60-second takeaway
VoxCPM 2 full SFT did fit on a single RTX 3090 Ti, but vanilla full fine-tuning did not.
The working stack was `gradient_checkpointing: true`, `PagedAdamW8bit`, `PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True`, `batch_size: 1`, gradient accumulation, and a cleaned grouped train/validation/test split.
The final 9000-step checkpoint was not the best checkpoint. Held-out validation selected `step_0002000`.

Who this is for

This is for engineers trying to fine-tune a modern open-source TTS model on one 24 GB NVIDIA GPU:

- RTX 3090
- RTX 3090 Ti
- RTX 4090
- A10
- L40 / L40S

The practical question is not whether VoxCPM 2 has an official full-SFT entrypoint. It does. The practical question is:

What has to change before a 2B voice model can actually complete full SFT on a 24 GB card?

Short answer

Use LoRA first if you are still exploring a dataset. For VoxCPM 2, LoRA is fast, the checkpoints are small, and a 9000-step run completed in about 2.4 hours in our setup.

Move to full SFT when the target voice needs deeper adaptation than an adapter can provide. In our run, full SFT completed 9000 steps in about 5 hours with the memory stack below:

```
PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True \
VOXCPM_PAGED_OPTIM=1 \
```

```
python scripts/train_voxcpm_finetune_full_sft.py \
  --config_path conf/voxcpm_v2/voxcpm_finetune_all_female01.yaml
```

The config also needs:

- gradient_checkpointing: true
- batch_size: 1
- grad_accum_steps: 8
- clean train, validation, and test manifests
- frequent checkpoint saves
- post-hoc validation in a separate process

The important result:

Result	Value
Model	VoxCPM 2, about 2B parameters
GPU	RTX 3090 Ti, 24 GB
Dataset	IMDA NSC FEMALE_01
Clean train rows	10,776
Validation rows	599
Test rows	598
Final training step	8999, with step_0009000 written
Best validation checkpoint	step_0002000
Runtime	about 5 hours for 9000 steps
Peak VRAM behavior	effectively full card, stable at the ceiling

Why vanilla full SFT failed

VoxCPM 2 is not huge by LLM standards, but full SFT pays for more than weights.

For a 2B-parameter model in BF16 mixed precision, the rough memory bill is:

Component	Approximate size
BF16 weights	4 GB
BF16 gradients	4 GB
FP32 AdamW optimizer state	16 GB
Subtotal before activations	24 GB
Activations	no room left

That means the usual "2B should fit" intuition is wrong. The card is already spent before activation memory enters the picture.

Our failed phases showed the boundary clearly:

Phase	Change	Result	Lesson
0	Vanilla `torch.optim.AdamW`, `batch_size=2`	OOM at first forward	Activations did not fit
1	`AdamW8bit`, `batch_size=1`	OOM at first forward	Optimizer state was not the first problem
2a	Gradient checkpointing + GPU-resident 8-bit Adam	Step 0 worked, step 1 OOMed	Activations fit, optimizer state became persistent
2b	Added allocator tuning	Still OOMed	Fragmentation was not enough to explain it
2c	Gradient checkpointing + `PagedAdamW8bit` + allocator tuning	50-step full-SFT smoke completed	This was the first stable recipe

The diagnostic rule is useful beyond VoxCPM:

OOM during first forward means activation memory. OOM on the next forward after step 0 succeeds usually means optimizer state.

Adam state is lazy. It appears after the first `optimizer.step()`. If the run dies before that, changing the optimizer cannot fix the failure point.

The working memory stack

The successful stack attacked each memory class separately:

Constraint	Fix	Why it mattered
Activation memory	Gradient checkpointing	Stores fewer activations and recomputes layers during backward
Optimizer state	`PagedAdamW8bit`	Quantizes optimizer state and pages it through CPU memory
Allocation churn	`expandable_segments:True`	Reduces PyTorch allocator fragmentation under checkpointing and paging

Per-microbatch memory	batch_size: 1	Keeps the forward pass small enough
Gradient quality	grad_accum_steps: 8	Restores effective batch size without increasing microbatch memory

This ran at the ceiling. A VRAM snapshot during the production run showed the training process using about 23,074 MiB, with total GPU use around 24,112 MiB. In other words: it fit, but there was almost no spare headroom.

That matters for future recipes. Inline validation, audio generation, larger clips, or larger microbatches can push the run back over the edge. For heavy validation, use a separate post-hoc process without optimizer state loaded.

The dataset problem we almost missed

The first 2000-step preflight used the old manifest and completed, but the logs were noisy. The giveaway was a large stop-loss outlier around step 1610.

The manifest audit found:

Problem class	Count	Share
Empty-text rows	5,229	30.66%
Boilerplate rows	86	0.50%
Sub-100ms audio	439	2.57%

For TTS, empty text paired with real audio is not harmless. The model receives audio signal, but the text label says to stop immediately. With batch_size: 1 and grad_accum_steps: 8, a 31 percent contamination rate means almost every optimizer step sees at least one bad row:

P(step has at least one bad row) = 1 - (1 - p)^8

At p = 0.31, that is about 95 percent of steps.

After removing empty-text and boilerplate rows, the clean preflight changed immediately:

Metric	Dirty manifest	Clean manifest	Improvement
Max grad_norm	50.96	6.09	8.4x lower
Max post-warmup grad_norm	46.06	about 3.6	about 13x lower
Max loss/stop	27.000	0.345	78x lower

Step 1610 loss/stop	27.0	0.027	normal step

The cleaned split kept 11,973 rows, then grouped by parent recording before assigning train, validation, and test. That avoided slice leakage across splits.

Production run

The final run used the clean grouped split:

Split	Rows
Train	10,776
Validation	599
Test	598

The production launch used the known-good full-SFT stack:

- gradient checkpointing enabled
- PagedAdamW8bit
- PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True
- batch_size: 1
- grad_accum_steps: 8
- train-only process, with validation handled post-hoc

Completion evidence:

Item	Value
Final logged train step	8999
Final checkpoint	step_0009000
latest checkpoint state	{"step": 9000}
Final loss/diff	0.757150
Final loss/stop	0.000005
Final grad_norm	0.871526
Error scan	no Traceback, OutOfMemory, Killed, or RuntimeError markers
Run size before cleanup	159 GB

The run completed, but the disk cost was not subtle. Full-SFT checkpoints are operational artifacts, not tiny adapter files.

Checkpoint selection

After the 9000-step run, every numbered checkpoint was evaluated on the 599-row held-out validation set in a separate process.

Checkpoint	loss/total	loss/diff	loss/stop
step_0000000	1.068616	0.916918	0.151699
step_0001000	0.888265	0.838856	**0.049409**
step_0002000	**0.885899**	0.827947	0.057952
step_0003000	0.930273	0.823368	0.106905
step_0004000	0.915051	0.818194	0.096857
step_0005000	0.931633	0.814075	0.117557
step_0006000	0.959932	0.811755	0.148178
step_0007000	0.955816	0.809483	0.146333
step_0008000	0.959506	0.808552	0.150954
step_0008999	0.960920	**0.808400**	0.152520
step_0009000	0.960920	**0.808400**	0.152520

If you select by the configured total validation objective, the answer is step_0002000.

The late checkpoints had the lowest diffusion loss, but their stop loss got worse. That is exactly the final-checkpoint trap: training can keep improving one scalar while the validation objective tells a more complicated story.

Listening pass

Before checkpoint cleanup, we generated the same no-reference transcript from every numbered checkpoint:

Today I walked through the quiet morning light, and every small detail felt clear, steady, and close.

All 11 samples generated successfully. Durations ranged from 6.24 seconds to 7.68 seconds, so none were empty or obviously truncated.

The user read was:

- The fine-tuned checkpoints all sounded useful and Singaporean.
- The base-like checkpoint did not sound Singaporean.

- There was no obvious single-transcript listening winner across the fine-tuned checkpoints.

That is why the practical selection is `step_0002000`: validation had a clear winner, and the same-transcript listening pass did not give a strong reason to override it.

What to copy

For a 24 GB full-SFT attempt, copy the recipe shape, not every exact number.

Start with:

```
batch_size: 1
grad_accum_steps: 8
gradient_checkpointing: true
valid_interval: 99999
save_interval: 1000
```

Launch with:

```
PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True \
VOXCPM_PAGED_OPTIM=1 \
python scripts/train_voxcpm_finetune_full_sft.py \
  --config_path conf/voxcpm_v2/your_full_sft.yaml
```

Then evaluate checkpoints separately:

- keep all numbered checkpoints until validation and listening are done
- validate on a held-out manifest with zero source-recording overlap
- generate the same transcript from each checkpoint
- pick by validation unless listening clearly contradicts it

What not to copy

Do not copy these mistakes:

- Do not launch vanilla AdamW and hope 24 GB is enough.
- Do not assume an 8-bit optimizer fixes first-forward OOM.
- Do not run long full SFT before empty-text cleanup.
- Do not random-split sliced audio rows without grouping by source recording.
- Do not trust the final checkpoint just because it is final.
- Do not delete middle checkpoints before validation.

LoRA vs full SFT after this run

This run changed our priors, but not the starting recommendation.

LoRA is still the first experiment for VoxCPM 2 because it is faster and easier to compare. Full SFT is now a realistic escalation on 24 GB, not a fantasy. The trade is operational:

Path	Best use
VoxCPM 2 LoRA	fast dataset check, adapter A/B tests, small artifacts
VoxCPM 2 full SFT	deeper voice adaptation, branded voice defaults, validation-backed production checkpoints

The key is not "LoRA or full SFT". The key is when the evidence says the adapter boundary is the bottleneck.

Sources and related posts

- VoxCPM official repository - VoxCPM 2 release notes, CLI examples, and official fine-tuning entrypoints.
- VoxCPM fine-tuning guide - official LoRA and full fine-tuning documentation.
- VoxCPM 2 model documentation - model architecture, language support, and 16 kHz to 48 kHz AudioVAE details.
- VoxCPM2 Hugging Face model card - model details, 2B parameter note, and fine-tuning entrypoints.
- LoRA vs Full SFT for Voice Models
- Voice Cloning on a 24GB GPU
- VoxCPM 1.5 LoRA Finetuning on IMDA NSC FEMALE_01