⛩️ ANIME CHANNEL PIPELINE

Complete System Guide

13 Skills · OpenClaw · xSkill · Mem0 · Auto-Scheduler

What Is This?

This is a fully automated AI video production pipeline built for a "Real World as Anime" YouTube channel. You give it a news story. It produces a complete, ready-to-post anime-style video — script, visuals, voiceover, music, and all — with minimal input from you.

The whole system runs inside OpenClaw on your machine. You control it from your phone or PC through chat. A dashboard shows you what's happening in real time.

The pipeline has 13 skills (agents), each responsible for one job:

SKILL	NAME	JOB
🗺️	**Director**	Orchestrates everything. The brain of the pipeline.
🔍	**Scout**	Finds story candidates from news and web every morning.
📚	**Researcher**	Deep-dives each story, adds anime framing and cultural context.
✍️	**Scriptwriter**	Writes the full narrated script with scene descriptions.
🎨	**Visual**	Generates all images and video clips via xSkill API.
🎙️	**Audio**	Creates voiceover (TTS) and background music.
🎬	**Assembly**	Stitches everything into a final video using Remotion.
📤	**Publisher**	Uploads and posts to YouTube, TikTok, Instagram, X.
🛡️	**Resilience**	Error handling, retries, and fallback rules for every agent.
🔧	**Tools Registry**	Maps capabilities to providers — swap APIs without rewriting skills.
🧠	**Memory**	Long-term channel memory via Mem0. Gets smarter over time.
⏰	**Scheduler**	Cron jobs — Scout runs at 6am, analytics auto-check, etc.
📊	**Dashboard**	Web UI showing pipeline status, scenes, logs, credits.

How a Video Gets Made — Step by Step

Every video follows the same 8-stage pipeline. You are only asked to make decisions at specific approval gates — everything else runs automatically.

The 8 Stages

#	STAGE	WHAT HAPPENS	YOUR INPUT
1	🔍 **Scout**	Finds 3 story candidates from news, social media, Reddit. Scores each on anime potential, emotional stakes, visual richness.	**Pick 1 of 3**
2	📚 **Research**	Deep research on your chosen story. Adds cultural context, anime framing, tone. Checks Mem0 for past preferences.	None — auto
3	✍️ **Script**	Writes the full narrated script: hook, world setup, conflict, resolution, CTA. 8 scenes, ~5 minutes.	**Approve / edit**
4	🎨 **Style Anchor**	Generates scene 1 as a style test image. You confirm the look before the full batch runs.	**Approve the look**
5	⚡ **Generate**	All scenes generated in batches of 3. Images are instant. Videos take a few minutes to 20+ minutes each depending on provider and queue — they poll in the background automatically.	None — auto
6	🎙️ **Audio**	Voiceover recorded via MiniMax TTS in your chosen voice. Background music generated. Synced to scene durations.	None — auto
7	🎬 **Assemble**	Everything stitched together with Remotion. Captions, transitions, thumbnail options generated.	**Approve preview**
8	📤 **Publish**	Posted to your configured platforms with optimised titles, descriptions, tags, and thumbnails.	**Confirm publish**

⏱️ Total time per video: varies. Images generate instantly. Video scenes can each take a few minutes to 20+ minutes depending on provider and queue load. Your active time: under 10 minutes.

Your 4 Approval Gates

You make exactly 4 decisions per video. Everything else the pipeline handles automatically.

✋ **Gate 1 — Story Selection**
After Scout runs, you get 3 ranked candidates in chat. Reply with 1, 2, or 3. Done.

✋ **Gate 2 — Script Approval**
The full script arrives in chat. Read it, approve it as-is, or tell the pipeline what to change. Once approved, generation starts automatically.

✋ **Gate 3 — Style Anchor**
A single test image arrives. This is what your entire video will look like visually. Say yes or describe what to change. This is your only chance to adjust the visual direction before all scenes generate.

✋ **Gate 4 — Preview & Publish**
The assembled video preview is ready. Watch it, approve it, and the pipeline publishes to all your platforms automatically.

💡 **Between gates — nothing to do**
The pipeline runs in the background. Check the dashboard at any time to see progress. You don't need to stay in the chat window.

Technology Stack

Here is everything the pipeline uses and why.

Frameworks

TOOL	WHAT IT IS	ROLE IN PIPELINE
OpenClaw	Agent framework	Runs all 13 skills. Your interface for approvals. Handles cron scheduling.
xSkill.ai	AI API aggregator	Single API key for 50+ models. Images via Flux, videos via Seedance 2.0, TTS via MiniMax.
MiniMax	AI provider	Backup TTS provider. Also accessible via xSkill.
Mem0	Long-term memory	Stores what you approve, reject, and prefer. Pipeline gets smarter every run.
Remotion	Video assembly	Programmatic video editing. Stitches scenes, audio, captions into final MP4.
Node.js server	Dashboard backend	Serves the dashboard HTML + workspace files. Runs on port 7777.
Cloudflare tunnel	Mobile access	Exposes dashboard to your phone without port forwarding.

The xSkill API — How Video Generation Works

This is the most important thing to understand about why the pipeline works the way it does. xSkill has two completely different modes.

Images — Instant (synchronous)

When the Visual agent generates an image, it calls sync_generate_image. The result comes back immediately — URL in the response, download it, done. No waiting, no polling.

Videos — Asynchronous (time varies)

Video generation is completely different. This is a two-step process and getting it wrong is what caused repeated generations and errors in the past.

Generation time is unpredictable — it can take anywhere from a few minutes to 20+ minutes depending on the provider, the model, queue load, video length, and whether reference media is attached. The pipeline never assumes a fixed time. It just keeps polling until the job completes.

Step 1 — Submit the job:

```
submit_task → returns { task_id: "task_xxx" } immediately
```

The task_id is written to state.json. The agent session ENDS here. It does NOT wait.

Step 2 — Poll separately (every 10 seconds):

```
get_task(task_id) → status: "pending" | "processing" | "completed" | "failed"
```

A separate polling agent checks every 10 seconds. When status = completed, it downloads the video, verifies the file size is over 100KB, and marks the scene done.

> ⚠️ **Why this matters**
> The old approach would submit a video job, immediately try to get the result, find nothing there, and panic — regenerating the same scene over and over. The correct approach: submit → save task_id → exit → poll later. Never call get_task in the same session as submit_task.

Character consistency — the @image_file_1 pattern:

To keep characters looking the same across scenes, Seedance 2.0 uses image references — not text descriptions. The Visual agent uploads the scene 1 image to xSkill, then passes it as a reference in every subsequent video prompt.

```
prompt: "@image_file_1 [scene description], same character"
media_files: ["https://xskill-stable-url-of-scene-1.png"]
```

Without this pattern, Seedance generates a different-looking character every time.

state.json — The Pipeline's Brain

Every agent reads from and writes to a single file: workspace/in-progress/state.json. This is how agents communicate across sessions without needing to stay in memory.

Key things stored in state.json:

- Current pipeline status (scouting / scripting / generating / assembling / etc.)
- The approved story, research brief, and script
- Style anchor — the approved visual profile and reference image URL
- Character cards — image reference URLs for each recurring character
- generation_jobs — one entry per scene, with task_id, status, file_path
- pipeline_log — every agent action timestamped
- degradations — any scenes that fell back to static image instead of video
- phase_completion — flags for when each phase finishes

💡 **Why this design?**

AI agent sessions have limited context windows and don't persist between calls. By writing everything to state.json, each agent can pick up exactly where the last one left off — even if OpenClaw restarts or a session times out.

Tools Registry — Swapping Providers

The tools registry (anime-channel-tools/SKILL.md) is a single file that maps capabilities to providers. No agent hardcodes a provider name.

Instead of writing "call xSkill Flux" in a skill, agents say "I need TEXT_TO_IMAGE" and look up the current provider in the registry.

If you want to switch from xSkill Flux to a different image model, you change one line in the registry. All 13 skills adapt automatically — nothing else needs updating.

CAPABILITY	PRIMARY	FALLBACK	DEGRADE TO
TEXT_TO_IMAGE	xskill: flux-2-flash	seedream-4.5	—
TEXT_TO_VIDEO	xskill: seedance_2.0_fast	wan-2.6, sora-2	static image
TEXT_TO_SPEECH	xskill: minimax-tts	minimax direct	—
MUSIC_GEN	xskill: music-gen	—	royalty-free track

Memory — How the Pipeline Gets Smarter

The Memory skill connects the pipeline to Mem0. After every run, the pipeline learns from your decisions. Over time it stops making the same mistakes and starts anticipating your preferences.

What It Remembers

- Stories you picked vs. skipped — builds a taste profile
- Visual styles you approved without changes — becomes the default starting point
- Scripts you edited — learns to avoid the same issues next time
- How each video performed (views, retention, CTR) — routes toward what works
- Hard preferences: "never do X" — permanently filters those out

The Learning Loop

Every Monday the Director reads memory and sends you a plain-English summary: "This month I learned you prefer dark historical stories over feel-good ones. I've adjusted the Scout's scoring accordingly." Then it applies those learnings automatically to future runs.

💡 **First run note**

Memory is empty on the first run — that's normal. It builds over 3–5 videos and becomes genuinely useful around week 2.

Scheduler — What Runs Automatically

The Scheduler sets up cron jobs so the pipeline is working before you even wake up.

JOB	WHEN	WHAT IT DOES
Morning Scout	**6:00 AM daily**	Finds + researches 3 candidates. Ready before you wake up.
Analytics Check	**9:00 AM daily**	Checks performance of videos posted 24–72h ago. Writes to Mem0.
Weekly Report	**Monday 8:00 AM**	Full week analytics + intelligence summary of what it learned.
Credit Check	**Sunday 8:00 PM**	Checks xSkill balance. Warns if running low before the week starts.
Disk Cleanup	**Saturday 2:00 AM**	Archives old published runs if disk is over 80% full.

Dashboard — Monitoring Without Asking

The dashboard is a web page you open in your browser (or phone) to see what the pipeline is doing without having to type anything in OpenClaw.

What It Shows

- Current pipeline stage — which of the 8 stages is active right now
- Scene grid — all 8 scenes with status: waiting / submitted / done / failed / degraded
- Live log — every agent action, timestamped, newest first
- Morning queue — today's 3 story candidates with scores
- Published videos — your last 8 videos with views and retention
- Credit bar — xSkill balance with estimated days remaining
- Degradation alerts — any scenes that fell back to static image

How to Access It

- Local: http://localhost:7777
- Mobile: Cloudflare tunnel URL (changes each restart — bookmark it)
- Auto-refreshes every 15 seconds — no manual refresh needed

📱 **Tip for mobile**

Add the Cloudflare tunnel URL to your phone's home screen as a web app. During a pipeline run you can check progress without opening OpenClaw at all.

Installation — What to Install and In What Order

All 13 skills are packaged as .skill files. Install them in this order — some skills depend on others being present first.

Order matters for the first 4:

1. **tools-registry.skill** *— Everything else reads this first*

2. **resilience.skill** *— Error handling rules used by every agent*

3. **memory.skill** *— Mem0 integration — must be ready before Director runs*

4. **director.skill** *— Reads tools-registry, resilience, and memory on every run*

5–12. **All other skills** *— Any order: scout, researcher, scriptwriter, visual, audio, assembly, publisher, scheduler*

13. **dashboard.skill** *— Install last — contains both the SKILL.md and the dashboard HTML*

After installing all skills:

- Connect xSkill MCP to OpenClaw
- Connect Mem0 to OpenClaw
- Set your Cloudflare tunnel up (or use ngrok as alternative)
- Run the dashboard server: node workspace/dashboard-server.js
- Trigger the Director manually once to do a dry run with no story — just to verify all skills load correctly

🚀 **First real test**

Don't try to make a full video on run 1. Ask the Director to just run the Scout stage and stop. Verify you get 3 story candidates. Then approve one and run just the Research stage. Build up gradually — this is how you find what's broken before it wastes credits on generation.

Workspace Folder Structure

All pipeline files live in a workspace/ folder. Here is the layout:

```
workspace/
  in-progress/
    state.json              ← The active pipeline state
    assets/
      images/               ← Generated scene images
      videos/               ← Generated scene videos
      audio/                ← Voiceover and music files
  published/
    [run-id]/               ← One folder per completed video
      state.json            ← Final state snapshot
      final-video.mp4       ← The published video
  queue/
    morning_YYYY-MM-DD.json ← Morning Scout candidates
  analytics/
    credits.json            ← xSkill credit balance
    [video-id].json         ← Per-video analytics
  scheduler_config.json     ← Pause/resume scheduler
  dashboard-server.js       ← Dashboard Node.js server
```

Troubleshooting — Common Issues

Same image generated multiple times

Cause: The Visual agent is submitting a video job and immediately calling get_task before the video is ready — it gets an empty result and retries the whole thing.

Fix: Check that the Visual skill is correctly installed. It should submit video jobs, write task_id to state.json, and exit — not poll in the same session.

Characters look different in each scene

Cause: The @image_file_1 reference syntax is not being used. The Visual agent is describing the character in text instead of passing the scene 1 image as a reference.

Fix: Verify the style anchor step ran correctly and that style_anchor.xskill_reference_url is populated in state.json before batch generation starts.

Pipeline stuck — nothing happening

Cause: Usually an approval gate waiting for your response, or a failed job with no retry.

Fix: Check the dashboard log for the last entry. If it says "awaiting owner" — just reply in OpenClaw chat. If it shows a failed job — check state.json degradations[] for the error message.

Dashboard not loading

Cause: The Node.js server isn't running, or the Cloudflare tunnel expired.

Fix: Run node workspace/dashboard-server.js again. If accessing from mobile, get the new tunnel URL — it changes every restart.

xSkill credits depleted mid-run

Cause: A batch of video generations ran without a credit check first.

Fix: The Visual skill checks get_balance before every batch. If you bypassed this or the check failed silently, top up xSkill credits and resume. The pipeline will not regenerate completed scenes — it picks up from where it left off.