# Orchestrating "Ovi English School": A Local-First Autonomous Content Pipeline Architecture on Apple Silicon

## 1. Executive Summary and Architectural Philosophy

The convergence of high-performance localized hardware, specifically the Apple Silicon M-series architecture, and accessible workflow automation platforms like n8n has necessitated a re-evaluation of traditional cloud-native media pipelines. The project "Ovi English School" represents a quintessential case study in this shift: a daily, fully automated, multi-modal content generation system designed to operate entirely within the constraints and capabilities of a local MacBook Pro M5.

This report serves as a comprehensive technical blueprint for architecting this pipeline. Unlike traditional DevOps approaches that rely on distributed cloud microservices—where compute, storage, and orchestration are decoupled across vast data centers—this architecture adopts a "LocalOps" paradigm. Here, the MacBook Pro M5 acts as a monolithic, high-performance edge node. The unified memory architecture (UMA) of the M5, specifically the 24GB configuration, allows for zero-copy data sharing between the CPU (orchestrating via Node.js/n8n), the GPU (rendering via DaVinci Resolve or FFmpeg), and the Neural Engine (synthesizing audio via Qwen3-TTS).

The primary objective is to construct a resilient, self-healing automated system that fetches trending topics, synthesizes educational scripts using Large Language Models (LLMs), generates hyper-realistic audio, renders video assets, and distributes them across platforms (YouTube, RSS, Cloudflare R2), all triggered precisely at 06:00 JST daily. This report rigorously examines the installation environment, advocating for a bare-metal npm approach over Docker due to specific hardware access requirements, and details the implementation of all twelve pipeline stages with a focus on reliability and error handling.

## 2. The Computational Environment: Apple Silicon M5

### 2.1 The Strategic Advantage of Unified Memory Architecture

The selection of the MacBook Pro M5 with 24GB of Unified Memory is not merely a hardware choice but a defining architectural constraint that shapes the software stack. In a conventional x86 workstation, data must be copied between the system RAM and the discrete GPU VRAM over a PCIe bus, introducing latency and creating bottlenecks during heavy media processing. The M5’s Unified Memory Architecture eliminates this bottleneck.

For the "Ovi" pipeline, this is critical during **Stage 5 (Audio Generation)** and **Stage 7 (Video Rendering)**. The Python processes managing the Qwen3-TTS model can load the 1.7B or larger parameter weights directly into unified memory, where they are accessible by the Neural Engine for inference without redundant data duplication.1 Similarly, DaVinci Resolve, when functioning as the rendering engine, can access video buffers directly from the same memory pool.

However, the 24GB limit poses a specific challenge for concurrency. While a cloud architecture might spin up separate instances for TTS and Video Rendering to run in parallel, the local M5 architecture demands serial execution to avoid Out-Of-Memory (OOM) crashes. If the LLM (taking ~8-10GB), the TTS model (taking ~4-6GB), and the Operating System are active simultaneously, there is insufficient headroom for a 4K render in DaVinci Resolve. Consequently, the workflow design patterns detailed in Section 5 prioritize strict sequential processing over parallelism.

### 2.2 Neural Engine Utilization for Python Workloads

A key component of this pipeline is the local generation of audio using Qwen3-TTS. To achieve the requirement of daily throughput without overheating the machine or tying up the main CPU cores needed for n8n's orchestration, the pipeline must leverage the Metal Performance Shaders (MPS) backend.

Standard Python installations often default to CPU execution for PyTorch operations, which is inefficient. The configuration detailed in Section 3 ensures that the torch and torchaudio libraries are compiled with MPS support, allowing the Python scripts to offload tensor operations to the M5's specialized Neural Engine. This hardware acceleration is vital for keeping the "daily" generation time under a threshold that makes the 6:00 AM deadline viable.

## 3. Execution Environment: The Docker vs. npm Paradigm

The most significant architectural decision in deploying n8n on macOS is the choice between containerization (Docker) and native execution (npm/Node.js). While Docker is the industry standard for server-side deployments due to its reproducibility and isolation, the "LocalOps" requirements of the Ovi pipeline heavily favor a native approach.

### 3.1 Comparative Analysis of Runtimes

| **Feature** | **Docker Container** | **Native (npm) Installation** | **Architectural Implication** |
| --- | --- | --- | --- |
| **System Isolation** | **High.** Runs in a Linux VM layer (VirtioFS). Dependencies are sandboxed. | **Low.** Runs directly on macOS. Shares host libraries and kernel space. | Docker provides safety; npm provides access. |
| **Hardware Access (GPU/Neural Engine)** | **Restricted.** Accessing Metal/MPS from within a container is experimental and complex. | **Native.** Python scripts have direct access to the M5's Neural Engine via local PyTorch. | Native is essential for performant TTS and LLM inference.2 |
| **Application Control** | **Blocked.** Containers cannot launch or control host GUI apps (DaVinci Resolve). | **Seamless.** Can execute open, osascript, and control GUI apps via API. | Native is required to automate DaVinci Resolve Studio.3 |
| **Filesystem Performance** | **Moderate.** Volume mounting introduces I/O overhead on macOS. | **Maximum.** Direct NVMe SSD speeds for media read/write. | Native reduces render times for large video files. |
| **Maintenance Overhead** | **Low.** docker pull to update. | **Medium.** Managing Node versions and Python venvs manually. | Native requires more disciplined environment management. |

### 3.2 The Decision for Ovi English School: Native npm

Based on the specific requirement to utilize **DaVinci Resolve** and **Qwen3-TTS** locally, a Docker-based installation is functionally disqualified for this pipeline without resorting to extreme workarounds.

DaVinci Resolve's Python API requires the script to be executed in an environment that has access to the running Resolve process on the host machine.5 A Docker container is an isolated Linux environment that cannot "see" the macOS process table or interact with the window manager required to launch Resolve. Furthermore, while passthrough for Nvidia GPUs in Docker is mature, support for Apple Silicon's Metal acceleration inside containers remains in early developmental stages and is prone to instability.7

Therefore, the recommended architecture is a **bare-metal npm installation** of n8n, managing Python dependencies via strict virtual environments (venv) to emulate the isolation benefits of Docker while retaining native hardware access.

## 4. Installation and Configuration on Apple Silicon

To establish a robust "LocalOps" foundation, the installation process must go beyond simple binary installation. It requires configuring the macOS environment to act as a reliable server while retaining its desktop capabilities.

### 4.1 Prerequisites and System Dependencies

The backbone of the system relies on **Homebrew**, the macOS package manager. This ensures that the underlying tools—FFmpeg for media, Python for logic, and Node.js for orchestration—are kept up to date.

The installation begins with the core dependencies. ffmpeg is critical for audio post-processing (Step 6) and video transcoding (Step 7). python@3.12 provides the runtime for the AI models.

Bash

# Install Homebrew (if not present)  
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"  
  
# Install core binaries  
brew install node@20 ffmpeg python@3.12 sox portaudio

Installing sox and portaudio is a precautionary measure for the audio processing libraries used by Qwen3-TTS, which often require low-level audio driver access.8

### 4.2 Installing n8n via npm

Unlike the Docker image which comes pre-configured, the npm version requires manual setup to ensure persistence and accessibility.

Bash

# Global installation ensures the 'n8n' command is in the system PATH  
npm install n8n -g

To manage the local data effectively, specific environment variables must be configured. By default, n8n stores workflows in an SQLite database in the user's home directory. For a media-heavy pipeline, it is crucial to configure the binary data mode. n8n allows passing binary data (like the generated 50MB video file) either by memory or by filesystem references. Given the 24GB memory constraint and the large file sizes of video, forcing filesystem mode prevents the Node.js process from crashing due to heap exhaustion.

### 4.3 Python Virtual Environment Strategy

To maintain system hygiene, the pipeline's Python dependencies must not be installed in the global macOS Python environment. A dedicated virtual environment serves as the "container" for the AI workers.

Bash

# Create the project directory structure  
mkdir -p ~/OviPipeline/python\_workers  
mkdir -p ~/OviPipeline/assets  
mkdir -p ~/OviPipeline/temp\_media  
  
# Initialize the Virtual Environment  
cd ~/OviPipeline/python\_workers  
python3 -m venv venv  
  
# Activate and Install Specific AI Libraries  
source venv/bin/activate  
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu  
pip install qwen-tts transformers scipy soundfile beautifulsoup4 pandas reportlab requests pytrends

*Note on PyTorch:* The command above references the CPU wheel, but on a Mac, pip will automatically look for the version compatible with arm64. It is vital to verify that torch.backends.mps.is\_available() returns True after installation to ensure the Neural Engine will be utilized.1

### 4.4 System-Level Power Management

A pipeline scheduled for 06:00 AM JST faces the immediate hurdle of macOS power management. The operating system is aggressive about entering "sleep" or "power nap" modes to conserve battery, even when plugged in. n8n, as a user-level process, cannot wake the machine from deep sleep to trigger a workflow.

The solution requires a two-tiered approach:

1. **Scheduled Wake-Up:** The hardware must be instructed to power on the CPU subsystems prior to the job start. The pmset utility interacts with the Power Management Unit (PMU) directly.  
   Bash  
   sudo pmset repeat wake MTWRFSU 05:55:00  
     
   This command schedules a wake event for 05:55 AM every day (Monday-Sunday), giving the system 5 minutes to re-establish network connections and mount drives before the n8n trigger fires.10
2. **Execution Insomnia:** Once the workflow begins, the system must be prevented from sleeping until the rendering is complete. This is handled within the workflow logic (discussed in Section 5) using the caffeinate utility.12

## 5. Workflow Design Patterns

To manage complexity, the "Ovi" pipeline is structured using a modular pattern rather than a single linear flow. This improves debuggability and allows for partial re-runs (e.g., if the video render fails, one does not need to re-fetch the news).

### 5.1 The Orchestrator Pattern

n8n acts as the "General," while Python scripts act as the "Soldiers." n8n does not perform the heavy lifting (scraping, synthesis, rendering); it delegates these tasks to the local shell environment via the **Execute Command** node.

**The Input/Output Protocol:**

Communication between n8n and the Python scripts relies on stdin and stdout.

* **Input:** n8n passes parameters to Python scripts as command-line arguments (e.g., --topic "AI News").
* **Output:** Python scripts must print **only** a valid JSON string to stdout. Any logging or debug information must be printed to stderr.
* **Parsing:** n8n's Execute Command node captures the stdout and parses it. A subsequent code node or expression is often required to convert the stringified JSON back into a usable JavaScript object.14

### 5.2 The "LocalOps" Folder Structure

A disciplined folder structure is required for the scripts to locate assets without hardcoded paths breaking if the user directory changes.

* ~/OviPipeline/scripts/: Contains the functional Python logic (fetch.py, tts.py, render\_resolve.py).
* ~/OviPipeline/assets/: Static assets (Intro music, Channel Logo, Font files).
* ~/OviPipeline/temp/: Ephemeral storage for the daily run (cleared at the start of each execution).
* ~/OviPipeline/output/: Final archival of generated PDFs and MP4s.

## 6. Pipeline Implementation: The 12-Step Architecture

This section details the logic, tools, and configuration for each step of the Ovi English School pipeline.

### 6.1 Phase I: Ingestion and Intelligence (Steps 1-4)

**Goal:** Transform raw internet noise into a structured educational script.

#### Step 1: Fetch Trending Topics

* **Tool:** Python Script (fetch\_trends.py) using pytrends.
* **Mechanism:** The script queries Google Trends for specific keywords related to "Technology," "Science," or "Business" within the user's target region.
* **Why Local?** While n8n has HTTP nodes, pytrends handles the complexities of Google's unofficial API cookies and rate limiting more robustly than raw HTTP requests.
* **Output:** A list of 5 trending keywords.

#### Step 2: Fetch News Articles

* **Tool:** n8n **RSS Feed Read** Node.
* **Configuration:** The node is configured with a predefined list of high-quality RSS feeds (e.g., BBC Learning English, TechCrunch, The Verge).
* **Filtering:** An **If** node filters articles published in the last 24 hours to ensure freshness.

#### Step 3: Score and Select Best Article

* **Tool:** Python Script (score\_article.py) + Local LLM (Optional).
* **Logic:** This is the filtering brain. The script receives the RSS headlines and the Trending Keywords. It computes a semantic similarity score.
* **Implementation:**
  + *Simple:* Jaccard similarity between trend keywords and article titles.
  + *Advanced:* Use a local embedding model (like all-MiniLM-L6-v2) to vectorise the titles and trends, calculating cosine similarity. This ensures "AI" (trend) matches "Artificial Intelligence" (article).
* **Output:** The JSON object of the single "Winning Article" is returned to n8n.

#### Step 4: Generate Learning Script

* **Tool:** LLM API (OpenAI/Anthropic) or Local LLM (Ollama).
* **Constraint:** For highest quality English instruction, a hosted model (GPT-4o) is recommended over local models, as subtle nuances in teaching explanations are critical. However, for a fully local stack, Llama-3-8B running on Ollama is sufficient.
* **Prompt Engineering:** The prompt must be strict.
  + *Role:* "You are an expert ESL teacher producing a podcast script."
  + *Structure:* "Create a dialogue between Host A (Teacher) and Host B (Student). Include 3 vocabulary definitions."
  + *Format:* "Return ONLY valid JSON with keys: dialogue (array of objects with speaker and text) and vocabulary (array)."
* **Recovery:** If the LLM returns markdown (json... ), a cleanup node in n8n must strip the backticks before parsing.14

### 6.2 Phase II: Synthesis and Production (Steps 5-7)

**Goal:** Convert text into broadcast-ready media.

#### Step 5: Generate Audio (Qwen3-TTS Locally)

* **Tool:** Python Script (generate\_audio.py) leveraging qwen-tts.
* **M5 Optimization:** The script must explicitly target the MPS device:  
  Python  
  device = "mps" if torch.backends.mps.is\_available() else "cpu"  
  model.to(device)  
    
  This line alone can reduce generation time from 10 minutes to 45 seconds on the M5.1
* **Process:** The script iterates through the dialogue JSON array. It concatenates the audio segments into a single .wav file. It also generates a separate "Vocabulary" segment.
* **Storage:** The raw file is saved to ~/OviPipeline/temp/raw\_audio.wav.

#### Step 6: Post-Process Audio (FFmpeg)

* **Tool:** FFmpeg (via n8n **Execute Command**).
* **The Problem:** Raw TTS output is often flat and lacks dynamic range.
* **The Solution:** Apply a compander (compressor-expander) and normalization filter to achieve broadcast standards (EBU R128).
* **Command:**  
  Bash  
  ffmpeg -y -i raw\_audio.wav -filter\_complex "compand=attacks=0:points=-80/-900|-45/-15|-27/-9|0/-7|20/-7:gain=5, loudnorm=I=-16:TP=-1.5:LRA=11" mastered.wav  
    
  This chain boosts quiet sections and caps loud peaks, ensuring the student can hear clearly.16

#### Step 7: Generate Video (The Divergence)

The pipeline offers two paths depending on the desired visual complexity.

**Path A: The Static Waveform (FFmpeg)**

For simple daily updates, a static image with the podcast audio is sufficient and extremely robust.

* **Command:**  
  Bash  
  ffmpeg -loop 1 -i assets/bg.jpg -i mastered.wav -c:v libx264 -tune stillimage -c:a aac -b:a 192k -shortest output.mp4  
    
  This creates a video file that is exactly as long as the audio.18

**Path B: The Professional Edit (DaVinci Resolve Studio)**

For high-value production with dynamic text overlays of the vocabulary words.

* **Constraint:** This **requires** DaVinci Resolve Studio ($295). The Free version purposefully disables the external API access required for automation.20
* **Mechanism:** The Python script (render\_resolve.py) imports the DaVinciResolveScript module.
* **Workflow:**
  1. Initialize the Resolve object: resolve = dvr\_script.scriptapp("Resolve").6
  2. Create a new Timeline.
  3. Import mastered.wav and the background video asset.
  4. **Dynamic Text:** The script iterates through the vocabulary list (generated in Step 4). It creates "Text+" titles on the timeline at the timestamps where those words are spoken.
  5. Trigger Render: The script sets the render settings (YouTube 1080p) and executes project.StartRendering().
  6. Wait Loop: The script polls project.GetRenderJobStatus() until completion.22

### 6.3 Phase III: Distribution and Notification (Steps 8-12)

**Goal:** Publish the content and alert the user.

#### Step 8: Upload to YouTube

* **Tool:** n8n **Google YouTube** Node.
* **Auth:** Requires OAuth2 setup in Google Cloud Console.
* **Metadata:** The "Title" comes from the Article Headline (Step 3). The "Description" includes the generated Script (Step 4) and Vocabulary list.
* **Privacy:** Set to Unlisted initially for manual review, or Public for full automation.

#### Step 9: Generate PDF Learning Guide

* **Tool:** Python Script (create\_pdf.py) using ReportLab.
* **Content:** Takes the JSON vocabulary list and formats it into a clean, branded PDF worksheet. This is a high-value asset for "Ovi English School" students.

#### Step 10: Upload Assets to Cloudflare R2

* **Tool:** n8n **S3** Node.
* **Configuration:** Cloudflare R2 is S3-compatible, but n8n's S3 node can be finicky with it.
  + **Endpoint:** https://<ACCOUNT\_ID>.r2.cloudflarestorage.com.
  + **Region:** Must be set to auto or us-east-1 (even though R2 is global, the SDK often expects a value).
  + **Force Path Style:** Enable this option if connection fails.
* **Output:** The node returns the Public URL of the uploaded MP4 and PDF.23

#### Step 11: Update RSS Feed

* **Tool:** Python Script (update\_rss.py).
* **Logic:**
  1. Download current feed.xml from R2.
  2. Parse XML, append new <item> with the link to the MP4 on R2 and the PDF.
  3. Re-upload feed.xml to R2.
  4. Invalidate Cloudflare Cache (optional via API) to ensure listeners get the new episode immediately.

#### Step 12: Send Notifications

* **Tool:** n8n **Slack** Node.
* **Payload:** "🚀 Episode Ready! Title: {{Title}}. YouTube: {{Link}}. PDF: {{Link}}."
* **Error Channel:** If any previous step failed, a different message is sent to the #alerts channel.

## 7. Reliability Engineering and Error Handling

A "set and forget" system requires robust failure management.

### 7.1 The "Circuit Breaker" Pattern

Not every trend yields a good lesson. If the "Score" in Step 3 is below a certain threshold (e.g., 0.6), the pipeline should gracefully terminate rather than producing low-quality content.

* **Implementation:** An **If** node follows the scoring script.
  + True (Score > 0.6): Proceed to Script Generation.
  + False (Score < 0.6): Route to a Slack node: "Skipped: No high-quality trends found today." -> End Workflow.

### 7.2 Retry Strategies with Exponential Backoff

External APIs (Google Trends, RSS Feeds, YouTube) are prone to transient network failures.

* **Configuration:** In n8n, every HTTP Request and Execute Command node has a "Retry on Fail" setting.
* **Best Practice:** Set "Max Retries" to 3 and "Wait Between Retries" to 5000ms. This resolves 90% of connectivity glitches.25

### 7.3 Global Error Workflow

A dedicated "Error Workflow" serves as the catch-all for catastrophic failures (e.g., Disk Full, Python Crash).

1. Create a new workflow named "Error Handler."
2. Use the **Error Trigger** node (triggers on any workflow failure).
3. Connect to a Slack node sending {{ $execution.error.message }} and the workflow URL.
4. Link this workflow in the "Settings" tab of the main Ovi Pipeline workflow.

## 8. Monitoring and Scheduling Setup

### 8.1 The "Daily Daily" Schedule

To satisfy the "6 AM JST" requirement, the scheduling relies on the interaction between the system clock and the workflow trigger.

1. **System Time:** The startup script (Section 4.2) sets GENERIC\_TIMEZONE="Asia/Tokyo".
2. **Workflow Trigger:** The **Schedule Trigger** node is configured for Every Day at 06:00.

### 8.2 Overcoming Sleep (The caffeinate Wrapper)

The most common failure mode for local Mac automation is the computer being asleep at the trigger time. The pmset command wakes the computer at 05:55. However, if the machine is idle for a few minutes, it might sleep again before the heavy rendering finishes.

**Integration Strategy:**

The first node in the n8n workflow after the trigger should be an **Execute Command** node running:

Bash

caffeinate -t 3600 &

This command creates a power assertion preventing sleep for 3600 seconds (1 hour), covering the duration of the pipeline execution. The & allows n8n to continue to the next node immediately without waiting for the hour to pass.12

## 9. Alternatives Comparison

| **Alternative** | **Pros** | **Cons** | **Verdict** |
| --- | --- | --- | --- |
| **Cloud-Native (AWS/Lambda)** | High availability, scalable. | Expensive for GPU/LLM tasks. Complex state management. | **Rejected** (Cost/Privacy). |
| **Hybrid (Cloud n8n + Local Worker)** | Reliable orchestration. | Requires tunneling (ngrok/VPN) to control Mac. Complex networking. | **Rejected** (Complexity). |
| **Fully Local (AppleScript/Automator)** | Native integration. | Poor error handling, no visual logic flow, hard to debug. | **Rejected** (Fragility). |
| **Local n8n (Current Proposal)** | Balance of power, cost, and control. Direct hardware access. | Requires managing local env. Single point of failure (the Mac). | **Recommended.** |

## 10. Implementation Artifacts

### 10.1 Sample Workflow JSON (Skeleton)

This JSON provides the structural skeleton for the Ovi Pipeline. It includes the logic for scheduling, sleep prevention, ingestion, and conditional processing.

JSON

{  
 "nodes": [  
 {  
 "parameters": {  
 "rule": {  
 "interval": [  
 {  
 "field": "cronExpression",  
 "expression": "0 6 \* \* \*"  
 }  
 ]  
 }  
 },  
 "name": "Daily Schedule (6AM)",  
 "type": "n8n-nodes-base.scheduleTrigger",  
 "typeVersion": 1,  
 "position":   
 },  
 {  
 "parameters": {  
 "command": "caffeinate -t 3600 &"  
 },  
 "name": "Prevent Sleep",  
 "type": "n8n-nodes-base.executeCommand",  
 "typeVersion": 1,  
 "position":   
 },  
 {  
 "parameters": {  
 "command": "/Users/ovi/OviPipeline/python\_workers/venv/bin/python /Users/ovi/OviPipeline/scripts/fetch\_trends.py"  
 },  
 "name": "Fetch Trends",  
 "type": "n8n-nodes-base.executeCommand",  
 "typeVersion": 1,  
 "position":   
 },  
 {  
 "parameters": {  
 "url": "https://feeds.feedburner.com/TechCrunch/"  
 },  
 "name": "Fetch RSS",  
 "type": "n8n-nodes-base.rssFeedRead",  
 "typeVersion": 1,  
 "position":   
 },  
 {  
 "parameters": {  
 "mode": "json",  
 "json": "={{ JSON.parse($json.stdout) }}"  
 },  
 "name": "Parse Trends",  
 "type": "n8n-nodes-base.set",  
 "typeVersion": 1,  
 "position":   
 },  
 {  
 "parameters": {  
 "conditions": {  
 "number": [  
 {  
 "value1": "={{ $json.score }}",  
 "operation": "larger",  
 "value2": 0.6  
 }  
 ]  
 }  
 },  
 "name": "Quality Filter",  
 "type": "n8n-nodes-base.if",  
 "typeVersion": 1,  
 "position":   
 }  
 ],  
 "connections": {  
 "Daily Schedule (6AM)": {  
 "main":  
 },  
 "Prevent Sleep": {  
 "main":  
 },  
 "Fetch Trends": {  
 "main":  
 }  
 }  
}

### 10.2 Best Practices Checklist for Reliable Automation

* **[ ] Path Consistency:** Always use absolute paths in Python scripts (/Users/username/...). Relative paths will fail because n8n's working directory varies.
* **[ ] Environment Hygiene:** Do not put API keys in Python scripts. Use os.environ.get('API\_KEY') and pass them via the "Environment Variables" section of the n8n Execute Command node.
* **[ ] Output Sanitization:** Ensure Python scripts intended for n8n consumption print *nothing* to stdout except the final JSON result. Use stderr for logging.
* **[ ] Disk Maintenance:** The M5 SSD is fast but not infinite. Implement a cleanup step that deletes the /temp folder contents after a successful upload to R2.
* **[ ] Version Pinning:** When installing Python libraries, pin the versions in a requirements.txt file (e.g., torch==2.1.2) to prevent an auto-update from breaking the pipeline weeks later.

## 11. Conclusion

The "Ovi English School" pipeline represents a sophisticated application of edge computing principles. By leveraging the specific hardware advantages of the Apple Silicon M5—specifically the Unified Memory for efficient data handling and the Neural Engine for accelerated inference—and combining them with the orchestration capabilities of n8n, it is possible to build a broadcast-quality automated media facility on a single laptop.

The architecture defined in this report prioritizes **native execution** over containerization to solve the unique challenges of local hardware access and application control. While this approach requires more initial configuration of the host environment, it yields a system that is significantly more performant, cost-effective, and capable of handling the multi-modal demands of modern content creation. The implementation of strict error handling patterns and power management strategies ensures that the system will reliably meet its 6:00 AM daily deadline, transforming the MacBook Pro M5 from a passive tool into an active, autonomous production partner.

#### Works cited

1. Qwen3-TTS: The Complete 2026 Guide to Open-Source Voice Cloning and AI Speech Generation | by cheng zhang - Medium, accessed February 1, 2026, <https://medium.com/@zh.milo/qwen3-tts-the-complete-2026-guide-to-open-source-voice-cloning-and-ai-speech-generation-1a2efca05cd6>
2. Self Hosted AI Starter Kit - Questions - n8n Community, accessed February 1, 2026, <https://community.n8n.io/t/self-hosted-ai-starter-kit/56525>
3. FFmpeg + n8n + Docker on Render — How I Solved a Real Client Problem in Under 24 Hours | by Damipe Tiwo - Medium, accessed February 1, 2026, <https://medium.com/@damipetiwo/ffmpeg-n8n-docker-on-render-how-i-solved-a-real-client-problem-in-under-24-hours-a91bc6fb9c93>
4. Install FFMPEG with N8N on docker for video editing - 27 second guide - Reddit, accessed February 1, 2026, <https://www.reddit.com/r/n8n/comments/1lov13o/install_ffmpeg_with_n8n_on_docker_for_video/>
5. Unofficial DaVinci Resolve Scripting Documentation | DaVinciResolve-API-Docs, accessed February 1, 2026, <https://deric.github.io/DaVinciResolve-API-Docs/>
6. diop/davinci-resolve-api: Davinci Resolve Python Api Documentation - GitHub, accessed February 1, 2026, <https://github.com/diop/davinci-resolve-api>
7. No Docker - Self Hosted AI Starter Kit Help? - Questions - n8n Community, accessed February 1, 2026, <https://community.n8n.io/t/no-docker-self-hosted-ai-starter-kit-help/69252>
8. Alibaba Cloud Model Studio:Real-time speech synthesis - Qwen, accessed February 1, 2026, <https://www.alibabacloud.com/help/en/model-studio/qwen-tts-realtime>
9. Qwen3-TTS is an open-source series of TTS models developed by the Qwen team at Alibaba Cloud, supporting stable, expressive, and streaming speech generation, free-form voice design, and vivid voice cloning. - GitHub, accessed February 1, 2026, <https://github.com/QwenLM/Qwen3-TTS>
10. Schedule your Mac to turn on or off in Terminal - Apple Support, accessed February 1, 2026, <https://support.apple.com/guide/mac-help/schedule-your-mac-to-turn-on-or-off-mchl40376151/mac>
11. Scheduling a Shut Down on macOS - DssW, accessed February 1, 2026, <https://www.dssw.co.uk/blog/2024-05-02-scheduled-shut-down-macos/>
12. Quick Tip: caffeinate your Terminal - BrettTerpstra.com, accessed February 1, 2026, <https://brettterpstra.com/2014/02/20/quick-tip-caffeinate-your-terminal/>
13. Terminal Hacks - Caffeinate your Mac - HEY World, accessed February 1, 2026, <https://world.hey.com/james.pearson/terminal-hacks-caffeinate-your-mac-6d897d7a>
14. Converting string to json object in function node - Questions - n8n Community, accessed February 1, 2026, <https://community.n8n.io/t/converting-string-to-json-object-in-function-node/338>
15. Parsing Data from Execute Command node / stdout - Questions - n8n Community, accessed February 1, 2026, <https://community.n8n.io/t/parsing-data-from-execute-command-node-stdout/11035>
16. Are you trying to Automate FFmpeg In N8N Docker Instance? - Reddit, accessed February 1, 2026, <https://www.reddit.com/r/automation/comments/1p07pid/are_you_trying_to_automate_ffmpeg_in_n8n_docker/>
17. Creating a video from an image and multiple audio files (uploading music albums to youtube) : r/ffmpeg - Reddit, accessed February 1, 2026, <https://www.reddit.com/r/ffmpeg/comments/19es0sv/creating_a_video_from_an_image_and_multiple_audio/>
18. Adding an image to an audio file : r/ffmpeg - Reddit, accessed February 1, 2026, <https://www.reddit.com/r/ffmpeg/comments/1byaxwx/adding_an_image_to_an_audio_file/>
19. Create video with an image and audio file using ffmpeg - Ask Ubuntu, accessed February 1, 2026, <https://askubuntu.com/questions/1250544/create-video-with-an-image-and-audio-file-using-ffmpeg>
20. View topic - Resolve - no python in free version? - Blackmagic Forum, accessed February 1, 2026, <https://forum.blackmagicdesign.com/viewtopic.php?f=21&t=77764>
21. Resolve free version limitations : r/colorists - Reddit, accessed February 1, 2026, <https://www.reddit.com/r/colorists/comments/1d5v85c/resolve_free_version_limitations/>
22. Creating Scripts for DaVinci Resolve - Blackmagic Forum • View topic, accessed February 1, 2026, <https://forum.blackmagicdesign.com/viewtopic.php?t=175315>
23. External storage for binary data - n8n Docs, accessed February 1, 2026, <https://docs.n8n.io/hosting/scaling/external-storage/>
24. jezweb/n8n-nodes-cloudflare-r2 - GitHub, accessed February 1, 2026, <https://github.com/jezweb/n8n-nodes-cloudflare-r2>
25. Advanced retry and delay logic | n8n workflow template, accessed February 1, 2026, <https://n8n.io/workflows/5447-advanced-retry-and-delay-logic/>
26. Tried to collate error handling, graceful retries and security for production ready n8n - Reddit, accessed February 1, 2026, <https://www.reddit.com/r/n8n/comments/1pao6wa/tried_to_collate_error_handling_graceful_retries/>