NEURAL NETWORKS FOR PHYSICAL PARAMETER ESTIMATION OF PLATE REVERBERATION FROM IMPULSE RESPONSES

Jia-Chang Yang

Independent Researcher
Taiwan, ROC
random1473283696@gmail.com

ABSTRACT

This paper presents our Task A submission to the 1st DAFx Parameter Estimation Challenge. We use the official ModalPlate dataset generator to synthesize 1000 one-second plate impulse responses with randomly sampled parameters inside the public ranges. A time-domain CNN-GRU regressor then estimates the six official Task A parameters from each unnormalised waveform. The model combines three one-dimensional convolutional blocks with a bidirectional gated recurrent unit and is trained with mean squared error on min-max normalised targets. The generated data are split into $700/150/150$ train/validation/test examples, and the test split is never used during training or model selection. The implementation follows the official Task A format and exports evaluation-compatible prediction files for both development evaluation and blind-set submission.

1. INTRODUCTION

Task A of the DAFx challenge asks participants to recover a compact physical description of a plate reverb from measured impulse responses [1]. Plate reverberation is a canonical physically grounded audio effect, and the challenge adopts a simulation framework closely connected to classical plate-reverb modelling [2]. Deep learning is now a practical tool for black-box audio effect modelling, including end-to-end raw-audio models for nonlinear processors and controllable amplifier emulation [3, 4, 5]. Because the development data are synthetic, temporally aligned, and noise-free, we adopt a direct waveform regressor and preserve the absolute amplitude of the official `.npz` responses.

2. MOTIVATION

We choose a time-domain representation for three reasons. First, the challenge input is itself an impulse response, so the excitation onset, phase alignment, modal beating, and long decay are all directly encoded in the waveform. A spectrogram would introduce additional design choices such as window size, hop size, and magnitude compression, which may blur physically informative details before learning begins. For Task A, the waveform is therefore a direct representation for estimating parameters that affect the modal structure, amplitude scale, plate geometry, and output position.

Prior work [6] has investigated neural impulse response modelling for audio effects, including real time guitar cabinet emulation from microphone and placement parameters. This line of work shows that learned models can represent impulse response variations in audio effect systems without requiring explicit storage of all measured IRs. Although our task is physical parameter estimation rather than IR generation, it motivates the use of neural models for compact representations of impulse response data. In Task A, we therefore operate directly on the unnormalised displacement IR and learn a mapping from the waveform to the official physical parameters.

3. METHOD

This section describes the proposed supervised estimator for Task A. Each input is an official unnormalised displacement IR stored in a `.npz` container, kept at $44.1\,\mathrm{kHz}$ without waveform normalisation. We operate in the time domain because the waveform preserves the physical amplitude scale and avoids a hand designed spectral representation [3, 4]. Motivated by neural time domain audio effect modelling [5] and neural impulse response modelling for audio effects [6], the estimator combines a convolutional encoder with a two layer bidirectional GRU. The convolutional encoder extracts local temporal patterns, while the GRU aggregates longer modal dependencies. The resulting sequence representation is temporally averaged and mapped by a fully connected regression head to the six normalised Task A parameters. Section 3.1 defines the CNN-GRU estimator, and Section 3.2 describes the training objective.

3.1. Time Domain CNN-GRU Estimator

The proposed system estimates the official Task A physical parameters directly from the unnormalised displacement impulse response. Each input is loaded from the official `.npz` file and kept at the native sampling rate of $44.1\,\mathrm{kHz}$. Let $\mathbf{x} \in \mathbb{R}^T$ denote a one-second input impulse response, where $T = 44100$ is the number of audio samples. No waveform normalisation is applied, so the absolute displacement scale is preserved.

Following the official Task A definition, the network predicts the six physical parameters

$$\mathbf{y} = [\mu, D/\mu, T_0/\mu, L_y, x_o, y_o]^\top. \tag{1}$$

Here, $\mu = \rho h$ is the density–thickness parameter used by the official challenge, D/μ is the rigidity ratio, T_0/μ is the tension ratio, L_y is the plate height, and (x_o, y_o) denotes the output position. This follows the official challenge parameterisation and avoids directly estimating the non unique raw material parameters.

The estimator is composed of a one dimensional convolutional encoder, a bidirectional recurrent module, and a fully connected regression head. The convolutional encoder maps the waveform to a lower rate temporal feature sequence:

$$\mathbf{H} = f_{\mathrm{CNN}}(\mathbf{x}), \qquad \mathbf{H} \in \mathbb{R}^{T' \times C}, \tag{2}$$

where T' is the sequence length after convolutional pooling and C is the number of output channels. In the reported implementation, f_{CNN} consists of three Conv1d blocks with batch normalisation, ReLU activation, max pooling, and dropout.

The temporal feature sequence is then processed by a bidirectional GRU:

$$\mathbf{Z} = f_{\mathrm{BiGRU}}(\mathbf{H}), \qquad \mathbf{Z} \in \mathbb{R}^{T' \times 2h}, \tag{3}$$

where h is the hidden size of each recurrent direction. The factor $2h$ results from concatenating the forward and backward recurrent states.

A fixed dimensional representation is obtained by temporal mean pooling:

$$\mathbf{g} = \frac{1}{T'} \sum_{t=1}^{T'} \mathbf{Z}_t, \qquad \mathbf{g} \in \mathbb{R}^{2h}, \tag{4}$$

where $\mathbf{Z}_t$ denotes the bidirectional GRU output at time index t. The pooled representation is then passed to a fully connected regression head:

$$\hat{\mathbf{y}}_{\mathrm{norm}} = \sigma\left(f_{\mathrm{FC}}(\mathbf{g})\right), \qquad \hat{\mathbf{y}}_{\mathrm{norm}} \in [0,1]^{N_S}, \tag{5}$$

where $N_S = 6$, $f_{\mathrm{FC}}(\cdot)$ denotes the fully connected regression head, and $\sigma(\cdot)$ is the sigmoid function. The network therefore outputs range normalised estimates of the six Task A parameters in Eq. (1).

3.2. Loss Function

All six physical parameters are range normalised using the public Task A lower and upper bounds. For the ith quantity, the normalised target is

$$y_{i,\mathrm{norm}} = \frac{y_i - y_i^{\min}}{y_i^{\max} - y_i^{\min}}, \qquad i = 1, \ldots, N_S. \tag{6}$$

The inverse transformation used at inference time is

$$\hat{y}_i = \hat{y}_{i,\mathrm{norm}} \left(y_i^{\max} - y_i^{\min}\right) + y_i^{\min}. \tag{7}$$

The model is trained with mean squared error on the normalised targets. For a mini batch of B impulse responses, the loss is

$$\mathcal{L}_{\mathrm{NMSE}} = \frac{1}{B N_S} \sum_{b=1}^{B} \sum_{i=1}^{N_S} \left(\hat{y}_{b,i,\mathrm{norm}} - y_{b,i,\mathrm{norm}}\right)^2. \tag{8}$$

This objective follows the implementation, where both predictions and reference parameters are represented in the normalised Task A range during optimisation. The predicted parameters are converted back to the physical scale using Eq. (7) before evaluation and export.

4. EXPERIMENTAL SETUP

4.1. Datasets

We generate the development set with the official ModalPlate simulator provided with the challenge. The dataset contains 1000 one-second displacement impulse responses at 44.1 kHz, each paired with a parameter CSV file. The non-fixed plate parameters are sampled within the public Task A ranges, while the fixed constants specified by the challenge remain unchanged.

Following the official Task A definition, each full parameter vector is converted to the six identifiable parameters: the areal density $\mu = \rho h$, the rigidity-to-mass ratio D/μ, the tension-to-mass ratio T_0/μ, the plate length L_y, and the output position (x_o, y_o). Here, D denotes the flexural rigidity of the plate. This target representation is used because the raw material parameters (ρ, h, E, T_0) are not uniquely identifiable from the impulse response, whereas the three invariants μ, D/μ, and T_0/μ define the identifiable Task A parameterisation. In the implementation and submission files, these six parameters correspond to the official columns `mu`, `D_mu`, `T0_mu`, `Ly`, `op_x`, and `op_y`.

The 1000 examples are split randomly into $700/150/150$ training, validation, and test examples using seed 42. The test split is not used during training or model selection.

4.2. Training and Evaluation Protocol

All six Task A targets are min–max normalised using the public lower and upper bounds specified by the challenge, rather than statistics estimated from the training split. This keeps the regression target space consistent with the official NMSE evaluation. The model is trained to minimise mean squared error on the normalised targets using AdamW with learning rate 10^{-3}, weight decay 10^{-2}, and batch size 16. Training is run for up to 150 epochs with ReduceLROnPlateau scheduling, validation-loss-based checkpoint selection, early stopping, gradient clipping, and deterministic random seeds.

The training split is used for gradient updates, the validation split for checkpoint selection and scheduler control, and the held-out test split is used only for post-training evaluation. After training, the checkpoint with the lowest validation loss is evaluated on the 150-example development test split. The prediction script exports the estimated Task A parameters in the official six-column format and produces evaluation-compatible files. The same export pipeline is used for the blind challenge set, ensuring that the development evaluation and final submission follow the same file format and preprocessing assumptions. Table 1 summarises the reported configuration.

5. EXPERIMENTAL RESULTS

5.1. Parameter Estimation Performance

We first assess the accuracy of the estimated physical parameters on the development test split, which is not used during training or model selection. Performance is measured using the parameter normalised mean squared error (NMSE) over the six official Task A parameters, where lower values indicate more accurate parameter estimates. Table 2 reports the performance of the proposed CNN-GRU estimator and the official Particle Swarm Optimisation (PSO) baseline on the same 150 examples. The CNN-GRU model obtains a mean parameter NMSE of 0.0499, compared with

Table 1: Settings used in the reported run.

Component	Setting
Data generation	Official `DatasetGen.py`; 1000 random IRs in `random-IR-1000-1.0s`; 1.0 s duration at 44.1 kHz
Split / targets	Random 700/150/150 train/validation/test split with seed 42; targets $[\mu, D_\mu, T0_\mu, L_y, op_x, op_y]$ scaled by public min-max ranges
Input	Official unnormalised `.npz` waveform, fixed length 44,100 samples, waveform normalisation set to `none`
CNN front end	Conv1d blocks $1 \to 64$ ($k = 9$), $64 \to 128$ ($k = 9$), and $128 \to 128$ ($k = 5$), each with batch normalisation, ReLU, max-pooling by 4, and dropout 0.3
Sequence model	Two-layer bidirectional GRU with hidden size 128
Regression head	Temporal mean pooling, fully connected layer of size 256, dropout 0.3, fully connected output layer of size 6, sigmoid-bounded outputs
Loss / optimiser	MSE loss with AdamW, learning rate 10^{-3}, weight decay 10^{-2}, and batch size 16
Scheduling	Up to 150 epochs, ReduceLROnPlateau factor 0.5, plateau patience 8, early stopping patience 20, gradient clipping 1.0
Hardware / eval	One NVIDIA RTX A6000 GPU; evaluation on the unseen 150-example test split

0.0618 for the PSO baseline. This indicates that a direct supervised estimator can estimate the identifiable Task A parameters effectively on unseen examples, while avoiding iterative optimisation during inference.

Table 2: Parameter NMSE comparison between the official PSO baseline and the proposed CNN-GRU estimator on the development test split of 150 examples. Values are reported as mean ± standard deviation.

Method	Parameter NMSE ↓
PSO baseline	0.0618 ± 0.0450
Ours	$\mathbf{0.0499 \pm 0.0249}$

5.2. Physical Parameter Error Analysis

To further examine the source of the aggregate parameter NMSE improvement, we analyse the error of each estimated physical quantity separately on the same development test split of 150 examples. This split is not used during training or model selection. Table 3 reports the mean absolute error and range normalised mean absolute error for the six official Task A parameters. The proposed CNN-GRU estimator gives lower errors than the PSO baseline for five of the six parameters: D/μ, T_0/μ, L_y, x_o, and y_o. The only exception is the areal density $\mu = \rho h$, for which the PSO baseline obtains a slightly lower error. This analysis by physical quantity indicates that the improvement is not dominated by a single parameter, but is distributed across most of the estimated parameters.

Table 3: Physical parameter error comparison between the official PSO baseline and the proposed CNN-GRU estimator on the development test split of 150 examples. Values are reported as mean absolute error and range normalised mean absolute error.

	PSO baseline		Ours	
Parameter	MAE	Norm. MAE	MAE	Norm. MAE
$\mu = \rho h$	**16.31**	**0.162**	18.11	0.175
D/μ	13.44	0.087	**13.12**	**0.065**
T_0/μ	18.85	0.066	**18.20**	**0.044**
L_y	0.717	0.247	**0.711**	**0.245**
x_o	0.129	0.263	**0.113**	**0.232**
y_o	0.130	0.266	**0.124**	**0.253**

5.3. Inference Efficiency

We evaluate inference efficiency to characterize the computational difference between iterative optimization and direct regression. The PSO baseline optimizes a loss surface separately for each input impulse response, whereas the CNN-GRU estimator predicts the six Task A parameters with a single forward pass. As shown in Table 4, the CNN-GRU estimator reduces the average inference time from 36.46 s to 1.10 s per impulse response, requiring only 3.0% of the runtime of the PSO baseline.

Table 4: Average inference time per impulse response on the generated development set of 1000 examples.

Method	Average time
PSO baseline	36.46 s
Ours	**1.10** s

5.4. Training and Validation Loss

We analyse the training and validation losses to justify the checkpoint used for evaluation. Figure 1 shows the MSE loss curves computed on the normalised Task A parameters. The validation loss reaches its minimum at epoch 5, where the training and validation losses are 0.0510 and 0.0525, respectively. Training continues until epoch 25 before early stopping is triggered, while the validation loss remains close to its minimum and reaches 0.0532 at the final epoch. The small separation between the training and validation losses indicates that the selected checkpoint is not associated with severe overfitting. We therefore use the epoch 5 checkpoint for all reported development test and blind set predictions.

5.5. Architecture Ablation Study

We evaluate the contribution of the convolutional front end and the recurrent temporal model by removing one component at a time. All variants are trained and evaluated using the same data split and training protocol, and performance is measured on the development test split of 150 examples. As shown in Table 5, the complete model obtains the lowest parameter NMSE. Removing the GRU increases the error to 0.0550, indicating that recurrent temporal modelling is important for the physical parameter regression

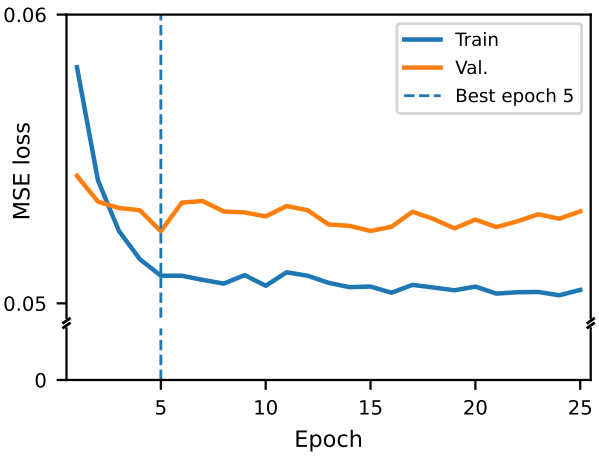

Figure 1: Training and validation MSE losses on the normalised Task A parameters. The broken vertical axis highlights the convergence region around 0.05 while retaining the zero baseline. The validation loss reaches its minimum at epoch 5, which is selected as the final checkpoint.

task. Removing the CNN front end gives a smaller degradation, suggesting that, in this configuration, the recurrent component accounts for most of the observed performance, while the CNN front end provides a modest additional improvement.

Table 5: Architecture ablation on the development test split of 150 examples. Values are reported as mean ± standard deviation.

Variant	Parameter NMSE ↓
w/o GRU	0.0550 ± 0.0294
w/o CNN	0.0499 ± 0.0249
Ours	$\mathbf{0.0497 \pm 0.0253}$

6. CONCLUSION

This paper presented our Task A submission to the 1st DAFx Parameter Estimation Challenge. The proposed system uses a time-domain CNN-GRU regressor trained on official ModalPlate development data to estimate the six identifiable Task A parameters directly from unnormalised displacement impulse responses. By preserving the original time domain response and its absolute amplitude scale, the model has access to onset structure, phase relationships, modal interactions, and decay behaviour without relying on handcrafted spectral preprocessing. The full pipeline, including data splitting, target normalisation, checkpoint selection, and prediction export, is designed to remain compatible with the official challenge evaluation format and reproducible from the submitted code.

7. REFERENCES

[1] M. Ducceschi and L. Gabrielli, "The 1st DAFx parameter estimation challenge: A benchmark for plate reverb system identification," 2026, challenge documentation.

[2] S. Bilbao, "A digital plate reverberation algorithm," *J. Audio Eng. Soc.*, vol. 55, pp. 135–144, Mar. 2007.

[3] M. A. M. Ramírez, E. Benetos, and J. D. Reiss, "Deep learning for black-box modeling of audio effects," *Applied Sciences*, vol. 10, no. 2, p. 638, 2020.

[4] A. Wright, E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time guitar amplifier emulation with deep learning," *Applied Sciences*, vol. 10, no. 3, p. 766, 2020.

[5] L. Juvela, E.-P. Damskägg, A. Peussa, J. Mäkinen, T. Sherson, S. I. Mimilakis, K. Rauhanen, and A. Gotsopoulos, "End-to-end amp modeling: From data to controllable guitar amplifier models," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2023, pp. 1–5.

[6] T. Sinjanakhom and S. Chivapreecha, "Neural networks for real-time digital emulation of guitar speaker cabinet impulse response," in *Proc. Int. Conf. on Knowledge and Smart Technology (KST)*. IEEE, Jan. 2022, pp. 131–136.