Love that. Let's build this together in tiny, testable steps. I'll give you **what to write**, **what to check**, and **what "done" looks like**—no big blobs of code. You'll fill in the gaps.

I'm assuming PyTorch. If you're using something else, say so and I'll adapt.

Milestone 0 — Scaffold + Environment (15–30 min)

Goal: clean repo, pinned env, runnable "hello audio".

Do:

1. Create folders:

```css
src/
  data/
  models/
  train/
  utils/
tests/
```

2. Create `requirements.txt` with: `torch`, `torchaudio`, `numpy`, `soundfile`, `tqdm`, `einops` (optional).
3. Add a tiny script `src/utils/audio_check.py` that:
   - loads a WAV (or generates a 1 kHz sine at 16 kHz for 1 s),
   - prints `shape`, `dtype`, `sr`,
   - writes it back out.

Check:

- Running `python src/utils/audio_check.py` prints `(16000,)` or `(1,16000)` and saves a file you can play.

Done when: you can round-trip a 16 kHz mono signal without errors.

Milestone 1 — Fixed st ndow math (10–20 min)

Goal: lock the latency/stride math so all later code fits it.

Do:

- Decide these constants and put them in `src/utils/constants.py` :
  - `SR = 16000`
  - `FRAME_MS = 20 → FRAME = SR * FRAME_MS // 1000` (→ 320 samples)
  - `HOP_MS = 10 → HOP = SR * HOP_MS   // 1000` (→ 160 samples)
  - TasNet encoder kernel `L = 32` (2.0 ms) or `L = 20` (1.25 ms). Pick one (start with `L=32`).
  - Channels: `N=64` , `B=64` , `H=64` , `P=3` , `X=6` , `R=3` , `C=2` .

Check:

- Write a 5-line test that asserts:
  - `FRAME % (L//2) == 0` so overlap-add is simple.
  - `HOP <= FRAME` and `HOP % (L//2) == 0` .

Done when: these assertions pass.

Milestone 2 — Encoder/Decoder only (no separator yet) (30–45 min)

Goal: implement the learned analysis/synthesis filterbank and prove **perfect (or near) reconstruction** with overlap-add.

Do:

- In `src/models/encoder.py` , implement **one** `nn.Conv1d` with:
  - in=1, out= `N` , `kernel_size=L` , `stride=L//2` , `bias=False` .
  - Activation after it (ReLU or PReLU).
- In `src/models/decoder.py` , implement **one** `nn.ConvTranspose1d` that inverts the above:
  - in= `N` , out=1, `kernel_size=L` , `stride=L//2` , `bias=False` .
- Init weights with small random normal; that's fine for now.

Micro-test (no big code):

- Generate a 1 s sine and a 1 s white-noise clip at 16 kHz.

- Do: `e = enc(x)` , then `y = de`
Skip to content
- Match lengths (crop/pad tail if nee).
- Compute `SNR = 10*log10( sum(x^2) / sum((x - y)^2) )` .

Pass criteria:

- You likely won't get huge SNR yet (random filters!), but **it should not explode or vanish**. SNR > 5 dB is fine at this stage. We'll train it later.

Gotcha: Carefully track shapes: input `(B,1,T)` → enc `(B,N,T')` → dec `(B,1,T)` .

Milestone 3 — Streaming ring buffer (no model) (20–30 min)

Goal: a small class that yields **overlapping frames** and reassembles via **OLA** (overlap-add). This gives you the real-time skeleton.

Do:

- In `src/utils/stream.py` , write:
  - `FrameCutter(FRAME, HOP) → .push(x_chunk)` and `.pull()` frames (size `FRAME`).
  - `OLA(FRAME, HOP) → .add(frame)` and `.read(HOP)` to output continuous audio.
- For now, just pass input to output (identity), using OLA with a Hann or rectangular window.

Test:

- Feed a 2 s signal in 256-sample chunks into `FrameCutter` , for each frame call `OLA.add(frame)` , and periodically `OLA.read(HOP)` .
- After the stream, flush any remainder and concatenate.
- SNR between original and reconstructed should be **> 30 dB** if your windowing/OLA is correct (rectangular with integer overlap usually ≫ 30 dB).

Done when: identity streaming path is numerically clean.

Milestone 4 — One TCN block (tiny) and shape tests (45–60 min)

Goal: implement a single **depthwise** Skip to content d block and confirm residual & skip connections behave.

Do:

- In `src/models/blocks.py` , implement **one** block with:
  - `1x1` conv: `B → H`
  - **depthwise** conv: `H → H` , `kernel=P` , `dilation=d` , `groups=H`
  - `1x1` convs for **residual** (`H → B`) and **skip** (`H → B`)
  - Activation: PReLU after each conv
  - **Causal padding**: pad **left** by `d*(P-1)` ; no right pad.
- In a tiny test, feed `(B,B,T')` through the block:
  - Assert output shapes match (`res` same as input channels/time, `skip` same as B/T).
  - Check it runs for dilations 1,2,4,8.

Pass criteria: No shape mismatches; causal padding produces **no time shift** (i.e., input and output lengths identical).

Tip: keep weights small (Kaiming normal); you're not training yet.

Milestone 5 — Separator skeleton (no training yet) (45–60 min)

Goal: chain blocks into the classic TasNet separator (no need to optimize yet).

Do:

- In `src/models/separator.py` :
  - `bottleneck: 1x1` from `N → B`
  - A list of blocks with dilations `[1,2,4,8,16,32]` per repeat, **R repeats**.
  - Sum all **skip** outputs and run a final `1x1` to produce `N*C` channels (masks).
  - `sigmoid` the masks and **reshape** to `(B, C, N, T')` .

Micro-test:

- Hook up **encoder** → **separator** → **decoder**.
- Input batch `(2,1,16000)` , forward pass, assert output `(2, C, 16000)` after crop/pad.
- **Param count**: write a 3-liner that sums `p.numel()` for `p.requires_grad` .

- Expect ≈ **200k–260k** with th Skip to content ms (N=64, B=64, H=64, P=3, X=6, R=3).

Done when: shapes OK, params in range, forward pass <100 ms on your PC.

Milestone 6 — Toy training loop (15–30 min)

Goal: prove the plumbing works on **synthetic data** before touching real music.

Do:

- Make a tiny dataset: 5,000 snippets of length 1 s:
  - "vocals": a random AM or FM tone + mild noise
  - "accomp": sum of 2–3 random sines at different freqs
  - mixture = sum
- Loss: **negative SI-SDR** on the estimated vocal vs target vocal **+** mixture-consistency (project estimates to sum to mixture; optional).
- Train for 1–2 epochs, batch size 8.

Pass criteria: SI-SDR on toy val improves by ≥ 3 dB vs untrained.

Tip: Don't chase perfection here—just verify gradients flow and losses go down.

Milestone 7 — Real data loader (MUSDB/your stems) (60–90 min)

Goal: clean 16 kHz mono data pipeline with random 2–4 s crops.

Do:

- `src/data/mix_dataset.py` :
  - Given folders of `vocals/` and `accomp/` wavs, load random pairs, random 2–4 s crop, downmix to mono, resample to 16 kHz.
  - Return `(mixture, target_vocal, target_accomp)` .
- Add **small augmentations**: random gain ±6 dB; optional tiny pitch-shift/time-stretch (only if you have those utils handy).

Check: iterate the loader; print per-batch shapes and min/max values.

Milestone 8 — Proper t Skip to content ` QAT yet) (multi-hour run)

Goal: get a baseline float model.

Do:

- Optim: Adam, lr 1e-3; warmup 2k steps; cosine decay.
- Loss: `SI-SDR(vocal) + SI-SDR(accomp)` (weight 1.0 each) **+** `MR-STFT L1` (weight 0.2).
- Train 100k–200k steps; save best on **val SI-SDR (vocal)**.

Pass criteria: On a MUSDB-like val set at 16 kHz, aim for **≥5 dB** vocal SI-SDR. (If you're at ~3–4 dB, it's still okay for first pass.)

Milestone 9 — Streaming harness (with your trained weights) (30–60 min)

Goal: prove **<30 ms** algorithmic latency, glitch-free audio.

Do:

- Use the `FrameCutter/OLA` from Milestone 3.
- For each frame:
  - encode → separate → pick mask c → decode
  - OLA back into a continuous stream.
- Measure:
  - **Algorithmic latency** ≈ `FRAME_MS` (20 ms) + a few ms CPU. Keep `FRAME_MS` ≤ 20 to stay comfy.

Check: listen for frame-edge artifacts. If you hear "zipper" noise, try a short fade on outputs (2–5 ms) before OLA.

Milestone 10 — Quantization-Aware Training (QAT) (1–2 sessions)

Goal: get to **int8** with ≤0.5 dB SI-SDR drop.

Do:

- Swap BN → fused into convs (o… folding).
- Insert fake-quant stubs (per-channel on weights, per-tensor on activations). PyTorch quantization or Brevitas both work.
- Calibrate ranges on 10–20 min of audio.
- Fine-tune 10k–50k steps with a **lower LR** (1e-4).

Pass criteria: SI-SDR drop ≤ 0.5 dB vs float; param bytes ≤ **442 KB** (weights only).

If too big: reduce `H` to 48 or `R` from 3→2; retrain a bit.

Milestone 11 — MAX78000 mapping prep (paper exercise now)

Goal: ensure ops are supported **before** you touch the toolchain.

Do:

- Confirm you only used: Conv1d (→ Conv2d), depthwise/group conv, 1×1 conv, ReLU/PReLU, Sigmoid. No BatchNorm at inference.
- Note tensor shapes per layer for **activations**. Keep peak feature map sizes modest.

Optional rehearsal: Temporarily replace depthwise with group=1 standard conv (same kernel) and check param blow-up. If your toolchain struggles with depthwise, you'll know the fallback cost.

Handy formulas (so you can sanity-check yourself)

- **Conv1d length:** `T_out = floor((T_in + 2*pad − dilation*(K−1) − 1)/stride + 1)`
- **Causal pad (left-only):** `pad_left = dilation*(K−1)`, `pad_right = 0`
- **Param counts:**
  - `1x1 conv (Cin→Cout)`: `Cin*Cout`
  - **Depthwise** conv: `C * K`
  - Transposed conv mirrors the regular conv param count
- **TasNet block (rough):** `B*H (1x1) + H*K (dw) + H*B (res) + H*B (skip)`
  - With `B=H=64, K=3 → 64*64 + 64*3 + 64*64 + 64*64 ≈ 12.5k` per block
  - `18` blocks (X=6, R=3) → ~225k + enc/dec + final 1×1 ≈ **~240k**

How we’ll work together

You implement each milestone, and when you hit something weird (shape off by 1? SNR too low?), paste just the **error message**, the **layer shapes you saw**, and if needed a **tiny snippet** (like 5–10 lines max). I’ll point you exactly where to nudge—no walls of code.

If you want, start with **Milestone 2** (encoder/decoder) now. Once you’ve run the SNR sanity check, tell me:

- your `L`, `N`,
- input/output lengths,
- the SNR you measured on sine and on noise.

Then we’ll lock in the streaming buffer (Milestone 3) and keep stacking from there.