There is a new AI voice recorder on the market. What is it?

The market for AI voice recorders has exploded recently, moving past simple apps and evolving into specialized "Dual-Engine" hardware. These physical devices pack advanced microphone arrays with back-end Large Language Models (LLMs) like GPT-4o, GPT-5, or Claude to handle transcription, speaker separation (diarization), and structured summaries instantly.

Skywork

Depending on which specific launch or style you might have spotted, several notable new entries dominate the landscape right now:

1. The Coin-Sized Wearable: Anker Soundcore Work

Released in **March 2026**, Anker heavily disrupted the form factor trend with this device.

note

- **The Hardware:** It features a tiny, ultra-lightweight microphone unit measuring just 23.2 mm in diameter and weighing only 10 grams. It comes with a pocketable charging case that extends total battery life to 32 hours.

note

- **The AI Capabilities:** It supports transcription in over 150 languages with high accuracy, filters out intense background noise, and includes automated multi-speaker labeling. Because it's an MFi-certified device, it also integrates natively with Apple's "Find My" network.

Android Police

2. The Premium Executive Upgrade: Plaud Note Pro & NotePin S

Plaud essentially pioneered the ultra-thin MagSafe form factor, and their latest lineups continue to lead the corporate space.

Skywork

- **The Hardware:** The Note Pro is a credit-card-thin device that snaps to the back of a phone. It uses a **Vibration Conduction Sensor (VCS)** to physically pick up chassis vibrations, allowing it to natively record cell phone calls while entirely bypassing mobile operating system restrictions.

Skywork+ 1

- **The AI Capabilities:** It allows users to route audio through their choice of models (including GPT and Claude) to generate highly specialized templates—like medical SOAP notes, legal case summaries, or structured corporate mind maps.

3. The Budget & Privacy Challenger: UMEVO Note Plus

Targeted squarely at users who are tired of recurring monthly software fees, this is an all-in-one standalone option.

- **The Hardware:** A sleek, screenless pocket recorder packed with 64GB of local storage (good for roughly 400 hours of uncompressed audio) and a massive 40-hour continuous recording battery.

- **The AI Capabilities:** To fight "subscription fatigue," it bundles a full year of unlimited AI transcription with the hardware purchase and defaults to a generous free monthly tier afterward.

4. The All-in-One Multi-Media Recorder: BOYA Notra

A rugged, field-ready entry focused heavily on cross-platform flexibility.

- **The Hardware:** Features a "Triple-Mode" physical switch that natively transitions between earphone recording, direct smartphone call recording, and ambient boardroom recording.
- **The AI Capabilities:** Handles 140+ languages. For security-conscious professionals or investigative journalists, it includes a unique on-device **Voice Mask** feature that scrambles or disguises a speaker's vocal characteristics before the file is sent over AWS cloud infrastructure for transcription.

What to Look For If You're Buying

Soundcore

If you're trying to figure out which one fits your routine, look closely at how they handle **Total Cost of Ownership (TCO)**. While the hardware usually costs between $100 and $200, most brands (like Plaud) lock high-volume monthly transcription limits behind a $70–$100/year subscription, whereas brands like UMEVO or iZYREC lean more heavily on local storage and free cloud tiers.

Is there AI software for MX25 Linux that will do the same thing using local ollama

Yes, absolutely. Because MX25 is built on Debian 12 (Bookworm) stable packages, you can replicate this exact "Dual-Engine" hardware behavior locally.

The secret to matching those physical gadgets on Linux is decoupling the tasks: you use **Faster-Whisper** (C++ optimized transcription) to handle the speech-to-text layer, and then pipe that text to **Ollama** via its local API to generate summaries, action items, or mind maps.

The best tools and approaches to achieve this locally on MX25 include:

1. Dedicated Open-Source Applications

OpenWhispr (Desktop App)

- **What it is:** A native, privacy-first open-source desktop application available for Linux that handles local speech-to-text and AI summaries.

  YouTube
- **How it works:** It allows you to run Whisper models completely offline on your own hardware (supporting models from `Whisper Tiny` up to `Whisper Turbo`). It includes built-in hooks to connect directly to local LLM providers or structured text formats, acting like a software clone of a Plaud or UMEVO recorder.

Buzz (Desktop App)

- **What it is:** One of the most popular graphic interfaces for offline transcription on Linux, built on top of OpenAI's Whisper.

- **How it works:** You import any audio format (MP3, WAV, FLAC, etc.), and it transcribes it or translates it locally, spitting out `.txt` or time-stamped `.srt` caption files. You then feed that text directly into your Ollama pipeline.
- **MX25 Install Note:** You can grab Buzz easily as a Flatpak, or install it inside a Python virtual environment via `pip`.

`ollama-transcriber` (CLI / Script automation)

If you prefer terminal-driven toolchains, the open-source **`chumphrey-cmd/ollama-transcriber`** project on GitHub is explicitly built for this workflow. It wraps the entire pipeline into a single command:
GitHub

1. It takes an audio file input.
   GitHub
2. It calls a local Python Whisper implementation to transcribe it.
3. It hits your local `ollama` instance (via `config.yaml` where you specify models like `llama3.2` or `mistral`) and feeds it a custom `summary_prompt`.

2. Rolling Your Own Pipeline (The Unix Way)

If you don't want a heavy GUI and want to feed recordings directly from a pocket recorder or phone mic into your system, you can build a flawless, lightning-fast bash pipeline using `faster-whisper` and `curl`.

Step 1: Install System Dependencies

MX25 has all of this native in the repos:

Bash
```
sudo apt update && sudo apt install ffmpeg python3-pip python3-venv -y
```

Step 2: Set up Faster-Whisper

Standard OpenAI Whisper can be heavy, but `faster-whisper` uses CTranslate2, making it up to 4x faster and heavily reducing VRAM/RAM overhead.

Bash
```
python3 -m venv whisper_env
source whisper_env/bin/activate
pip install faster-whisper
```

Step 3: The Automation Script

You can throw this simple Python snippet into a local script (e.g., `process_audio.py`). It transcribes the audio locally, then fires a structured request to your local Ollama instance:

Python
```
import requests
```

```
from faster_whisper import WhisperModel

# 1. Transcribe Local Audio File
model_size = "base"  # Options: tiny, base, small, medium, turbo
model = WhisperModel(model_size, device="cpu", compute_type="int8")
# Note: If running on an NVIDIA GPU, change to device="cuda",
compute_type="float16"

print("Transcribing audio file locally...")
segments, info = model.transcribe("meeting_recording.mp3", beam_size=5)
transcript = " ".join([segment.text for segment in segments])

# 2. Pipe to Local Ollama for Structured Summary
print("Generating AI summary via Ollama...")
ollama_url = "http://localhost:11434/api/generate"
payload = {
    "model": "llama3.2", # Or your preferred local model
    "prompt": f"Analyze the following meeting transcript. Provide a concise
bulleted summary and a clear checklist of action items:\n\n{transcript}",
    "stream": False
}

response = requests.post(ollama_url, json=payload)
print("\n--- AI SUMMARY REPORT ---")
print(response.json()['response'])
```

Hardware Tip for Linux Users

If you want the portability of the dedicated AI hardware mentioned before without their cloud software ecosystems, buy a standard, high-quality **"dumb" dictaphone / voice recorder** that mounts as a mass storage device over USB-C (like a standard flash drive).

When you plug it into your MX25 machine, a simple `udev` rule or short shell script can auto-detect the device, copy the `.wav` or `.mp3` files off the recorder, pass them to your local Whisper environment, trigger Ollama, and output a markdown summary file directly into your notes directory seamlessly.

Buy high-quality "dumb" dictaphone / voice recorder links

When choosing a standalone "dumb" dictaphone to pair with a local Linux transcription pipeline, the most critical features are a **native USB mass storage interface** (so it mounts as a standard directory under `/media/`) and **uncompressed PCM (WAV) or high-bitrate MP3 recording**. These options act as pure storage devices, requiring no proprietary software, cloud logins, or platform-specific sync drivers to retrieve your audio.

Key Recommendations

OM Digital Voice Recorder WS-883
$99.99·OM System
4.4★(35)

The OM System WS-883 features a slide-out, **built-in USB-A connector** that plugs straight into your computer like a thumb drive, making audio file ingestion completely seamless.

- **Key Specs:** 8GB internal memory, **microSD expansion** up to 32GB, dual 90° stereo directional mics, **110-hour battery life** (via 2x AAA NiMH rechargeable batteries), and a built-in vibration-damping kickstand.
  Genesis Technologies, Inc.
- **Audio Formats:** High-quality mono linear PCM (WAV) and standard stereo MP3.

om system

Sony ICD-UX570 Digital Voice Recorder
$186.66·Sony
4.8★(1.2k+)

The Sony ICD-UX570 is a sleek, pocket-sized recorder with an **integrated slide-out USB port** and a highly legible OLED display.

- **Key Specs:** 4GB internal storage, **microSD card slot**, highly sensitive S-Microphone array with automatic background noise reduction, and an **internal rechargeable battery** that supports quick charging.
  B&H
- **Audio Formats:** Linear PCM (WAV) and standard MP3.

Philips Voice Tracer Dvt2110 Digital Recorder
$124.99·Philips
4.3★(11)

The Philips VoiceTracer DVT2110 offers native **plug-and-play compatibility** for Linux and includes a crisp color display.

- **Key Specs:** 8GB internal storage, high-fidelity stereo dual-microphone array, a large **rechargeable Li-ion battery** delivering up to 36 hours of continuous use via USB-C.

- **Audio Formats:** Uncompressed 2-channel PCM (WAV) and stereo MP3.

Hardware Feature Comparison

If you are choosing between these models to build your automated transcription workflow, here is a breakdown of how they compare across core functional hardware metrics:

		
OM Digital Voice Recorder WS-883	**Sony ICD-UX570 Digital Voice Recorder**	**Philips V**
$99.99· OM System **4.4(35)**	**$186.66· Sony** **4.8(1k+)**	
USB Interface Integrated Slide-out USB-A	USB Interface Integrated Slide-out USB-A	USB Inter USB-C Po
Power Source 2x AAA Rechargeable NiMH	Power Source Internal Rechargeable Battery	Power Sou Internal Li
Storage Expansion microSD (up to 32GB)	Storage Expansion microSD slot available	Storage Ex Internal O
Audio Performance 90-degree True Stereo Mic system	Audio Performance High-sensitivity S-Microphone array	Audio Per 2-Mic Hig

Recommendation

- Choose the OM System WS-883 if you want **swappable AAA battery power** and maximum long-term fieldwork endurance without waiting on a charger.
- Choose the Sony ICD-UX570 if you prefer a **compact form factor** and a premium OLED interface for quick navigation.

If you'd like, I can write a custom `udev` rule and bash automation script that listens for your specific recorder's hardware ID, auto-mounts the storage volume, sweeps the files, and routes them straight

through your local `faster-whisper` and Ollama setup. Would you like me to put that script together?

give lowest cost links. I have Blutooth earbuds with mic

Since you already have a **Bluetooth earbud and microphone setup**, you do not need to buy an expensive standalone voice recorder at all. You can use your existing hardware to feed audio directly into your **MX25 Linux machine** for local processing.

If your Linux rig does not have a built-in Bluetooth card, the absolute lowest-cost approach is to add a cheap, Linux-native USB Bluetooth dongle.

Low-Cost Bluetooth USB Adapters

Xcellon Bluetooth 5.0 USB Adapter
$9.99·Xcellon
4.2★(14)

The Xcellon Bluetooth 5.0 USB Adapter is a tiny, budget-friendly dongle that provides solid wireless connectivity for Linux desktop systems.

- **Key Specs:** Bluetooth 5.0 Low Energy, ultra-compact nano design, and an integrated status LED indicator.
- **Range:** Up to 65 feet of wireless coverage.

Plugable USB Bluetooth Adapter
$24.00·Plugable Technologies
4.5★(53)

The Plugable USB Bluetooth Adapter is heavily favored in open-source communities for its out-of-the-box, **plug-and-play Linux kernel support** without manual firmware hacking.

- **Key Specs:** Bluetooth 5.0 dual-mode Classic + LE, backward-compatible with older hardware, and allows up to 7 concurrent device connections.
- **Range:** Up to 130 feet (40 meters) in clear conditions.

Hardware Feature Comparison

These budget-focused USB adapters give your Linux workstation immediate wireless access to your existing audio gear:

Xcellon Bluetooth 5.0 USB Adapter

$9.99· Xcellon
4.2(14)

Bluetooth Version	Bluetooth 5.0 LE
Linux Compatibility	Standard Plug & Play
Wireless Range	Up to 65 Feet

Plugable USB Bluetooth Adapter

$24.00· Plugable Technologies
4.5(53)

Bluetooth Version	Bluetooth 5.0 Dual-Mode
Linux Compatibility	Excellent Native Kernel Support
Wireless Range	Up to 130 Feet

Recommendation

- Choose the Xcellon Adapter if your goal is the **absolute lowest entry price** for a basic desktop setup.
- Choose the Plugable Adapter if you need **extended wireless range** to move around a room while recording or dictating notes.

The Linux Command Line Setup

Once your earbuds are paired to your MX25 machine, you can record your voice directly from the terminal using standard ALSA/PulseAudio/PipeWire tools. To record your dictation directly to a high-quality audio file for your pipeline, run:

Bash

```
arecord -D pulse -f cd local_dictation.wav
```

(Press `Ctrl+C` when you are finished speaking, then pass `local_dictation.wav` directly to your local Whisper + Ollama automation script).
If you want to map a keyboard shortcut to instantly start and stop recording your earbuds' mic input, I can show you how to bind a quick bash toggle script. Would you like me to guide you through that?

Can I Vox record
Yes, you can absolutely do VOX (Voice-Activated / Silence-Skipping) recording natively on Linux using standard audio tools.
GitHub
To achieve this without proprietary software, you use `sox` (Sound eXchange), the swiss-army knife of Linux audio processing. It has a built-in `silence` filter that acts as a real-time noise gate—it dynamically listens to your Bluetooth mic input, strips out any dead air, and only writes to disk when you are actually speaking.

Step 1: Install SoX on MX25
Open your terminal and pull down `sox` along with its MP3 and ALSA handlers:

Bash
```
sudo apt update && sudo apt install sox libsox-fmt-all -y
```

Step 2: The Core VOX Command
To record directly from your current active PulseAudio/PipeWire source (which will be your paired Bluetooth earbuds), run this command:

Bash
```
rec -v 1.0 dictation_vox.wav silence 1 0.1 1% 1 2.0 1%
```

How to read that `silence` tweak engine:
- **`1 0.1 1%` (The Trigger):** Tells it to wait for **1** period of sound above a **1%** volume threshold lasting at least **0.1** seconds before it starts writing to the file.
- **`1 2.0 1%` (The Stop/Pause):** Tells it to stop writing to the file if it detects **1** period of silence lasting longer than **2.0** seconds below the **1%** volume threshold.
(If you stop talking for more than 2 seconds, the recording stops or splits, dropping background hiss entirely).

The Pro Setup: Automated Audio Trimming
If you want to run a continuous background session where it clips out silence effortlessly and automatically chunks your thoughts into individual files for your local `faster-whisper` and Ollama setup, you can use a loop framework like this:

Bash
```
while true; do
    # Records a single block of speech and exits the moment you pause for 3 seconds
```

```
    rec -q chunk_$(date +%s).wav silence 1 0.1 1% 1 3.0 1%

    # Optional pipeline hook: push to your local LLM transcription pipeline here
done
```

Tweaking Hint: If your Bluetooth mic has aggressive ambient noise canceling or a high noise floor, change the `1%` values to `2%` or `3%` so it doesn't accidentally trigger on heavy breathing or keyboard clicks.

what about audacity

You can use Audacity on MX25 for this, too. It has a native feature built right into the interface called **Sound Activated Recording**.

Audacity Manual

If you prefer a graphic user interface (GUI) over a command-line script, it works well—though because it records everything into one long continuous timeline, it requires one extra step to export the clips for an AI pipeline.

Here is how to set it up:

1. Enable Sound Activation

1. Launch Audacity.
2. Go to **Edit** > **Preferences** (or press `Ctrl + P`).
3. Select the **Recording** tab from the left-hand menu.

   MakeUseOf
4. Check the box that says **Enable** under the **Sound Activated Recording** section.

   MakeUseOf

2. Calibrate the Level (Crucial for Bluetooth Mics)

Right below that checkbox, you will see a slider for **Sound Activation Level (dB)**.

Audacity Manual

- By default, it is usually set around `-30 dB`.
- **If it keeps recording background hiss:** Move the slider to the right (e.g., `-25 dB` or `-20 dB`) to make it less sensitive.
- **If it cuts off the beginning of your words:** Move it to the left (e.g., `-35 dB` or `-40 dB`) to make it more sensitive.

Click **OK** to save.

The Workflow Difference

When you hit the **Record** button (or press `R`), Audacity will instantly go into a blue **Pause** state. The moment you speak into your Bluetooth earbuds, it unpauses and visualizes your track. When you stop talking, it pauses again.

Managing the Audio Files:

MakeUseOf

While `sox` in the terminal easily spits out separate independent `.wav` files every time you speak, Audacity bundles all your spoken thoughts into a single track, simply skipping over the silences.
To get those chunks out to feed your local Whisper + Ollama environment:

1. When you are done talking, hit **Stop**.
2. Go to **Analyze > Silence Finder** or **Label Sounds** to automatically mark the gaps between your audio blocks.
3. Go to **File > Export Audio > Export Multiple** to split the project into separate named tracks (like `chunk-1.wav`, `chunk-2.wav`) all at once.

is Analyze > Silence Finder or Label Sounds just for splitting
In Audacity, **Silence Finder** and **Label Sounds** are primarily used to auto-detect gaps for splitting, but their underlying engine serves a dual purpose: **audio analysis and cleanup**.
You can repurpose these tools to act exactly like an offline, post-processing **VOX (Voice-Activated) gate**. Instead of just chopping the audio up, you can use them to isolate, audit, or completely erase background hiss, mouth clicks, or dead air before exporting your track to an AI model.

1. Stripping Out Backgound Noise (The Visual Noise Gate)

If you don't want to split your files, you can use **Label Sounds** to identify exactly where you spoke, and then use that selection to silence the background.

- **The Trick:** Run **Analyze > Label Sounds** to mark all spoken audio segments.
- **The Cleanup:** You can then select the labeled regions, invert your selection (`Ctrl + Shift + I`), and hit `Ctrl + L` (**Silence Audio**). This instantly zeros out the waveform in every gap, leaving you with a pristine, silent background between your words without changing the timeline or splitting the file.

2. Auditing Your Bluetooth Mic's Behavior

Because Bluetooth earbuds often have aggressive, built-in ambient noise cancellation (ANC) or delayed mic activation, they can truncate the first syllable of a sentence.
Using **Label Sounds** gives you a precise visual map of your audio behavior:

- Look at the generated labels. If the label start-point consistently slices off the very beginning of your spoken words, your threshold is set too high.
- You can adjust the **Minimum silence duration** and **Threshold** settings in the prompt box to fine-tune how aggressively Audacity detects audio, ensuring no syllables are lost before sending the file to Whisper.

3. Creating a "Timeline Map" for Local LLMs

If you use Audacity's **File > Export Other > Export Labels** feature, it saves a tiny text file containing precise timestamps of every spoken chunk:

Plaintext

```
0.500000    4.200000    Sound 1
7.100000    12.300000   Sound 2
```

If you are building an advanced local AI pipeline, you can parse this tiny text file with a shell script. Instead of making Whisper process hours of blank room hiss, your script can read these timestamps and tell `ffmpeg` to slice out and transcribe only the active audio blocks, slashing your local GPU/CPU processing times significantly.
Would you like a quick script that parses an Audacity label file to automate local Whisper transcription for only the marked segments?

yes
Here is a Python script tailored for your **MX25** setup. It works by monitoring a directory where you save your Audacity label files.
When you export a label file, this script reads the exact start and end timestamps, uses `ffmpeg` to slice only those voice-activated chunks out of your main recording, and pipes them directly to **Ollama** and a local **Faster-Whisper** engine. This eliminates processing hours of silence or background hiss.

1. Prerequisites
Make sure your Python virtual environment has the necessary dependencies installed:

Bash
```
source whisper_env/bin/activate
pip install faster-whisper requests
```

2. The Automation Script (`process_labels.py`)

Python
```
import os
import re
import subprocess
import requests
from faster_whisper import WhisperModel

# --- CONFIGURATION ---
AUDIO_FILE = "dictation_session.wav"      # Your main Audacity recording
LABEL_FILE = "dictation_labels.txt"      # The exported Audacity label file
OUTPUT_DIR = "./processed_chunks"        # Where clipped audio chunks go
OLLAMA_URL = "http://localhost:11434/api/generate"
OLLAMA_MODEL = "llama3.2"                # Change to your preferred local model

os.makedirs(OUTPUT_DIR, exist_ok=True)

# 1. Initialize Faster-Whisper (Uses CPU Int8 for low overhead on Debian/MX)
print("Loading local Whisper model...")
whisper = WhisperModel("base", device="cpu", compute_type="int8")

def parse_labels(label_path):
    """Reads Audacity standard label file formats."""
    chunks = []
    with open(label_path, "r") as f:
        for line in f:
            if not line.strip():
                continue
            # Audacity formats labels as: [start_time]\t[end_time]\t[label_name]
```

```python
            parts = line.replace("\n", "").split("\t")
            if len(parts) >= 2:
                chunks.append((float(parts[0]), float(parts[1])))
    return chunks

def extract_audio_chunk(input_audio, start, end, index):
    """Uses ffmpeg to losslessly extract speech segments."""
    output_chunk = os.path.join(OUTPUT_DIR, f"chunk_{index}.wav")
    duration = end - start

    # Fast, copy-codec slicing
    cmd = [
        "ffmpeg", "-y", "-ss", str(start), "-i", input_audio, 
        "-t", str(duration), "-c", "copy", output_chunk
    ]
    subprocess.run(cmd, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
    return output_chunk

# --- MAIN EXECUTION PIPELINE ---
if not os.path.exists(LABEL_FILE) or not os.path.exists(AUDIO_FILE):
    print("Error: Ensure both the audio file and Audacity label file exist.")
    exit(1)

speech_segments = parse_labels(LABEL_FILE)
print(f"Found {len(speech_segments)} voice-activated segments to process.\n")

full_transcript = []

for i, (start, end) in enumerate(speech_segments):
    print(f"Processing segment {i+1}: {start}s to {end}s...")

    # Slice the audio file
    chunk_path = extract_audio_chunk(AUDIO_FILE, start, end, i)

    # Transcribe the audio chunk locally
    segments, _ = whisper.transcribe(chunk_path, beam_size=5)
    chunk_text = " ".join([seg.text for seg in segments]).strip()

    if chunk_text:
        full_transcript.append(chunk_text)
        print(f"  [Whisper]: {chunk_text}")

# Consolidate text and send to Ollama
complete_text = " ".join(full_transcript)

if complete_text:
    print("\nSending consolidated text to Ollama...")
    payload = {
        "model": OLLAMA_MODEL,
        "prompt": f"Clean up, structure, and provide action items for this voice dictation:\n\n{complete_text}",
        "stream": False
    }

    try:
        response = requests.post(OLLAMA_URL, json=payload)
        print("\n=== LOCAL AI ANALYSIS ===")
        print(response.json()['response'])
    except requests.exceptions.ConnectionError:
```

Conversation with Gemini