Audion Voice AI - User Guide

Contents

- 1. Choose an Edition
- 2. Installation
- 3. API Keys
- 4. Main Window
- 5. App Language and Transcription Language
- 6. File Transcription
- 7. Supported Formats
- 8. API Workflow
- 9. Local Models Workflow
- 10. CUDA Workflow in Studio
- 11. Live Dictation
- 12. Overlay
- 13. Live Text Cleanup
- 14. Export
- 15. Tray
- 16. Settings Persistence
- 17. Reset App
- 18. Cleanup
- 19. Recommendations
- 20. Troubleshooting
- Studio Batch Checklist
- API And Local Processing
- Batch Monitoring
- Recovery
- Interface And Configuration Reference

This guide describes the current workflow for Audion Voice AI Live and Audion Voice AI Studio.

1. Choose an Edition

Use **Audion Voice AI Live** when you need a compact distribution with API Live and local models. This is the primary laptop and daily-use build.

Use **Audion Voice AI Studio** when you have a CUDA workstation and need fast local large-model processing, faster-whisper, PyTorch, and GPU diarization.

2. Installation

Installing the recognition models

The build ships without model weights — about 7 GB: the GigaAM cache and whisper.cpp with large-v2 and large-v3-turbo; with pyannote for speaker separation (NVIDIA only) about 10.6 GB. Weights carry their own licences, apart from the licence of this program, and pyannote's terms are accepted personally on HuggingFace — so you download them yourself, under your own account.

Until the models are installed, transcription will not run: the window opens, but there is nothing behind it to recognise speech.

On the first start the app offers to download what is missing: the "Download models and engines" window lists the modules with their download size: GigaAM, whisper.cpp CUDA with the Turbo model, the Large V2 model and, on an NVIDIA machine, the speaker-separation layer; about 10.6 GB together. "Download and install" installs them one after another; progress is shown on the Maintenance page, and once everything is in place every mode there is green. "Later" postpones the question until the next start, "Don't ask again" hides the window for good. Modules can still be installed by hand on the same page.

The same can be done by hand: run `builder_main.cmd` and pick, in order:

#	Menu entry	What it installs
08	`GIGAAM ONNX`	the main Russian speech recognition model
09	`WHISPER.CPP CUBLAS`	the whisper.cpp engine with CUDA acceleration
10	`WHISPER.CPP LARGE V2`	the primary model for Russian files
11	`CUDA / PYANNOTE`	speaker diarization — optional

The first three are required. Step 11 is only needed if you label dialogue by speaker; it also requires accepting the model's terms on HuggingFace.

Open `builder_main.cmd` or the `Maintenance` tab in the GUI.

Recommended order:

1. Start the app. On the first start the "Download models and engines" window installs everything recommended with one button; the remaining steps are for manual repair or checks.
2. `builder_main.cmd` is only needed when the app is not built yet: it checks the folder structure and installs the Python runtime.
3. FFmpeg ships with the distribution. `Reinstall` on the `Maintenance` page is only needed after an NVIDIA driver change, because the build is picked to match the driver.
4. Live dependencies (microphone, streaming dictation) are installed by the GUI itself at start-up from `wheelhouse\live`, without Internet access. The manual row remains available for repair.
5. `Dependency wheel cache` (`wheelhouse`) ships with the distribution and shows as `Installed`. `Reinstall` is only needed if the folder was deleted: it downloads the GigaAM/ONNX Runtime wheels again.
6. Click `Check` in the `Microphone check` card: it tests the Windows default recording device, then a separate communications default, including native 44.1/48 kHz modes. Audio is not saved.
7. `GigaAM ONNX pack`: onnx-asr, the ONNX Runtime provider, the GigaAM v3 models and Silero VAD for splitting long recordings.
8. `whisper.cpp pack`: Live installs the CPU build, Studio the CUDA/cuBLAS build; the Turbo model comes with it.
9. Studio: `whisper.cpp Large V2 model` is the primary file model in CUDA mode. `GPU diarization` (torch + pyannote) is optional and NVIDIA only.
10. Run base verify/smoke checks.

The Maintenance tab shows the detected GPU, recommended profile, progress, speed, and ETA. Installer output is routed automatically to the left `Activity log`; there is no empty terminal area on the right. Rows are marked `Recommended`, `Optional`, or `Not needed`; non-recommended actions remain legible and available.

`builder_main.cmd` and the GUI use the same install scripts. The builder is for the first portable build, recovery, and manual maintenance; regular users install modules from the GUI `Setup` tab once the app can launch.

3. API Keys

API modes read keys from `config`.

```
config/
  api_key_*.txt
```

Reset App does not delete these files. Cleanup must also protect working configs.

4. Main Window

Main tabs:

- `Live` - dictation, overlay, API/Local sources, models, and OpenAI cleanup.
- `Files` - file queue, models, transcription language, post-processing/cleanup, subtitles, and export.
- `Settings` - theme, app language, tray, Notion/Obsidian integrations, global options, and settings reset.
- `Maintenance` - recommended profile, runtimes, payloads, models, and service actions.

The left side contains the log and queue. The right side contains workflow settings. Inactive backend cards are hidden or dimmed so API, Local Models, and CUDA settings are not confused.

Right-click the compact top overlay for an 85%-opaque quick menu. Its first actions are `Past dictations` and `Paste last dictation`, followed by Live dictation mode, file recording mode, `Input` / `Output`, the main window, Settings, and Exit. The quick overlay window shows the latest 20 entries. Tray `All dictations` opens the scrollable cache of up to 200 completed Live dictations; the oldest entries are removed automatically at the limit. Click a two-line card to paste it back into the previously active window; use its compact round buttons to copy or delete it. In Live mode the ready overlay shows the microphone; in file mode it shows a red Record symbol and `Start recording`. During a file recording, Pause omits that interval from the WAV while keeping the microphone open, Stop saves, and Cancel discards the temporary recording. The overlay and tray have no tooltips because every action is labeled or uses standard transport symbols.

The default overlay scale for a fresh installation is 70%. Later changes are saved. The mouse wheel scrolls the Settings page without changing the scale value under the pointer.

The far-left `⋮` kebab is visible beside the ready microphone. Drag it to move the overlay before recording without starting voice capture.

The `Right Alt + F12` hotkey is activated automatically with the application. Local STT has a separate **Preload local model** option: it speeds up the first dictation but reserves RAM/VRAM in advance. It does not control the overlay, tray, or hotkey readiness.

5. App Language and Transcription Language

These are separate settings.

- **App language** changes the interface.
- **Transcription language** tells the engine what language is spoken in the recording.

Use `Auto` when the language is unknown. If the recording is definitely Russian or English, choose the exact language.

Live dictation has a separate **Primary language**. Its default is `Window layout`: when dictation starts, the app reads the current Windows keyboard layout of the actual target window where text will later be pasted (`ru` or `en`). Russian, English and Auto explicitly override it. The shared **Term dictionary** is visible on both the `Live` and file-operation tabs in every API/Local mode. It contains only exact spellings of names, abbreviations and formats, separated by commas. The dictionary is sent to Live and file STT as provider keyterms/prompt where the selected engine supports prompting, and protects spellings during optional transcript formatting. GigaAM ONNX accepts neither prompts nor hotwords, so the dictionary does not change its raw recognition. The separate **Recording description** field is free-form context about the topic, participants, purpose and important facts; the dictionary does not need to be repeated there.

6. File Transcription

1. Open `Files`.
2. Add files or a folder with the separate round picker buttons beside `Add...`. Use the checkboxes in the first queue column to target particular files. If no rows are checked, Start processes the complete queue; bulk remove/cleanup actions still fall back to ordinary row selection.
3. Choose the file engine: OpenAI, Local Models, or CUDA in Studio.
4. Choose the transcription language.
5. For OpenAI, choose a transcription profile; for local processing, choose the model and backend.
6. Enable required export formats.
7. Start processing.

Results are saved next to the source file by default. This is better for large archives because the transcript stays with the original recording.

A folder is added with all its subfolders: supported files join the queue as a flat list and each transcript is saved next to its file in that folder. The full format list (17 audio and 18 video) is shown in the tooltips of `Files…`, `Folder…` and the `Formats` label. External files selected through the picker or drag-and-drop are not copied into the project. The queue retains their original paths and processes them in place. Only WAV files recorded by Audion itself are created automatically in `input`. The `Input` and `Output` buttons share the remaining width and open those project folders.

7. Supported Formats

The app relies on FFmpeg, so common audio and video formats are preferred:

- audio: WAV, MP3, M4A, AAC, FLAC, OGG, OPUS;
- video: MP4, MOV, MKV, WEBM, AVI.

If a file uses an exotic container, convert it to WAV, MP3, or MP4 first.

8. API Workflow

OpenAI is the fastest path for high-quality cloud transcription and text cleanup. In Live, the OpenAI model is not selected manually: Realtime uses `gpt-realtime-whisper`, and batch fallback uses `gpt-4o-mini-transcribe`. xAI and ElevenLabs are currently used as fixed realtime Live providers.

Important settings:

- valid API key for the selected provider;
- OpenAI file transcription profile;
- post-processing model;
- cleanup prompt when text cleanup is enabled.

OpenAI file profiles:

- `Fast / economical` -> `gpt-4o-mini-transcribe`;
- `Max accuracy` -> `gpt-4o-transcribe`;
- `With diarization` -> `gpt-4o-transcribe-diarize`.

Each profile has a tooltip explaining the tradeoff. The model refresh button remains useful for post-processing and cleanup models.

9. Local Models Workflow

Local Models keep audio out of external APIs. They are useful for private work, long recordings, and machines with an installed GPU/CPU backend.

Local models:

- GigaAM - preferred local choice for the Russian UI layout;
- whisper.cpp - CPU fallback in Live and CUDA/cuBLAS GPU pack in Studio;
- backend: auto, CUDA, DirectML, or CPU fallback;
- GigaAM Live keeps the model warm until the app exits;
- GigaAM cuts long recordings into pieces of up to 25 seconds at the quietest points (by Silero VAD, installed with the GigaAM ONNX pack; by signal energy without it). Nothing is discarded, quiet echoing speech still reaches the model, and every piece gets start and end times;
- unload/buffer threshold applies to whisper.cpp live scenarios.

Install the required runtimes/payloads and models from the `Maintenance` page before using this mode (or accept the first-start window). The GigaAM/ONNX Runtime wheels ship in the distribution's `wheelhouse`: `GigaAM ONNX pack` installs `onnx-asr` and the ONNX Runtime provider from them, preloads `gigaam-v3-e2e-ctc`/`gigaam-v3-e2e-rnnt` into `models\huggingface` and fetches Silero VAD. On Windows, auto uses DirectML as the lightweight universal backend; Studio uses CUDA on NVIDIA.

In Studio, `Transcription + diarization` runs `pyannote` as a second pass for local file engines, including GigaAM. For GigaAM, the app automatically uses shorter chunks (`diarization.gigaam_chunk_seconds`, 45 seconds by default), because GigaAM returns chunk-level text rather than word timestamps. Live mode does not use pyannote.

9.1. Local Backend Installation

Audion chooses the backend by actually loading provider DLLs/runtimes, not just by GPU name.

- **DirectML**: the lightweight Windows fallback for Live and a reserve path for Studio. `GigaAM ONNX pack` installs it as `onnxruntime-directml`; no external SDK is required. Keep the NVIDIA/AMD/Intel driver current.
- **CUDA**: the NVIDIA path for Studio. Install the current NVIDIA driver and Studio runtime/payloads from `Setup`/`builder_main.cmd`, then run `GigaAM ONNX pack`. ONNX Runtime uses `onnxruntime-gpu`; CUDA/cuDNN/MSVC DLLs must be visible to the process, and Audion also calls `onnxruntime.preload_dlls()`.

- **TensorRT**: not used in the current project profile. If ONNX Runtime exposes a TensorRT provider, Audion does not select it as a recommended backend.

10. CUDA Workflow in Studio

CUDA is available only in Studio. It targets NVIDIA GPUs and covers GigaAM ONNX CUDA, whisper.cpp CUDA/cuBLAS, and faster-whisper/pyannote.

The `CUDA` card includes a Faster-Whisper profile switch:

- `Quality` - the default mode. It uses regular Faster-Whisper/CTranslate2 without batched inference. GPU load is calmer, and it is safer for rough speech, quiet intros, production chatter, and later diarization because the timeline is usually more detailed.
- `Speed` - enables BatchedInferencePipeline with `batch_size=16`. This mode keeps CUDA/GPU much busier and speeds up long files or file queues, but it depends more on VAD, can merge larger segments, and can occasionally miss quiet boundary phrases. Use it when you need a fast run and can dedicate the GPU to the job.

Typical flow:

1. Install `whisper.cpp pack` (CUDA/cuBLAS) and `whisper.cpp Large V2 model`; the first-start window does this by itself.
2. Install `GPU diarization` (CUDA/pyannote) when speaker labels are needed.
3. Run verify.
4. Turbo remains the fast profile for comparisons against large-v2.
5. Choose the Studio GPU engine: GigaAM CUDA, whisper.cpp cuBLAS, or faster-whisper CUDA.
6. For speaker labels, choose `Transcription + diarization`.
7. Run a short smoke test on a small file.
8. Process long recordings after the smoke passes.

RTX 5070 smoke confirmed the target Studio GigaAM CUDA and Studio whisper.cpp CUDA/cuBLAS profiles. On a machine without NVIDIA GPU, CUDA smoke can only check imports.

11. Live Dictation

Live dictation can be started from the `Live` tab, the record button above the log, or the tray.

Modes:

- **API models** - OpenAI batch, OpenAI Realtime, xAI Realtime, or ElevenLabs Realtime. The OpenAI model is selected by the app and is not exposed as a catalog.
- **Local Models** - GigaAM or whisper.cpp with the selected backend.

Choose the source first (`API models` or `Local Models`), then choose the concrete provider or local model. The inactive card remains dimmed so the screen does not feel empty while still avoiding accidental configuration changes.

When `On startup` is enabled, dictation starts with the app.

12. Overlay

The overlay reserves the full controller's 420-680 px native width. In idle, a window mask leaves only the centered 120×12 oval visible and interactive. Hover clears the mask and immediately reveals the full-width Record capsule with a stationary microphone icon at its exact center. Clicking fades the icon over roughly 220 ms and swaps in the recording controls inside the same bounds without resizing or moving the window. The minimum/default height is 52 px.

The left `Activity log` stays empty until dictation, an error, or a maintenance operation occurs. The armed state and `Right Alt + F12` reminder are shown in the microphone tooltip and status bar.

The `Live` tab allows you to:

- enable or disable the overlay;
- adjust overlay height;
- control overlay behavior during live dictation.

13. Live Text Cleanup

Enable `Live Cleanup` for long dictation sessions.

Settings:

- cleanup model;
- cleanup prompt;
- sentence count threshold.

The app can periodically turn raw dictated text into a cleaner document.

14. Export

Export is available from the GUI and tray.

Formats:

- Markdown;
- TXT;
- JSON;
- SRT;
- WebVTT.

Actions:

- save the log to Markdown through a file picker;
- export a transcript;
- send material to Notion;
- send material to Obsidian.

15. Tray

The tray is for quick actions while the main window is hidden.

Minimizing the window hides the app in the tray. Closing the window exits the app completely, stops background work, and removes the tray icon.

Available actions:

- show or hide the app;
- start or stop live dictation;
- export the log to Markdown;
- send results to Notion or Obsidian.

Tray behavior can be disabled in `Settings`.

16. Settings Persistence

The app saves the following between restarts:

- theme;

- app language;
- checkboxes;
- filled fields;
- selected models;
- model lists;
- live settings;
- export settings;
- tray, integrations, and global app behavior settings.

Combo boxes do not change on mouse wheel, so accidental scrolling cannot alter workflow settings.

17. Reset App

`Reset App` restores default program settings. Use it when the interface or workflow was accidentally configured into a bad state.

Reset App must not delete:

- API keys;
- installed runtimes;
- payloads;
- downloaded models;
- user work files.

18. Cleanup

`cleanup_project.cmd` removes what can be restored on another system: temporary build artifacts, runtime, `Tools`, models, `install\download`, `install\wheels`, and working payload folders.

It also clears `input`, `output`, `logs`, `report`, `workspace`, and `release`. This is expected: `input` is a temporary working area for files, not a user archive.

Working configs, API keys, install scripts, `system_core`, `Docs`, `tests`, and important root project files are protected. After cleanup, the empty structure is recreated through `install\init_folders.cmd`.

The GUI module catalog and `builder_main.cmd` are synchronized. The target profiles were smoke-tested on RTX 5070: Live GigaAM DirectML, Live whisper.cpp CPU fallback, Studio GigaAM CUDA, and

Studio whisper.cpp CUDA/cuBLAS.

19. Recommendations

- Use API models for short and simple jobs.
- Use Local Models for private or long local work.
- Use Studio and CUDA for large archives on NVIDIA GPUs.
- Save output next to the source file.
- Do not confuse app language with transcription language.
- Run a short smoke test before processing multi-hour recordings.

20. Troubleshooting

- Check the log on the left.
- Make sure FFmpeg is installed.
- Check API keys for OpenAI, xAI, or ElevenLabs.
- For Local Models, check the model and runtime.
- For CUDA, check the NVIDIA driver and that the CUDA/cuBLAS `whisper.cpp pack` with the Large V2 model is installed; PyTorch is only needed for diarization.
- Long GigaAM recordings are cut into pieces of up to 25 seconds automatically. If a piece still fails with a DirectML error, update the GPU driver or switch the backend to CPU.
- Studio on a machine without NVIDIA: turn off `Transcription + diarization`, otherwise the pipeline stops at the missing pyannote after transcription.
- If models were installed by hand and the `Mode readiness` matrix is not green, open `Maintenance`: the row marked `Not installed` shows what is missing.
- Use Reset App if the issue looks like broken UI configuration.

Studio Batch Checklist

Review the complete source list, output folder, language, engine, model, diarization, timestamp, and export settings. Run one representative file before the batch and verify text quality, speaker boundaries, timing, and filenames.

For CUDA, confirm the NVIDIA driver, compatible PyTorch/runtime stack, model payload, and available VRAM. For CPU or DirectML fallback, expect different speed and potentially different backend

capabilities. Record the actual profile used by an accepted batch.

API And Local Processing

API routes require network access, key, model, quota, and data-policy approval. Local routes require installed backends and models. Keep provider errors separate from decode, FFmpeg, model, and output-format errors.

Do not send sensitive audio to an API when the task requires offline processing. Transcripts, subtitles, speaker maps, reports, and logs may be sensitive even after source audio is removed.

Batch Monitoring

Follow per-file progress and output timestamps. A quiet model load or provider wait is not automatically a hang. Check the child process, GPU/CPU activity, log, and last completed file before stopping the job.

Prevent overlapping jobs from writing to the same output. Keep enough disk space for decoded audio and temporary segments.

Recovery

Preserve the failed file, log, selected engine/model, hardware profile, and parameters. Retry only failed files when output naming is stable. Use Reset App for UI/settings corruption, not for missing dependencies or incompatible CUDA components.

Interface And Configuration Reference

Declarative GUI settings and configuration maps are structured documentation sources for available engines, models, fields, defaults, tooltips, and backend actions. This guide turns them into a production sequence: prepare a batch, choose local or API processing, estimate hardware and privacy impact, monitor files independently, and accept transcripts with evidence.

During release review, compare the visible controls, persisted settings, generated engine request, per-file reports, exports, and documentation. New batch options must state whether they affect decoding, segmentation, recognition, diarization, cleanup, export, concurrency, cost, or retention. Conditional controls must explain when they appear and what fallback occurs when a local model or CUDA component is unavailable.