Project Genesis: The Neural Un-Mixer v2.0

Technical Specification for Inverse Signal Chain Estimation

Updated Planning Document

February 2026

1 System Objective

The **Neural Un-Mixer** is an end-to-end MLOps system designed for *Inverse Parameter Estimation*. It decomposes a "Wet" audio sample x into its constituent synthesizer parameters θ_{synth} and effect chain configurations θ_{fx}, such that a re-rendering of these parameters in Ableton Live 12 minimizes the perceptual distance to the original source.

2 Mathematical Framework

The project treats audio reconstruction as a Non-Linear Inverse Problem. We define the forward mapping $G(\theta)$ as the Ableton Signal Chain. Our goal is to find the inverse mapping $F(x) \approx G^{-1}(x)$.

2.1 Multi-Task Architecture

The model employs a shared feature extractor (Encoder) with N specialized heads:

- **Regression Head**: $\hat{\theta}_{cont} \in [4]^d$ for continuous knob positions.
- **Classification Head**: $P(y|x)$ for discrete wavetable and filter indices.
- **Hyperbolic Sequence Decoder**: Unlike standard Euclidean decoders, this module embeds the effect chain $S = (fx_1, fx_2, ..., fx_8)$ into **Hyperbolic space (Poincaré ball)**. This geometry naturally captures the hierarchical and non-commutative nature of signal paths (where order matters), resolving the combinatorial explosion of effect permutations [1, 5].

2.2 Hybrid Loss Function

To ensure both parameter accuracy and auditory fidelity, we optimize:

$$L_{total} = \lambda_p L_{MSE}(\theta, \hat{\theta}) + \lambda_s L_{Spectral}(G_{proxy}(\hat{\theta}), x) + \lambda_r L_{reg}(\hat{\theta}) \tag{1}$$

Where G_{proxy} is a **Differentiable Neural Proxy** (TCN-based) that simulates the non-linearities of the Ableton environment, enabling gradient flow through the "black box" DSP chain [6].

2.3 Inference-Time Finetuning (ITF)

To address the "Proxy Gap" (discrepancy between the neural proxy and the real Ableton engine), we implement an ITF stage during inference. After the initial prediction $\hat{\theta}_{init}$, we freeze the proxy decoder weights ϕ^* and refine the input parameters θ for the specific test sample x_t [2]:

$$\theta_{final} = \text{argmin}_\theta \left(L_{Spectral}(G_{proxy}(\theta), x_t) + \lambda_B L_{regularization} \right) \tag{2}$$

This allows the model to "overfit" the specific audio sample at runtime, significantly reducing reconstruction error [7].

3 Statistical Guardrails & Optimization

Given the high-dimensional and non-convex nature of the parameter space, we implement the following:

1. **Mixture Density Networks (MDN)**: To address *Non-Identifiability* (one-to-many mapping), the model predicts a probability distribution $p(\theta|x) = \sum \pi_i \mathcal{N}(\mu_i, \sigma_i^2)$ rather than a point estimate [8].
2. **GCWD Optimization**: Standard Adam optimizers often fail on Spectral Loss landscapes due to ill-conditioning (gradients ranging from 10^{40} to 10^{-40}) [9]. We utilize **Gradient Clipping with Weight Decay (GCWD)** to prevent floating-point overflow and stabilize convergence [3, 10].
3. **Curriculum Learning**: Sequential training phases: Dry Synth $\rightarrow$ Non-Linear FX $\rightarrow$ Full Chain.

4 Technical Stack

- **Engine**: PyTorch, Torchaudio, AbletonOSC.
- **Optimization**: Geoopt (for Riemannian/Hyperbolic optimization) [11].
- **Data/Ops**: DVC (Versioning), MLflow (Tracking), FastAPI (Inference).
- **Domain**: Ableton Live 12 Native Devices (Wavetable + Big 8 FX).