CLOCK-DRIFT-CONSTRAINED STRUCTURED REGRESSION FOR MULTI-CHANNEL BIOACOUSTIC AUDIO ALIGNMENT

Technical Report

Ragib Amin Nihal, Benjamin Yen, Takeshi Ashizawa, Kazuhiro Nakadai

Institute of Science Tokyo
Department of Systems and Control Engineering
Tokyo, Japan

1. ABSTRACT

We tackle multi-channel audio alignment. Our last year's pipeline matched content across channels, then searched a grid of candidate offsets. We replace it with a single regressor that does neither. Instead, it regresses each keypoint's inter-channel offset directly. Every offset follows an affine law, $\delta = \alpha t + \beta + r$. The term $\alpha t + \beta$ is a constant offset plus a linear drift, shared across the whole recording. The residual r is small, bounded, and specific to each keypoint. We pool the file-level parameters with attention over all keypoints. The prior is almost exact: it fits the development data to $R^2 \geq 0.9999$. A wide channel-1 context window gives one model the range it needs. It recovers offsets from a few milliseconds on ARU up to several seconds on zebra finch, as much as 3.8 s, with no coarse-to-fine search. The gains are clear on the 2026 development sets. We reach 0.0138 MSE (91.8 ms MAE) on ARU and 0.315 MSE (442 ms MAE) on zebra finch. The 2026 deep-learning baseline reaches 0.040 and 1.183 MSE on the same two sets. Averaged, that is about 0.16 MSE for us against 0.61 for the baseline. We also run a set of diagnostics to find where the difficulty lies. A floor near 430 ms stays in place either way.

2. SYSTEM OVERVIEW

Our system is a single structured regressor. It predicts the inter-channel offset of every keypoint in one forward pass. There is no candidate search and no coarse-to-fine stage. The same model handles both datasets, changing only the drift bound E ($E = 0.5$ s for ARU, $E = 5$ s for zebra finch), and it covers offsets from milliseconds on ARU to 3.8 s on zebra finch.

The design rests on one observation: clock drift makes the offset an almost perfect straight line in time ($R^2 \geq 0.9999$ on the development data). So we do not predict 100+ independent offsets. We fit one line per file and let the model spend its capacity on its two parameters. For a file of duration T, each keypoint $k0_i$ on channel 0 has target offset $\delta_i = k1_i - k0_i$; we predict δ_i and emit $k1_i = k0_i + \delta_i$.

Figure 1 shows the pipeline. The stages are described below.

2.1. Inputs and encoding.

For each keypoint we cut two windows: a 1 s reference clip from channel 0, and a wider context window of length $1 + 2E$ from channel 1, centered so the true match lies inside it. The context is wide enough to contain the true match, even zebra finch offsets up to 3.8 s, so the model can see it directly instead of searching for it. A frozen BEATs encoder turns both windows into features. We mean-pool each into time bins and concatenate them into one per-keypoint vector p_i.

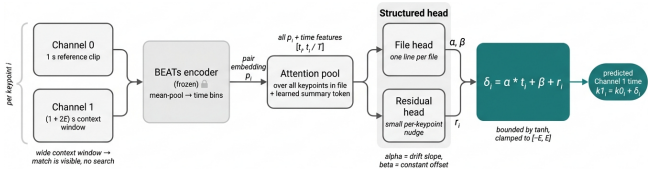

Figure 1: **Structured-regression alignment pipeline.** For each annotated keypoint i, we extract a 1 s reference clip from Channel 0 and a $(1 + 2E)$ s context window from Channel 1. The context window is centred so that the true match lies inside it without search, with E the maximum allowed error. A frozen BEATs model encodes both windows; the resulting features are mean-pooled into fixed time bins and concatenated to form a pair embedding p_i. All keypoint embeddings in the file are augmented with time features $[t_i, t_i/T]$ (absolute Channel-0 time and its fraction of file duration T) and passed through an attention-pooling layer that uses a learned summary token. This produces two outputs: a file head that emits a single affine drift line per file (slope α and offset β), and a residual head that emits a small per-keypoint residual r_i. The predicted offset is $\delta_i = \alpha \cdot t_i + \beta + r_i$, where α, β, and r_i are each individually $\tanh$-bounded to their physical scales and the sum is clamped to $[-E, E]$. The aligned Channel-1 time is recovered as $k1_i = k0_i + \delta_i$. This design formalizes the assumption that most cross-channel misalignment is a smooth clock drift, captured by the file-level affine line, while the residual head handles only small, keypoint-specific deviations.

2.2. Structured head.

A keypoint alone is noisy, so we combine all of them. The per-keypoint vectors $\{p_i\}$, with time features $[t_i, t_i/T]$, pass through a transformer-encoder layer with a learned summary token. From the summary the model reads two file-level numbers: the constant offset β and the drift slope α. A second head reads each keypoint token and adds a small residual r_i. The final offset is

$$\delta_i = \alpha \cdot t_i + \beta + r_i, \tag{1}$$

with

$$\alpha = \tfrac{E}{T}\tanh(\cdot), \quad \beta = E\tanh(\cdot), \quad r_i = s\,E\tanh(\cdot), \quad \text{clamped to } [-E, E]. \tag{2}$$

All outputs are bounded by $\tanh$ and clamped to $[-E, E]$, so the model cannot predict a physically impossible offset. The residual is capped at 10% of E ($s = 0.1$): the line is most of the answer, the residual only nudges. This affine prior is the inductive bias. The model spends almost all its capacity localizing two file-level numbers (α, β) rather than 100+ independent offsets, which is why a single direct model replaces the coarse-to-fine search a 1 s matching window would otherwise require (the next section shows the two-stage alternative was worse).

2.3. Training and inference.

We train the head with a Huber loss on δ ($\beta = 0.05$) and an L2 penalty on the residuals (weight 1.0) that keeps r_i near zero unless the line truly fails. The encoder is frozen by default, with the option to unfreeze the last N BEATs layers; the head is trained with Adam, gradient accumulation, and gradient clipping. We select the epoch based on validation MSE (the real metric), not the surrogate BCE used by the classifier baseline. For zebra finch, which has only 108 training files, we re-roll each file's drift every epoch and re-extract the channel-1 windows so the true match lands at a new offset. This forces the model to read the offset from content instead of memorizing it, while preserving the real cross-channel acoustic relationship (aligned-clip cosine ≈ 0.63) rather than warping a copy of channel 0 (cosine ≈ 1.0, an unrealistically easy task). At inference we cut the windows, run one forward pass per file, and emit $k1_i = \mathrm{clip}(k0_i + \hat{\delta}_i, 0, T)$.

3. RESULTS

All numbers are from the 2026 development validation set, evaluated with the official evaluate.py (MSE in sec², MAE in ms). We treat the 2026 deep-learning baseline as the primary reference.

Table 1: 2026 development *validation* results. Best per column in bold.

	ARU		zebra finch	
Method	MSE	MAE	MSE	MAE
nosync ($\delta = 0$)	0.0190	116	5.740	2297
GCC-PHAT	0.0201	122.4	7.995	2457
2026 DL baseline	0.0402	164.3	1.183	744.5
Ours (structured)	**0.0138**	**91.8**	**0.315**	**442**

The average MSE across the two datasets (the combined ranking metric) is ≈ 0.164, versus 0.611 for the 2026 deep-learning baseline. ARU is config-insensitive across 0.0138–0.0148 and far below both nosync and the baseline; zebra finch's 0.315 is statistically tied with the un-augmented model (0.320) within the 16-file validation noise, and we select the regularized augmented checkpoint for its better-behaved training dynamics under the expected evaluation-set domain shift.