settings/voice/ttsVoice`
SpeechRateSlider	Adjust TTS playback speed	`PUT /system/settings/voice/speechRate`
TestMicButton	Record & playback user mic	`POST /voice/test/mic`
PlaySampleButton	Play TTS sample phrase	`POST /voice/test/tts`

5. UI Customization & Multi-Modal Support

5.1 Multi-Modal Input Panel

```
+----------------------------------------------------------+
| LLM Multi-Modal Input                                     |
| [Text Tab][Voice Tab][Image Tab][File Tab]                |
|----------------------------------------------------------|
| • **Text:** Freeform prompt entry                         |
| • **Voice:** Record and transcribe voice input             |
| • **Image:** Drag-drop or upload image for OCR/analysis    |
| • **File:** Attach document (PDF, CSV) for summarization  |
+----------------------------------------------------------+
```

Component	Function	API/Data Call
InputModeTabs	Switch between Text, Voice, Image, File modes	N/A
TextPromptField	Standard prompt entry	N/A
VoiceRecordButton	Start/stop recording; stream to ASR	`POST /voice/stream`
ImageUploadButton	Select or drag image; send to vision inference	`POST /v1/llm/multi/image`

Component	Function	API/Data Call
FileUploadButton	Select document; send to extract & summarize	`POST /v1/llm/multi/file`

5.2 Model Parameter Controls

```
+-----------------------------------------------------------+
| Model Settings                                              |
| [Model ▼][Temperature ░░░][Top-P ░░░][Max Tokens ░░░]       |
+-----------------------------------------------------------+
```

Component	Function	API/Data Call
ModelSelectorDropdown	Choose loaded model (micro or heavy)	`POST /v1/llm/models/load`
TemperatureSlider	Control randomness (0–1)	`PUT /v1/llm/settings/temperature`
TopPSlider	Control nucleus sampling (0–1)	`PUT /v1/llm/settings/top_p`
MaxTokensInput	Limit response length	`PUT /v1/llm/settings/max_tokens`
ResetDefaultsButton	Restore default LLM parameters	`DELETE /v1/llm/settings`

5.3 Context Window & Memory Controls

```
+-----------------------------------------------------------+
| Context & Memory                                            |
| [Window Size ░░░] [Short-Term ▾] [Long-Term ▾] [Clear]     |
+-----------------------------------------------------------+
```

Component	Function	API/Data Call
WindowSizeSlider	Adjust context token window	`PUT /v1/llm/settings/context_window`
MemoryScopeDropdown	Choose memory scope (short-term, long-term)	`PUT /v1/llm/settings/memory_scope`
ClearMemoryButton	Purge selected memory store	`POST /v1/llm/memory/clear?scope={scope}`

5.5 Optimization & Auto-Tuning

```
+----------------------------------------------------------+
| LLM Optimization Panel                                    |
| [Auto-Tune ▼] [Profile Metrics] [Apply Recommendations]  |
|----------------------------------------------------------|
| • **Auto-Tune Mode:** [Off | Safe | Aggressive]           |
| • **Profile Metrics:** CPU %, GPU %, RAM usage, Latency   |
| • **Recommendations:** List of parameter changes          |
+----------------------------------------------------------+
```

Component	Function	API/Data Call
AutoTuneModeDropdown	Select optimization aggressiveness	`PUT /v1/llm/settings/auto_tune_mode`
ProfileMetricsPanel	Display real-time resource & performance metrics	`GET /v1/sys/resources?include=performance`
RecommendationsList	Show suggested parameter/ strategy adjustments	`GET /v1/llm/optimize/suggestions`
ApplyRecommendationsBtn	Apply selected recommendations	`POST /v1/llm/optimize/apply`

Optimization Logic:

- **Safe Mode:** Adjust only non-critical parameters (e.g., reduce chunk size, lower temp) to conserve resources without affecting output quality.
- **Aggressive Mode:** Dynamically switch to heavy LLM or increase threads/VRAM usage when underutilized, reduce padding, and enable advanced sampling strategies.
- **Profile Metrics:** Continuously monitor resource utilization and feedback into the optimization engine.
- **Suggestions API:** Returns prioritized adjustments based on current resource state and historical performance data (e.g., “reduce context window by 20% to lower latency”).

6. Implementation Steps. Implementation Steps. Implementation Steps

1. **Resource Profiler**: implement OS-agnostic detection (Node.js/WINAPI + `/proc`).
2. **Model Manager**: integrate with Ollama or Llama.cpp for downloads and loads.
3. **Workspace API**: reuse De minimus endpoints for file I/O.
4. **Voice Service**: embed VOSK/Coqui ASR and Coqui TTS in a local microservice.
5. **Settings UI**: extend System Settings in canvas with voice tab and model config.
6. **Audit Logging**: every load/unload and voice action emits to Vault.
7. **Testing**: unit tests for profiler, API, and E2E voice QA recordings.

7. Milestones

Milestone	Timeline	Owner
Profiler & Model Manager	Week 1	Core Team
Voice Microservice	Week 2	Voice Lead
Settings UI Integration	Week 3	Frontend Team
Memory Store Isolation	Week 4	Backend Team
Audit & Testing	Week 5–6	QA Team

8. Gathering Results

• Model load time <3 s for light model; <10 s for heavy model
• ASR latency <200 ms per utterance; TTS startup <50 ms
• Voice accuracy ≥90% in quiet and moderate noise
• Resource Profiler accuracy within 10% of true values

---

9. Alden UI Amendments

To enable Alden as the primary advisor for LLM and Voice settings, add the following voice-linked controls accessible from the persistent System Banner:

• **"Alden, open Settings"** → navigates to System Settings (Settings tab)
• **"Alden, show Memory & Storage"** → opens the Memory Isolation panel under LLM settings
• **"Alden, configure LLM"** → opens the Model Parameter Controls panel
• **"Alden, optimize LLM"** → opens the Optimization & Auto-Tuning panel with live metrics

Each voice command should be acknowledged by audio feedback (e.g., chime) and brief visual confirmation (highlighted banner icon). Alden's persona can narrate contextual help for each panel upon opening.

End of Local LLM & Voice Integration Spec