60-second takeaway
Start with LoRA. It is faster, cheaper to store, easier to compare, and safer when you are still learning whether the dataset is clean.
Move to full SFT when the voice must become the model: strong accent retention, branded voice consistency, long-form production output, or a model architecture where adapter strength is not enough.
On a 24 GB GPU, full SFT is possible for some voice models, but not as a vanilla recipe. It needs activation checkpointing, paged optimizer state, clean manifests, explicit checkpoint retention, and validation that is separate from training loss.

Who this is for

This guide is for engineers fine-tuning open-source TTS and voice cloning models on a single 24 GB NVIDIA GPU:

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

The question is not whether LoRA is "better" than full SFT in the abstract. The practical question is:

Which training mode should you try first, and what evidence tells you it is time to switch?

Short answer

Use LoRA first when you are still exploring:

- You are not sure the dataset is clean.
- You need to compare multiple checkpoints quickly.
- You want small adapter files that are easy to archive.
- You want a deployment-time control knob such as `lora_scale`.

- You are adapting a single speaker and the base model already knows the language and speaking style.

Use full SFT when the adapter is not enough:

- The target voice has a distinctive accent or rhythm that zero-shot and LoRA do not lock onto.
- You need a branded voice to sound identical across hundreds of lines.
- You need the model's default behavior to change, not just an adapter overlay.
- You can afford larger checkpoints, slower iteration, stronger validation, and stricter data cleanup.

The rough rule:

LoRA changes how the model leans. Full SFT changes what the model is.

That is why LoRA is the right first move and full SFT is the escalation path.

What LoRA changes

LoRA freezes the base model and trains small low-rank matrices attached to selected layers. In practice, this means the base model still carries most of the prior knowledge: language coverage, pronunciation habits, acoustic priors, and general prosody.

What LoRA is good at:

- Pulling speaker timbre closer to a target voice
- Adding a style or accent bias without rewriting every weight
- Training quickly on one GPU
- Saving tiny checkpoints
- Running many ablations
- Hot-swapping adapters for A/B listening tests

What LoRA is not as good at:

- Fixing a base model that fundamentally does not handle the target language or accent
- Removing a strong pretrained speaking habit
- Guaranteeing long-form consistency when the base model keeps drifting
- Making the fine-tuned voice the model's default identity

The operational win is enormous. A VoxCPM 2 r=32 LoRA checkpoint from our run was about 70 MB. That makes it cheap to save every checkpoint and compare them later. For Qwen3-TTS, the LoRA path also gives a useful inference knob: a lora_scale around 0.3 to 0.35 sounded better than scale 1.0 in our benchmark.

What full SFT changes

Full supervised fine-tuning updates the model weights directly. Instead of asking a frozen model to lean toward the voice through adapters, you are rewriting the model's behavior with the target dataset.

What full SFT is good at:

- Making the voice identity more deeply baked into the model
- Adapting across more internal components than LoRA targets
- Improving consistency when the base model is close but not close enough
- Creating a single checkpoint that does not need adapter plumbing

What full SFT costs:

- Larger checkpoints
- Larger optimizer state
- Slower iteration
- Higher risk of catastrophic forgetting
- Higher sensitivity to dirty data
- More disk pressure
- Stronger need for held-out validation and listening sweeps

The same property that makes full SFT powerful also makes it dangerous. If your manifest contains empty transcripts, boilerplate, wrong sample rates, or bad alignment, full SFT can learn those errors across the full model. LoRA often limits the blast radius. Full SFT does not.

24 GB GPU reality

Vanilla full SFT fails on a 24 GB GPU because the memory bill is not just the model weights.

For a 2B-parameter voice model in BF16 mixed precision, the structural budget is roughly:

Component	Approximate size
BF16 weights	4 GB
BF16 gradients	4 GB
FP32 AdamW optimizer state	16 GB
Subtotal before activations	24 GB
Activations	Does not fit

That is why "it is only a 2B model" is the wrong mental model. By the time you include gradients, optimizer states, and activations, a normal full-SFT run has already spent the card before the forward pass has room to breathe.

The working recipe has to attack the right memory class in the right order.

Why optimizer swaps alone do not fix first-forward OOM

One trap we hit in VoxCPM 2 full SFT: switching to 8-bit AdamW did not fix an OOM that happened during the first forward pass.

That makes sense once you look at when Adam state exists. Adam's `m` and `v` buffers are lazy-allocated on the first `optimizer.step()`. If the process dies during forward, optimizer state has not had a chance to exist yet. So an 8-bit optimizer cannot save memory at the failure point.

The rule:

Forward OOM means activation memory. Step-1 OOM after step 0 succeeds means optimizer state.

Why the working VoxCPM 2 full-SFT recipe fits

The successful VoxCPM 2 full-SFT run on RTX 3090 Ti combined four controls:

Constraint	Fix	Why it matters
Activation memory	Gradient checkpointing	Recomputes layer activations during backward

		instead of storing all of them
Optimizer state	PagedAdamW8bit	Keeps 8-bit optimizer state paged through CPU memory instead of resident on GPU
Fragmentation	PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True	Gives PyTorch more room to handle allocation churn
Bad rows	Clean grouped split	Removes empty-text and boilerplate rows before training

That stack changed the result from "vanilla full SFT OOMs on 24 GB" to "full SFT completes and validates."

The cost was still real:

- VoxCPM 2 LoRA, 9000 steps: about 2.4 hours in our run
- VoxCPM 2 full SFT, 9000 steps: about 5 hours with the validated memory stack
- Full-SFT checkpoint: several GB
- LoRA checkpoint: tens of MB

So full SFT is feasible, but LoRA remains the better iteration loop.

Evidence table

These are first-party run outcomes from the Instavar voice-model benchmark series. Treat them as configuration-bound results, not universal model rankings.

Model	Training mode tested	24 GB result	Best evidence	What it teaches
Qwen3-TTS 1.7B	LoRA	Fits	Best checkpoint in our run was epoch 10, with lora_scale around 0.3 to 0.35	LoRA can work well, but inference scale is part of the model selection
VoxCPM 1.5	LoRA	Fits	Step 4000 no-prompt generation gave the cleanest deployable output	LoRA is the lowest-friction path when the base model is already close
VoxCPM 2	LoRA and full SFT	LoRA fits comfortably; full SFT fits with memory controls	Full SFT selected step 2000 by held-out validation, not the final 9000-step checkpoint	Full SFT is possible on 24 GB, but only with the right recipe and validation
CosyVoice 3	Failed full SFT, corrected LoRA	LoRA fits comfortably	Full SFT failed after epoch 1; corrected LoRA run stabilized and kept small checkpoints	Full SFT can cause catastrophic forgetting when the dataset or recipe is not right
IndexTTS2	Full SFT	Fits	Best practical checkpoint was step 14000, not the final step 15949	Full SFT can produce a strong baseline, but checkpoint retention is non-negotiable

The important pattern is not "LoRA always wins" or "full SFT always wins." The pattern is:

Training mode changes the failure surface.

LoRA's failures are often about adapter strength, target modules, scale, and under-adaptation. Full SFT's failures are about memory, forgetting, bad rows, checkpoint churn, and validation discipline.

Failure modes

1. Catastrophic forgetting

CosyVoice 3 is the cautionary example. The first run trained all 506M parameters and failed after epoch 1. The corrected LoRA rerun trained a small adapter, saved tiny checkpoints, and remained stable for many more epochs.

The lesson is not that full SFT is bad. The lesson is that full SFT needs stronger proof before you trust it. If the model starts losing general speech behavior, LoRA may be the better adaptation boundary.

2. Lazy optimizer state

If a full-SFT run OOMs during the first forward pass, changing AdamW to AdamW8bit is not enough. The optimizer states have not been allocated yet.

Mitigation order:

1. Reduce per-microbatch activation memory.
2. Add gradient checkpointing.
3. Then address optimizer state with 8-bit or paged optimizers.
4. Use allocator settings only after the structural memory classes are addressed.

This distinction saves time. It stops you from "fixing" memory that does not exist yet.

3. Empty-text contamination

TTS manifests are unusually vulnerable to empty transcripts paired with real audio. The model receives audio signal, but the labels say stop immediately. With gradient accumulation, even a modest contamination rate can taint most steps.

For batch size 1 and gradient accumulation N, the chance that a step contains at least one bad sample is:

$$1 - (1 - p)^N$$

At 10 percent bad rows and $N=8$, more than half the steps are tainted. At 30 percent bad rows and $N=8$, nearly every step is tainted.

Before a long voice-model run, audit for:

- Empty text

- Boilerplate such as "thanks for watching"
- Very short audio paired with non-trivial text
- Non-speech segments
- Duplicate or overlapping slices across train, validation, and test

Full SFT makes this more important because every trainable weight can absorb the contamination.

4. Final checkpoint trap

The best checkpoint is often not the final checkpoint.

From our runs:

- VoxCPM 1.5 LoRA: step 4000 was the practical best checkpoint.
- VoxCPM 2 full SFT: held-out validation selected step 2000, not the final 9000-step checkpoint.
- IndexTTS2: step 14000 beat the final 15949-step checkpoint.
- Qwen3-TTS: epoch 10 was the practical best region in the original run.
- CosyVoice 3: the failed full-SFT run degraded early, while the LoRA rerun needed per-epoch gating.

Save more checkpoints than you think you need. Do not delete them until the listening sweep is done.

Decision tree

Situation	Start here	Escalate when
You are testing a new dataset	LoRA	The adapter improves but still cannot lock speaker identity
You need a deployable voice this week	VoxCPM 1.5 LoRA or Qwen3-TTS LoRA	Long-form consistency or accent retention is not good enough
You need a branded voice across hundreds of lines	LoRA preflight, then full SFT	LoRA proves the data is clean but under-adapts
You are not sure the manifest is clean	LoRA or a short smoke test	Never launch full SFT before manifest cleanup
You have only a short reference clip	Zero-shot or LoRA	Full SFT needs labelled rows, not just a longer prompt

You need small artifacts and fast A/B tests	LoRA	Only move if adapter limits are audible
You need the model's default identity to become the target voice	Full SFT	Only after LoRA and validation prove the dataset is worth trusting

Practical recipe for a 3090 Ti or 4090

Step 1 - Start with LoRA

Use LoRA to answer the first question:

Is this dataset capable of moving the model in the right direction?

Run a short LoRA preflight:

- 100 to 500 steps
- Save at least two checkpoints
- Generate the same transcript from each checkpoint
- Compare against base model output
- Check whether speaker identity, accent, pacing, and noise improved

If the model does not move in the right direction under LoRA, full SFT is not the next move. The dataset or recipe is probably wrong.

Step 2 - Audit the data

Before any long run:

- Remove empty transcripts.
- Remove boilerplate.
- Verify sample rate.
- Group splits by source recording, not random sliced rows.
- Ensure zero audio overlap across train, validation, and test.
- Keep a fixed listening transcript for cross-checkpoint comparison.

This is more important for full SFT than for LoRA, but it helps both.

Step 3 - Run a longer LoRA baseline

For a model that already supports LoRA:

- Save frequently.
- Keep every checkpoint until evaluation is complete.
- Run a scale sweep if the model exposes `lora_scale`.
- Hot-swap adapters in one process when possible to avoid repeated base-model loading.

This gives you a strong baseline and a clear "is full SFT worth it?" comparison point.

Step 4 - Move to full SFT only with the full memory stack

For a 24 GB full-SFT attempt, do not start with vanilla AdamW and hope.

Use the full stack from the beginning:

```
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
```

The model-specific wrapper will differ by repo, but the principles are the same:

- `batch_size: 1`
- gradient accumulation for the effective batch
- gradient checkpointing on transformer blocks
- paged or offloaded optimizer state
- frequent checkpoint saves
- validation outside the training process when inline validation is too memory-heavy

Step 5 - Select by validation and listening

Do not pick the last checkpoint by habit.

Use:

- Held-out validation loss
- Same-transcript checkpoint sweep
- Long-form listening sample
- Failure-mode-specific checks such as stop loss, truncation, accent drift, and room-noise copying

If validation selects an early checkpoint and listening does not clearly prefer a later one, trust the validation-selected checkpoint.

Recommended path

For most teams:

1. Start with zero-shot to establish a baseline.
2. Run LoRA to test whether the dataset moves the voice in the right direction.
3. Clean the manifest before any long run.
4. Keep every checkpoint until listening evaluation is done.
5. Escalate to full SFT only when LoRA under-adapts and the voice needs to become the model's default behavior.

The mistake is treating LoRA vs full SFT as a philosophical argument. It is an escalation ladder.

LoRA is the first serious experiment. Full SFT is the production-grade escalation when the evidence says the adapter boundary is the bottleneck.

Sources and related posts

- VoxCPM official repository - documents both LoRA and full fine-tuning entrypoints for VoxCPM.
- VoxCPM fine-tuning walkthrough - official SFT and LoRA workflow reference.
- Hugging Face PEFT LoRA guide - adapter mechanics and PEFT usage.
- bitsandbytes optimizer docs - 8-bit and paged optimizer options.
- PyTorch checkpointing docs - activation checkpointing behavior.
- VoxCPM 2 Full SFT on a 24 GB GPU - the detailed runbook and validation evidence behind the VoxCPM 2 full-SFT claim.
- Qwen3-TTS LoRA Fine-Tuning - Scale Sweeps, Checkpoints, and Production Defaults
- VoxCPM 1.5 LoRA Finetuning on IMDA NSC FEMALE_01
- CosyVoice LoRA Fine-Tuning - What Worked, What Didn't
- IndexTTS2 Finetuning on IMDA NSC FEMALE_01
- Voice Cloning on a 24GB GPU - What Actually Works in 2026