

stems

UNDER THE HOOD · STABLE AUDIO 3 · V1

how a sandboxed audio unit in logic gets stable audio 3 to write in whole bars and in your key, on the laptop — eight bars in about six seconds, a hundred and twenty-eight in about fifty.

this is the engineering behind stems v1: how the stems app sets the model up, where it lives, how it is served, what one generation does from key press to audio, and the numbers we measured. it's written for whoever builds the next version.

LOGIC PRO · THE HOST PROCESS



OUT: POST /GENERATE · JSON + B64 WAV
BACK: NDJSON · STEP I/8 · DONE + WAV

SA3_BRAIN.PY · A LAUNCHAGENT, FROM LOGIN



127.0.0.1 ONLY · BYTES IN, BYTES OUT
NO SHARED FILES · ONE READ-ONLY STATE.JSON

two processes, one wire. logic hosts the plug-in; the brain hosts the model. they only ever exchange json and audio bytes over the loopback interface.

the model can't live in the plug-in.

an auv3 is an app extension: sandboxed, hosted by logic's out-of-process audio unit service, and expected to be small. stable audio 3 medium is a 6.9 gb python/mlx program. so the plug-in stays a thin client and the model runs next door.

no processes	a sandboxed appex can't spawn python, bash or a launchd job. so the brain lives outside it: the stems app (the unsandboxed developer id container) installs a launchagent (com.s7ems.stems.brain, runatload + keepalive, --lean). when the plug-in finds nobody home it reads state.json through one read-only home-relative entitlement to tell 'not set up' from 'not running', and its open stems key opens stems://start – launchservices opens are allowed in the app sandbox (verified under apple's own application.sb profile).
no shared files	the stems loop app talks to its brain with file paths. a sandboxed plug-in can only write inside its own container, and a process reading another app's container can trip macos's 'access data from other apps' prompt. so every byte travels in the request and the response: json in, base64 wav both ways.
loopback http	the appex already has com.apple.security.network.client. plain http to 127.0.0.1 also needs NSAllowsLocalNetworking in the appex's app transport security dictionary. the server binds 127.0.0.1 only – nothing on the network can reach it.
its own port	the stems loop app runs its brain on 7431 and kills any brain there that isn't its own, and its every /outpaint cancels all other jobs. ours is 7432 with its own state, so the app, the regular plug-in and this one never cancel each other.
set up once	the app runs its bundled brain/setup.sh: stability's mlx runtime pinned to the loop app's commit (tree-hash verified), a python 3.11 venv under pinned constraints, and the four stock weights from the stems cdn (with the user's free key) or hugging face – resumable, sha256-checked, moved into place only once verified. if the loop app's install is intact it's reused read-only instead. the brain never downloads and refuses to load the loop-0 fine-tune by name.

stable audio 3 medium, 8-step.

there is no model called 'stable audio 3 turbo'. stable audio 3 (stability ai, may 2026) ships small-music, small-sfx, medium and an api-only large. each open model comes as a post-trained checkpoint – adversarial relativistic-contrastive (ARC) post-training – that samples in 8 steps with no cfg, and a slower -base checkpoint for fine-tuning. the fast 8-step medium is the 'turbo' one, and it's what stems runs, stock.

PART	DOES	FILE	SIZE
t5gemma	text encoder (tokenizer inside)	t5gemma_f16.npz	0.57 gb
same-l encoder	audio → latents (a2a only)	same_l_encoder_f32.npz	1.70 gb
dit medium	1.4b flow transformer, 24 blocks × 1536	dit_medium_f16.npz	2.91 gb
same-l decoder	latents → 44.1 khz stereo	same_l_decoder_f32.npz	1.70 gb

the autoencoder compresses 4096×: one latent is 4096 samples, 92.9 ms, 256 channels – about 10.8 latents a second. the dit is conditioned on the t5gemma prompt embedding (padded to 256 tokens with a learned pad) plus a *seconds_total* embedding, and on nothing else: no bpm, no key, no seconds_start. sampling is the ping-pong flow sampler on a log-snr-shifted schedule, 8 steps, cfg 1.

one worker, many listeners.

process	brain/run.sh execs the runtime's own venv python (3.11, mlx $\geq$ 0.30, numpy, soundfile, sentencepiece) – no torch. a ThreadingHTTPServer on 127.0.0.1:7432 answers requests on threads; stdout prints one {"event":"listening","port":7432} line, logs go to stderr.
one mlx worker	mlx's gpu streams belong to the thread that made them, so every model load and every eval happens on a single worker thread fed by a job queue. http threads only enqueue and relay events.
supersede + cancel	a new /generate marks every earlier job cancelled (cancel_upto); /cancel does the same. the worker checks between stages and after every sampling step, so a cancel lands in under a second. a client that hangs up cancels its own job.
lean by default	lean (what setup installs): each stage loads what it needs and frees it after – ~0.9 gb idle, ~1 s slower a part, 4.34 gb peak even for a 258 s render. warm: all four models stay resident (6.4 gb) and /warm compiles the metal kernels before real work.
memory hygiene	mlx's buffer cache is capped at 512 mb so freed models actually leave. the encoder runs in 128-latent windows with 16 latents of context either side – same-l sees ± 12 latents, so the windows equal one pass (≈ 80 db snr) at bounded memory; that took a 120 s a2a peak from 11.3 to 8.5 gb. the decoder runs in 64-latent windows.
prompt cache	t5gemma embeddings are cached per prompt string, so a re-roll skips the text encoder.

four endpoints, one stream.

```
GET /health -> {"ok":true,"state":"cold|loading|idle|busy","model":"stable audio 3 medium · 8-step",
               "warm":bool,"last_elapsed":s|null,"error":str|null,"version":"0.1"}
POST /warm  -> ndjson ... {"event":"ready"}
POST /cancel -> {"ok":true}
POST /generate {"prompt":str, "seconds":1..380, "steps":8, "seed":int|null,
               "init_audio":""|null, "init_noise_level":0..1, "negative_prompt":null, "cfg":1.0}
               -> application/x-ndjson, one json object per line:
{"event":"stage","stage":"queued"}           always first
{"event":"stage","stage":"loading"}           only when a model has to load
{"event":"stage","stage":"encoding"}          prompt (+ init audio)
{"event":"stage","stage":"sampling"}
{"event":"step","step":1,"total":8} ... {"event":"step","step":8,"total":8}
{"event":"stage","stage":"decoding"}
{"event":"done","seed":7,"seconds":18.0,"sample_rate":44100,"elapsed":5.65,"wav":""}
| {"event":"error","message":str} | {"event":"cancelled"}
```

streaming matters: the plug-in turns each step into the big key's sweep and the 'step 3 of 8' line, and can tell 'still loading' from 'stuck'. a bad request gets one error line and never disturbs the job that's running.

from key press to the downbeat.

8 bars of drums at 120 bpm, feel mid, in order:

	WHERE	WHAT HAPPENS
1	plug-in	bars → seconds: $4 \text{ beats} \times 60/120 = 2 \text{ s}$ a bar. asks for $8 + 1 = 9 \text{ bars} = 18 \text{ s}$ (one spare bar to cut back to the grid). up to 128 bars: the most whole bars whose length plus the spare fits the model's 380 s, said in words when it clamps.
2	plug-in	builds the prompt: TrackType: Instrument, Drum loop, dusty breakbeat with ghost notes, house style. Genre: House. Instruments: Drums. BPM: 120 – kept under 45 words; if over, your words give way from the end, never the structure. pitched parts also get your track's key before the tempo: ... Instruments: Bass. Key: A minor. BPM: 120.
3	plug-in	feel > 0: your track (the context mix, everything audible but this part) is peak-normalised to 0.95, looped to 18 s, resampled to 44.1 khz with core audio, encoded as a 16-bit wav. feel maps to init_noise_level: $0 \rightarrow 1.0$ (words only), $1 \rightarrow 0.55$.
4	brain	$T_{\text{lat}} = \text{ceil}(18 \times 44100 / 4096) = 194$ latents. prompt → t5gemma (cached) → cross-attention tokens + seconds_total(18) embedding.
5	brain	a2a: same-l encodes the init audio to [1, 256, 194]; the start is $\text{init} \cdot (1-s) + \text{noise} \cdot s$ and the ping-pong schedule begins at s instead of 1. t2a: pure noise from the seed.
6	brain	8 dit steps on metal, one ndjson step event each (~2 s for 18 s of audio, warm).
7	brain	same-l decodes in 64-latent windows, trims to exactly 793,800 samples, checks for nan, scales to -1 dbfs if hotter, never boosts; 16-bit wav, base64, done.
8	plug-in	downbeat: a 5 ms rms envelope, its positive flux, and the first rise inside the first beat that reaches 35% of the take's strongest attack. the cut lands on the first sample at 20% of that hit's peak, minus 2 ms of pre-roll; 3 ms fades at both ends.
9	plug-in	trims to exactly 8 bars, scores the take against the tempo grid, sets its level under your track (intake gain), keeps it as a take, plays the best of the press.

a thin client that knows music.

SA3Generator	a URLSession with a 600 s idle timeout between lines (a cold load, or encoding 380 s of your track, can be quiet) and 3600 s per render; session.bytes(for:) → .lines parses each event as it arrives. errors map to plain words: offline, cancelled, brain error, http status.
takes	one press renders 1-3 takes in series (each its own seed); a cancel keeps the takes that already landed. up to 8 are kept per part and < > flips between them.
roles	drums · bass · keys · pad · lead, each with its own default phrase, instrument field and intake level. older sessions' melody/voice parts map to lead – sa3 makes no words.
health	while the view is visible the controller polls /health every 2 s and sends the stage a brain message: offline · cold · loading · idle · busy. a poll that times out while our own part renders reads as busy, not off – a 128-bar render is a minute of heavy work. every webview call is hopped to main (webkit kills the process on off-main calls).
the key	KeyDetect.swift: hann-windowed 16384-point vdsp ffts spread over the audio → chroma from 60 hz to 2 khz → pearson correlation with the 24 rotated krumhansl-kessler profiles. sent when $r \geq 0.62$ (synthetic progressions score ~0.9, pads 0.65+, noise ~0.3, drum loops 0.54-0.61). auto uses your track, not the parts – a take in another key never drags the session along. each take's key is detected the same way and shown; v1 doesn't retune.
long parts	the session grows to the longest part (your track loops under it) and settles back when it goes; banks reach 386 s; a part drags out as its own bars; takes kept per part stop at 10 minutes of audio, so re-rolling 128 bars can't pile gigabytes into logic.
tests	a url-protocol stub stands in for the brain: ndjson parsing, done / error / cancelled, step progress, prompt builder, bar maths and the aligner – plus the headless ui suite. the brain has its own end-to-end test against the real model.

measured on an m3 pro, 18 gb.

WHAT	TIME
cold /warm – load all four + compile kernels	4.1-5.9 s
8 s text-to-audio, warm	2.8-3.3 s (sample ≈ 2.0 , decode ≈ 1.0)
8 s audio-to-audio, warm	3.4-4.2 s
8 bars @120 through the plug-in's code, lean, first part	5.7 s (18 s of audio)
8 bars @120 that follow those drums (a2a), lean	7.1 s
60 s / 120 s text-to-audio	10.4 s / 19.4 s
258 s (128 bars at 120) text-to-audio, lean	47.3-49.5 s · tempo and level steady across all four 32-bar sections
258 s audio-to-audio (feel), lean	67.9 s

a real 128-bar part through Conductor + SA3Generator	48.7 s end to end
120 s audio-to-audio	28.0 s (encode $\approx$ 9 s)
cancel during a 60 s render	$\approx$ 0.7 s
lean 8 s text / audio	4.1–4.5 s / 5.3 s
MEMORY	
warm, idle	6.4 gb on mlx (dit 2.7 · enc 1.6 · dec 1.6 · t5 0.5)
peak, 8 s / 120 s render	8.3 / 8.5 gb on mlx · $\approx$ 9.5–9.7 gb process
lean	$\approx$ 0.9 gb idle · 3.9 gb peak (8 s) · 4.34 gb peak (258 s, text or audio-to-audio)
wire	18 s of 16-bit stereo = 3.2 mb wav, 4.2 mb as base64 – milliseconds on loopback

why it's built this way.

mlx, not pytorch	stability's torch path for medium wants cuda + flash attention; on a mac it falls back and is known to add a static glitch. their official mlx port runs medium natively on metal with no torch at all.
8 steps, cfg 1	the ARC checkpoints are trained for it: more steps don't help, and cfg/negative prompts aren't what they were trained with. the contract still carries cfg for experiments.
text first, a2a for feel	sa3 has no 'write a part for this mix' mode – inpainting over a track re-renders the whole mix. text-to-audio with 'TrackType: Instrument' gives an isolated part; audio-to-audio from your track (feel) borrows its rhythm.
tempo by length	the generated drums measured 120.2 bpm by onset autocorrelation with nothing but 'BPM: 120' in the prompt and whole bars of length asked for – so stems never time-stretches.
key by hint	there's no key conditioning, but a 'Key: X.' field steers the key region (see 09). so stems sends it, shows each take's key, and leaves the choice to your ears.
align in the plug-in	the model's start can land a hair late and its end is ragged; asking for one bar more and cutting on the first hit is cheap, deterministic and testable.
not the neural engine	the loop app's ane + gpu split (5.1 s for a 96 s fork) is compiled for loop-0's weights. it can be built for the stock file – the obvious next speed-up.

a hint the model takes.

rhodes keys part, 100 bpm, stock sa3 medium, 2 seeds per row; key measured with krumhansl-schmuckler on chroma. phrasings 'Key: X.' (field), ', in X' and both were equivalent – stems uses the field form.

ASKED	LANDED	
no key (4 takes)	a minor ×2, c minor ×2	the model's default region
Key: A minor.	a minor ×2	exact
Key: D minor.	d minor, a minor	exact · a fifth up
Key: F# major.	c# major, c# minor	a fifth up
Key: Eb major.	bb minor ×2	a fifth up

exact 3/8, and all 8 on the asked tonic or a fifth away – against a/c minor when nothing is asked. a strong hint, not a lock. the v1 end-to-end run asked a 128-bar keys part for d minor and got d minor. prompt:

```
TrackType: Instrument, {role phrase}[, your words], {genre} style. Genre: {G}.
Instruments: {I}. Key: {Tonic} {major|minor}. BPM: {n}
tonics: C C# D Eb E F F# G Ab A Bb B · never for drums
over 45 words: your words give way first; the key and the tempo stay
```

from v1 to what's after.

neural engine	build the ane half for the stock dit and codec: the loop app's numbers say roughly 3-4× faster renders – a 128-bar part in well under 20 s.
retune	takes already carry their key; pitch-shifting a take a fifth back to your key would make the hint a lock.
keep takes	takes live in memory; only the playing take survives a reopen. write them next to the project like the regular plug-in's takes.
license	stable audio 3 is under the stability ai community license (plus gemma terms for the text encoder): free under \$1m annual revenue, registration for commercial use, an enterprise license above. the app shows the agreements and 'powered by stability ai'; the gemma use restrictions belong in the terms of service too.

WHERE IT LIVES

FILE	WHAT
brain/setup.sh	the install: reuse · runtime · python · weights (cdn / hf) · login item
brain/sa3_brain.py	the server: models, worker, sampler, a2a, ndjson
brain/run.sh · install-agent.sh	start it · keep it running from login
StemsHost/	the stems app: setup window, SetupProgress, stems:// · credits
StemsAU/KeyDetect.swift	chroma → krumhansl-kessler key, for your track and every take
brain/test_brain.py	end-to-end against the real model
StemsAU/SA3.swift	prompt builder, bar maths, aligner, SA3Generator
StemsAU/Conductor.swift	generatePart: context, takes, placement
StemsAU/web/index.html	the stage
docs/sa3-poc/	design spec, model research, the loop brain map

v1.0.0 · branch [release/v1](#) · github.com/mrcolo/stems-au