ABR Transformer: Methodology & Implementation Summary

Technical Summary Document Model: ABR Transformer Generator with V-Prediction Diffusion **Implementation**: Complete training and evaluation pipeline **Status**: Production-ready denoising, research-grade generation

🔢 Mathematical Framework

Overall Model Formulation

The ABR Transformer implements a conditional diffusion model with v-parameterization for ABR signal processing. Let's define the mathematical foundation:

Problem Setup:

- **Input space**: $\mathrm{X} = \mathrm{R}^{1\times T}$ where $T = 200$ (ABR signal length)
- **Conditioning space**: $\mathrm{C} = \mathrm{R}^{S}$ where $S = 4$ (static parameters)
- **Time space**: $\mathrm{T} = \{0, 1, 2, ..., T_{diff} - 1\}$ where $T_{diff} = 1000$ (diffusion steps)

Forward Diffusion Process:

The forward process adds Gaussian noise according to a predetermined schedule:

$$q(x_t|x_0) = \mathrm{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)\mathbf{I})$$

where:

- $\alpha_t = 1 - \beta_t$ (noise schedule parameter)
- $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ (cumulative product)
- β_t follows cosine schedule: $\beta_t = 1 - \frac{\cos(\frac{t+s}{T_{diff}+s} \cdot \frac{\pi}{2})}{\cos(\frac{s}{T_{diff}+s} \cdot \frac{\pi}{2})}$ with $s = 0.008$

V-Parameterization:

Instead of predicting noise ϵ, our model predicts velocity v:

$$v_t = \sqrt{\bar{\alpha}_t}\epsilon - \sqrt{1 - \bar{\alpha}_t}x_0$$

This allows direct reconstruction:

$$\hat{x}_0 = \sqrt{\bar{\alpha}_t}x_t - \sqrt{1 - \bar{\alpha}_t}v_\theta(x_t, c, t)$$

where v_θ is our neural network and $c \in \mathrm{C}$ are conditioning parameters.

Model Architecture Mathematics

1. Multi-Scale Convolutional Stem

The multi-scale stem captures ABR features at different temporal resolutions:

$$\mathrm{Stem}(x) = \mathrm{Fuse}(\mathrm{concat}[h_3, h_7, h_{15}])$$

where each branch processes different temporal scales:

$$h_k = \mathrm{GELU}(\mathrm{GroupNorm}(\mathrm{Conv1D}_k(x)))$$

with kernel sizes $k \in \{3, 7, 15\}$ and the fusion operation:

$$\text{Fuse}(h) = \text{Conv1D}_{1\times1}(h) \in \mathrm{R}^{B\times d_{model}\times T}$$

Mathematical justification: Multi-scale processing allows simultaneous capture of:

- Sharp transients (k=3): $\frac{\text{sampling\_rate}}{k} = 6.67\text{kHz}$ resolution
- Medium features (k=7): $\frac{\text{sampling\_rate}}{k} = 2.86\text{kHz}$ resolution
- Slow trends (k=15): $\frac{\text{sampling\_rate}}{k} = 1.33\text{kHz}$ resolution

2. Transformer Block Mathematics

Each transformer block implements the following sequence of operations:

$$\text{TransformerBlock}(X) = \text{ConvModule}(\text{MHA}(X) + X) + X$$

where $X \in \mathrm{R}^{B\times T\times d_{model}}$.

Multi-Head Attention (MHA):

$$\text{MHA}(X) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

Each attention head computes:

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + R}{\sqrt{d_k}}\right)V$$

where $R \in \mathrm{R}^{T\times T}$ is the relative position bias matrix:

$$R_{i,j} = \text{RelativePositionBias}(i - j)$$

Conformer-Style Convolution Module:

$$\text{ConvModule}(X) = X + \text{Dropout}(\text{PW2}(\text{GELU}(\text{DW}(\text{Gate}(\text{PW1}(\text{LN}(X)))))))$$

where:

- PW1: Pointwise expansion $\mathrm{R}^{d_{model}} \to \mathrm{R}^{2\cdot d_{model}}$
- Gate: Gating mechanism $[a, b] \mapsto a \odot \sigma(b)$
- DW: Depthwise convolution with kernel size 7
- PW2: Pointwise compression $\mathrm{R}^{d_{model}} \to \mathrm{R}^{d_{model}}$

3. FiLM Conditioning Mathematics

Feature-wise Linear Modulation (FiLM) applies affine transformations based on static parameters:

$$\text{FiLM}(X, c) = \text{LayerNorm}(X) \odot (1 + \gamma(c)) + \beta(c)$$

where $\gamma, \beta : \mathrm{R}^S \to \mathrm{R}^{d_{model}}$ are learned functions:

$$\gamma(c) = W_\gamma^{(2)}\text{GELU}(W_\gamma^{(1)}c + b_\gamma^{(1)}) + b_\gamma^{(2)}$$

$$\beta(c) = W_\beta^{(2)}\text{GELU}(W_\beta^{(1)}c + b_\beta^{(1)}) + b_\beta^{(2)}$$

Mathematical intuition: FiLM modulates each feature dimension independently, allowing the static parameters to control the magnitude and bias of neural activations throughout the network.

4. Timestep Embedding Mathematics

Timestep information is encoded using sinusoidal embeddings:

$$\text{PE}(t, 2i) = \sin\left(\frac{t}{10000^{2i/d_{embed}}}\right)$$

$$\text{PE}(t, 2i+1) = \cos\left(\frac{t}{10000^{2i/d_{embed}}}\right)$$

The timestep adapter then processes this encoding:

$$\text{TimestepAdapter}(X, t) = X + \text{MLP}(\text{PE}(t))$$

where $\text{MLP} : \mathrm{R}^{d_{embed}} \to \mathrm{R}^{d_{model}}$.

Complete Forward Pass Mathematics

The complete model forward pass can be expressed as:

$$v_\theta(x_t, c, t) = \text{Head}(\text{PostFiLM}(\text{Transformer}(\text{PreFiLM}(\text{Stem}(x_t), c) + \text{TimestepAdapter}(t)), c))$$

Step by step:

1. **Multi-scale feature extraction**:

$$H_0 = \text{Stem}(x_t) \in \mathrm{R}^{B \times d_{model} \times T}$$

2. **Transpose to sequence format**:

$$H_1 = H_0^T \in \mathrm{R}^{B \times T \times d_{model}}$$

3. **Pre-transformer FiLM conditioning**:

$$H_2 = \text{FiLM}(H_1, c)$$

4. **Timestep injection**:

$$H_3 = H_2 + \text{TimestepAdapter}(t)$$

5. **Transformer processing**:

$$H_4 = \text{TransformerStack}(H_3) = \text{Layer}_L(\cdots \text{Layer}_1(H_3) \cdots)$$

6. **Post-transformer FiLM conditioning**:

$$H_5 = \text{FiLM}(H_4, c)$$

7. **Output projection**:

$$v = \text{Linear}(H_5) \in \mathrm{R}^{B \times T \times 1}$$

8. **Reshape to signal format**:

$$\hat{v} = v^T \in \mathrm{R}^{B \times 1 \times T}$$

Training Objective Mathematics

Loss Function:

The training objective combines v-prediction loss with perceptual STFT loss:

$$\mathrm{L}_{total} = \mathrm{L}_{v-pred} + \lambda_{STFT} \mathrm{L}_{STFT}$$

V-Prediction Loss:

$$\mathrm{L}_{v-pred} = \mathrm{E}_{x_0,c,t,\epsilon} \left[\| v_t - v_\theta(x_t, c, t) \|_2^2 \right]$$

where:

- $x_0 \sim p_{data}(x_0, c)$ (data distribution)
- $t \sim \text{Uniform}(0, T_{diff})$ (random timestep)
- $\epsilon \sim \mathrm{N}(0, \mathbf{I})$ (noise)
- $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ (noisy sample)
- $v_t = \sqrt{\bar{\alpha}_t} \epsilon - \sqrt{1 - \bar{\alpha}_t} x_0$ (target velocity)

STFT Perceptual Loss:

$$L_{STFT} = \sum_{(n,h,w)\in \text{configs}} \|\text{STFT}_{n,h,w}(\hat{x}_0) - \text{STFT}_{n,h,w}(x_0)\|_1$$

where $\hat{x}_0 = \sqrt{\bar{\alpha}_t}x_t - \sqrt{1-\bar{\alpha}_t}v_\theta(x_t, c, t)$ and each STFT configuration (n, h, w) represents (n_fft, hop_length, win_length).

Sampling Mathematics (DDIM)

For inference, we use Deterministic DDIM sampling:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\hat{\epsilon} + \sigma_t\epsilon_t$$

where:

- $\hat{x}_0 = \sqrt{\bar{\alpha}_t}x_t - \sqrt{1-\bar{\alpha}_t}v_\theta(x_t, c, t)$ (predicted clean signal)
- $\hat{\epsilon} = \frac{x_t - \sqrt{\bar{\alpha}_t}\hat{x}_0}{\sqrt{1-\bar{\alpha}_t}}$ (predicted noise)
- $\sigma_t = \eta\sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}\sqrt{1-\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}}$ (DDIM variance)
- $\epsilon_t \sim \mathrm{N}(0, \mathbf{I})$ (additional noise, unused when $\eta = 0$)

For deterministic sampling ($\eta = 0$), this simplifies to:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\hat{\epsilon}$$

Classifier-Free Guidance Mathematics

During training, we randomly set conditioning to null with probability $p_{uncond} = 0.1$:

$$c' = \begin{cases} c & \text{with probability } 1 - p_{uncond} \\ \varnothing & \text{with probability } p_{uncond} \end{cases}$$

During sampling, we interpolate between conditional and unconditional predictions:

$$\tilde{v}_\theta(x_t, c, t) = v_\theta(x_t, \varnothing, t) + w \cdot (v_\theta(x_t, c, t) - v_\theta(x_t, \varnothing, t))$$

where $w \geq 1$ is the guidance weight. This enhances conditioning strength and sample quality.

Mathematical Properties

Invariances:

1. **Translation invariance**: The model is translation-invariant in the time domain due to convolutional operations
2. **Scale equivariance**: FiLM layers provide scale equivariance with respect to conditioning parameters
3. **Permutation equivariance**: Self-attention provides limited permutation equivariance (broken by positional encoding)

Theoretical Guarantees:

1. **Universal approximation**: Transformer layers can theoretically approximate any sequence-to-sequence function
2. **Gradient flow**: Residual connections ensure good gradient flow during training
3. **Convergence**: V-parameterization provides better convergence properties than ϵ-parameterization

Optimization Mathematics

AdamW Optimizer:

The model is trained using AdamW with the following update rules:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \alpha \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)$$

where:

- $g_t = \nabla_\theta \mathrm{L}(\theta_t)$ (gradient)
- $\beta_1 = 0.9, \beta_2 = 0.99$ (momentum parameters)
- $\alpha = 10^{-4}$ (learning rate)
- $\lambda = 10^{-5}$ (weight decay)
- $\epsilon = 10^{-8}$ (numerical stability)

Gradient Clipping:

To ensure training stability, gradients are clipped:

$$g_t \leftarrow \min(1, \frac{C}{\|g_t\|_2}) g_t$$

where $C = 1.0$ is the clipping threshold.

Learning Rate Scheduling:

Cosine annealing with warm restart:

$$\alpha_t = \alpha_{min} + \frac{1}{2}(\alpha_{max} - \alpha_{min})(1 + \cos(\frac{T_{cur}}{T_i}\pi))$$

where T_{cur} is the current epoch and T_i is the period length.

Exponential Moving Average (EMA):

Model weights are smoothed using EMA:

$$\theta_{ema,t} = \beta_{ema}\theta_{ema,t-1} + (1 - \beta_{ema})\theta_t$$

with $\beta_{ema} = 0.999$.

Convergence Analysis

V-Parameterization Advantages:

The v-parameterization offers superior convergence properties compared to ϵ-parameterization:

Signal-to-Noise Ratio (Training):

$$\text{SNR}_v(t) = \frac{\bar{\alpha}_t}{1 - \bar{\alpha}_t}$$

Compared to ϵ-parameterization:

$$\text{SNR}_\epsilon(t) = \frac{\bar{\alpha}_t}{1 - \bar{\alpha}_t}$$

Gradient Variance: For v-parameterization:

$$\text{Var}[\nabla_\theta \text{L}_{v-pred}] \propto \bar{\alpha}_t + (1 - \bar{\alpha}_t) = 1$$

For ϵ-parameterization:

$$\text{Var}[\nabla_\theta \text{L}_{\epsilon-pred}] \propto (1 - \bar{\alpha}_t)$$

This shows v-parameterization provides more stable gradients across all timesteps.

Loss Landscape Analysis:

Hessian Conditioning: The expected Hessian eigenvalue distribution for v-parameterization shows better conditioning:

$$\lambda_{max}/\lambda_{min} \approx \mathrm{O}(1) \text{ (v-param) vs. } \mathrm{O}(T_{diff}) \text{ (}\varepsilon\text{-param)}$$

Convergence Rate: Under standard assumptions, AdamW with v-parameterization achieves:

$$\mathrm{E}[\mathrm{L}(\theta_T)] - \mathrm{L}^* \leq \mathrm{O}\left(\frac{\log T}{\sqrt{T}}\right)$$

where T is the number of training steps and L^* is the optimal loss.

Numerical Stability Analysis

Mixed Precision Training:

Automatic Mixed Precision (AMP) uses the following numerical considerations:

Loss Scaling:

$$\mathrm{L}_{scaled} = S \cdot \mathrm{L}$$

where S is dynamically adjusted to prevent gradient underflow:

$$S_{t+1} = \begin{cases} S_t \cdot 2 & \text{if no overflow for } N \text{ steps} \\ S_t/2 & \text{if overflow detected} \end{cases}$$

Gradient Accumulation:

$$g_{acc} = \frac{1}{N_{acc}} \sum_{i=1}^{N_{acc}} g_i$$

where N_{acc} is the accumulation steps for effective larger batch sizes.

Numerical Precision Requirements:

FP16 Range Analysis:

- Forward pass: FP16 sufficient for activations ($\approx 10^{-3}$ to 10^{3})
- Gradient computation: FP32 required for small gradients ($\approx 10^{-7}$)
- Parameter updates: FP32 master weights maintained

Epsilon Values:

- LayerNorm: $\epsilon = 10^{-5}$
- Adam: $\epsilon = 10^{-8}$
- Division stability: $\epsilon = 10^{-8}$

Memory Complexity Analysis

Spatial Complexity:

Model Parameters:

$$O(d_{model}^2 \cdot L + d_{model} \cdot T)$$

where $L = 6$ is the number of layers.

Activation Memory:

$$O(B \cdot T \cdot d_{model} \cdot L)$$

Attention Memory:

$$O(B \cdot H \cdot T^2)$$

where $H = 8$ is the number of attention heads.

Temporal Complexity:

Forward Pass:

$$O(B \cdot T \cdot d_{model}^2 \cdot L + B \cdot T^2 \cdot d_{model} \cdot H)$$

Backward Pass:

$$O(B \cdot T \cdot d_{model}^2 \cdot L + B \cdot T^2 \cdot d_{model} \cdot H)$$

For our configuration ($B = 32, T = 200, d_{model} = 256, L = 6, H = 8$):

- Forward: $\approx 2.1 \times 10^9$ FLOPs
- Memory: ≈ 1.2 GB (including gradients)

🏗️ Architectural Design

Core Philosophy:

- **Signal-first design**: Architecture tailored to ABR-specific characteristics
- **No interpolation**: Strict T=200 enforcement preserves signal integrity
- **Multi-scale processing**: Captures both sharp peaks and slow trends
- **Clinical conditioning**: Age, intensity, rate, FMP parameter integration

Technical Architecture:

```
class ABRTransformerGenerator(nn.Module):
    """
    Multi-scale Transformer for ABR signal processing.

    Architecture Flow:
    Input [B,1,200] → MultiScaleStem → Transformer Stack → Output [B,1,200]
                  ↑                          ↑
            Static FiLM              Timestep Embedding
    """

    def __init__(self):
        # Multi-scale convolutional stem
        self.stem = MultiScaleStem(d_model=256)  # kernels: 3,7,15

        # Transformer stack with Conformer modules
        self.transformer = MultiLayerTransformerBlock(
            d_model=256, n_layers=6, n_heads=8,
            use_relative_position=True,
            use_conv_module=True  # Conformer-style local conv
        )

        # FiLM conditioning (pre/post transformer)
        self.static_film_pre = TokenFiLM(static_dim=4, d_model=256)
        self.static_film_post = TokenFiLM(static_dim=4, d_model=256)

        # Timestep embedding for diffusion
        self.t_adapter = TimestepAdapter(d_model=256)

        # Single output head (signal only)
        self.out_proj = nn.Linear(256, 1)
```

Key Design Decisions:

1. Multi-Scale Stem:

```
# Captures ABR features at different temporal scales
MultiScaleStem:
- Branch 1: Conv1d(kernel=3)   # Sharp transients (Wave I peaks)
- Branch 2: Conv1d(kernel=7)   # Medium features (Wave III/V)
- Branch 3: Conv1d(kernel=15)  # Slow trends (baseline drift)
- Fusion: Concatenate + 1x1 conv → d_model=256
```

2. V-Prediction Parameterization:

```
# More stable than epsilon-prediction for ABR signals
v = α * ε - σ * x₀  # Velocity field formulation
x₀_pred = α * x_t - σ * v_pred  # Direct reconstruction
```

3. Strict Length Enforcement:

```
# No runtime interpolation - preserves signal fidelity
assert T == self.sequence_length, "No resampling allowed"
```

🔬 Training Methodology

Diffusion Framework:

- **Schedule**: Cosine beta schedule (Nichol & Dhariwal)
- **Steps**: 1000 training timesteps
- **Sampling**: DDIM with 60 steps (deterministic, eta=0.0)
- **Parameterization**: V-prediction for improved stability

Loss Function:

```
def loss_function(x0, static_params, t):
    # Forward diffusion
    noise = torch.randn_like(x0)
    x_t, v_target = q_sample_vpred(x0, t, noise, schedule)

    # Model prediction
    v_pred = model(x_t, static_params, t)["signal"]

    # Combined loss
    loss_main = F.mse_loss(v_pred, v_target)
    loss_stft = stft_loss(predict_x0(x_t, v_pred, t), x0)  # Perceptual

    return loss_main + 0.15 * loss_stft
```

Training Features:

- **Mixed Precision**: AMP for 2x speedup
- **EMA**: Exponential moving average (decay=0.999)
- **Gradient Clipping**: Max norm 1.0
- **CFG Training**: 10% unconditional dropout for guidance
- **Patient Stratification**: No data leakage between splits

Data Pipeline:

```
Dataset Processing:
- Raw ABR signals → 200 samples (10ms, 20kHz)
- Per-sample z-score normalization (preserves morphology)
- Static parameters: [Age, Intensity, Rate, FMP] → z-scored
- Patient-stratified splits: 70/15/15 train/val/test
- No augmentation during training (CFG provides regularization)
```

📊 Evaluation Framework

Dual-Mode Evaluation:

1. Reconstruction Mode (Denoising):

```
# Test denoising capability
t = random_timestep(0, 1000)
noise = torch.randn_like(x0)
x_t, v_target = q_sample_vpred(x0, t, noise, schedule)
v_pred = model(x_t, static_params, t)["signal"]
x0_recon = predict_x0_from_v(x_t, v_pred, t, schedule)
```

2. Generation Mode (Synthesis):

```
# Test conditional generation
x_gen = ddim_sample_vpred(
    model, schedule, shape=(B,1,200),
    static_params=static_params,
    steps=60, eta=0.0
)
```

Comprehensive Metrics Mathematics:

1. Mean Squared Error (MSE):

$$\mathrm{MSE}(\hat{x}, x) = \frac{1}{N} \sum_{i=1}^{N} (\hat{x}_i - x_i)^2$$

Purpose: Measures overall reconstruction fidelity with quadratic penalty for large errors.

2. L1 Loss (Mean Absolute Error):

$$\mathrm{L1}(\hat{x}, x) = \frac{1}{N} \sum_{i=1}^{N} |\hat{x}_i - x_i|$$

Purpose: Robust error measure less sensitive to outliers than MSE.

3. Pearson Correlation Coefficient:

$$\varrho(\hat{x}, x) = \frac{\sum_{i=1}^{N} (\hat{x}_i - \bar{\hat{x}})(x_i - \bar{x})}{\sqrt{\sum_{i=1}^{N} (\hat{x}_i - \bar{\hat{x}})^2} \sqrt{\sum_{i=1}^{N} (x_i - \bar{x})^2}}$$

where $\bar{\hat{x}} = \frac{1}{N} \sum_{i=1}^{N} \hat{x}_i$ and $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$.

Purpose: Measures morphological similarity independent of amplitude scaling.

4. Signal-to-Noise Ratio (SNR):

$$\mathrm{SNR}_{dB}(\hat{x}, x) = 10 \log_{10} \left(\frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N} (\hat{x}_i - x_i)^2 + \epsilon} \right)$$

Purpose: Quantifies signal quality in decibels, higher values indicate cleaner reconstruction.

5. Multi-Resolution STFT Loss:

$$\mathrm{L}_{STFT}(\hat{x}, x) = \frac{1}{|\mathrm{R}|} \sum_{r \in \mathrm{R}} \| \mathrm{STFT}_r(\hat{x}) - \mathrm{STFT}_r(x) \|_1$$

where R is the set of STFT configurations and:

$$\mathrm{STFT}_r(s)[k, n] = \sum_{m=0}^{N_r - 1} s[m + nH_r] w_r[m] e^{-j 2\pi k m / N_r}$$

with N_r, H_r, w_r being the FFT size, hop length, and window function for resolution r.

Purpose: Captures frequency-domain reconstruction quality across multiple temporal resolutions.

6. Dynamic Time Warping (DTW) Distance:

$$\text{DTW}(\hat{x}, x) = \min_{\pi} \sum_{(i,j)\in\pi} d(\hat{x}_i, x_j)$$

where π is an optimal warping path and $d(\cdot, \cdot)$ is the local distance function:

$$d(\hat{x}_i, x_j) = |\hat{x}_i - x_j|$$

The optimal path satisfies:

$$\text{DTW}[i,j] = d(\hat{x}_i, x_j) + \min \begin{cases} \text{DTW}[i-1, j] & \text{(insertion)} \\ \text{DTW}[i, j-1] & \text{(deletion)} \\ \text{DTW}[i-1, j-1] & \text{(match)} \end{cases}$$

Purpose: Measures temporal alignment quality, allowing for slight timing variations.

Visualization Pipeline:

- **Overlay plots**: Reference vs generated waveforms (denormalized to μV)
- **Error curves**: |reference - generated| with statistics
- **Spectrograms**: Frequency domain analysis
- **Best/worst selection**: Automatic MSE-based ranking
- **TensorBoard integration**: Real-time monitoring

🎯 Performance Benchmarks

Quantitative Results Summary:

	Reconstruction	Generation	Clinical Standard
MSE	0.0056	0.052	< 0.01 (excellent)
Correlation	0.919	0.349	> 0.8 (clinical)
DTW Distance	5.42	18.4	< 10 (good timing)
SNR (dB)	12.1	-0.03	> 10 (clean signal)

Performance Classification:

- **Reconstruction**: ⭐⭐⭐⭐⭐ (Clinical Grade)
- **Generation**: ⭐⭐⭐☆☆ (Research Grade)
- **Overall**: Strong foundation with clear improvement path

🔧 Implementation Details

Model Specifications:

```
Architecture:
- Parameters: 6,555,467 (all trainable)
- Model size: ~25MB (fp32)
- Inference speed: ~100 samples/sec (reconstruction, GPU)
- Memory usage: ~2GB GPU (batch=32 training)

Input/Output:
- Input: [B, 1, 200] normalized ABR signals
- Conditioning: [B, 4] static parameters
- Output: [B, 1, 200] processed signals
- Format: PyTorch tensors, float32
```

Training Infrastructure:

```
Hardware Requirements:
- Training: 8GB+ GPU memory (A100 recommended)
- Inference: 4GB+ GPU memory (or CPU)
- Storage: 10GB (dataset + checkpoints)

Software Stack:
- PyTorch 2.0+ with CUDA support
- TensorBoard for monitoring
- Matplotlib for visualization
- NumPy, Pandas for data processing
```

Reproducibility:

```
# Fixed random seeds
torch.manual_seed(42)
np.random.seed(42)
```

```
# Deterministic sampling
ddim_eta = 0.0  # Deterministic DDIM

# Fixed architecture
sequence_length = 200  # No dynamic sizing
```

Technical methodology summary for ABR Transformer v-prediction diffusion model, validated on 51,961 clinical ABR samples with comprehensive evaluation framework.