# Robopiano

Jack Landers
*Santa Clara University*
jlanders@scu.edu

Anthony Angeles
*Santa Clara University*
aeangeles@scu.edu

Tin Nguyen
*Santa Clara University*
tnguyen31@scu.edu

***Abstract*—Learning to play the piano is an exercise in extreme robotic dexterity that proves a robot learning policy for bimanual control to be as precise as the most practiced of humans. Piano playing is an art of synchronous dynamics that must be perfected to sound correct. While there is little utility for robots that can do this, the performance of developing a generalized action policy for producing the hand movements to play back a song, is one that pushes the state of the field to the limits of what can be achieved. Should a model be capable of this, it would be safe to assume that it could learn control of the hands to the highest degree of any ability. For this, we introduce Sonata 3 as a novel architecture that builds on the works of Robopianist and RP1M, to produce a robot learning policy in 3 sequential stages: Primitive Learning, Attention Transformer, and Diffusion Refinement.**

## I. Introduction

Unlike engineered systems, biological hand coordination emerges from a tightly coupled nervous and musculoskeletal system that provides continuous closed loop feedback, making it difficult to design reward signals that capture the full complexity of the task. We explore how a robot learning policy can most effectively generalize from music to actions. The goal is to determine if robots can truly compute the maximum dextrous control that humans can achieve. We begin by examining what the best reward function would be for such a reinforcement learning policy, and go further to propose and evaluate a self-supervised learning policy.

Robopianist introduced an open-source MuJoCo simulation environment to train bimanual control models for playing piano, with an early stage policy that learned with Markov Decision Process based Reinforcement Learning from Demonstration. RP1M then expanded on this, developing a behavior cloning model that learned in that same environment to produce a more diverse dataset including undemonstrated songs, open sourcing outputs, and proving that a supervised diffusion model could be trained for effective action policy from these artificial demonstrations.

Our approach was to push the state of the field further towards increased generalization and accuracy sufficient for listening. To begin, we first aim to accelerate the Robopianist environment such that the hardware requirements for training are reduced. Next, we explored how a more complex reward function could potentially improve the Markov Decision Process, learning to a higher degree of accuracy. Ultimately, we propose Sonata as a novel 3-stage architecture, which first utilizes GMR for selecting primitives that are reusable across diverse song demonstrations for low level control. Then, a transformer learns to plan a structured sequence of primitives, and lastly a diffusion model refines the complete low level control policy for smooth and precise movements.

Altogether, we found that we can accelerate the Robopianist environment using MJX for GPU parallelism. Next, improve on the state of the art Markov Decision Process (MDP) reward function for reinforcement learning in Robopianist. Furthermore, we prove our concept of a novel deep architecture and begin training to push the state of the field further on Santa Clara University's WAVE HPC.

## II. Accelerating RoboPianist

In developing an efficient training pipeline for RoboPianist, we identified a fundamental bottleneck in the original implementation: although the original system used a capable algorithm, its configuration left the GPU almost entirely idle during training. The MuJoCo physics simulation runs sequentially on a single CPU thread at 407 steps/sec, and when paired with only one gradient update per step, even a modern GPU like our RTX 5090 FE spends the vast majority of wall-clock time waiting on the simulator. Compounding this, piano rewards are temporally sparse meaning reward only arrives at the exact moment a key is depressed thus making credit assignment difficult at low UTD. The work reconfgures the training loop to saturate the GPU and address both problems.

### A. Background

The original RoboPianist [2] trains with DroQ, a SAC variant that adds LayerNorm and dropout to twin Q-networks for critic stability. Despite using this sample-efficient architecture, the original implementation runs at an update-to-data (UTD) ratio of 1 (one gradient update per environment step) with a minibatch size of 256 in JAX across 4 Tesla K80 GPUs. Training a single song required approximately 5M environment steps. At 407 env steps/sec, UTD=1 yields only $\sim$529 effective gradient updates per wall-clock second, leaving the GPU idle the overwhelming majority of the time.

TABLE I: Original vs. Our DroQ Configuration

| **Aspect** | **Original [2]** | **Ours** |
|---|---|---|
| Algorithm | DroQ (UTD=1) | DroQ (UTD=**20**) |
| Framework | JAX | **PyTorch/CUDA** |
| Minibatch size | 256 | **1024** |
| Credit assign. | 1-step returns | **N-step** ($N = 3$) |
| Policy init. | Xavier random | **BC from RP1M** |
| Hardware | 4$\times$ K80 | RTX 5090 FE |
| Eff. updates/sec | $\sim$529 | $\sim$**10,560** |

### B. High-UTD DroQ in PyTorch

We port the training pipeline to PyTorch and increase the UTD ratio from 1 to 20, performing 20 gradient updates per environment step. This shifts the bottleneck from *environment-bound* (waiting on CPU simulation) to *GPU-bound* (saturating the critic and actor with updates):

Listing 1: UTD=20 inner loop (phase3_droq_finetune.py)

```
if step > seed_steps:
   for _ in range(utd_ratio): # 20 updates per
       env step
      agent.update(buf)
```

Running 20 gradient updates per step on identical hardware would normally cause standard SAC critics to overfit and diverge. DroQ prevents this by applying LayerNorm and dropout ($p = 0.01$) after every hidden layer of both Q-networks:

Listing 2: DroQ critic architecture (droq.py)

```
@staticmethod
def _build(in_dim, hidden, dropout):
   return nn.Sequential(
      nn.Linear(in_dim, hidden),
          nn.LayerNorm(hidden),
      nn.Dropout(dropout), nn.ReLU(),
      nn.Linear(hidden, hidden),
          nn.LayerNorm(hidden),
      nn.Dropout(dropout), nn.ReLU(),
      nn.Linear(hidden, hidden),
          nn.LayerNorm(hidden),
      nn.Dropout(dropout), nn.ReLU(),
      nn.Linear(hidden, 1),
   )
```

LayerNorm stabilizes the scale of Q-value estimates across the wide range of piano rewards; dropout prevents the critic from memorizing the replay buffer at high update frequency.

We additionally clamp the entropy temperature at $\alpha_{\min} = 0.05$ and clip gradients at max norm 1.0. Both were necessary in practice: early runs without these safeguards saw $\alpha$ collapse to 0.002 and critic gradients explode during high-UTD training.

### C. N-Step Returns for Sparse Piano Rewards

Piano rewards arrive only at the moment of keypress, giving the agent zero signal during the finger-positioning movements that *cause* the keypress. With 1-step returns, these preparatory actions receive no direct credit. We use $N = 3$ step returns:

$$r_t^{(N)} = \sum_{k=0}^{N-1} \gamma^k \, r_{t+k}, \qquad \text{bootstrap: } Q_{\text{target}} = r_t^{(N)} + \gamma^N \, V(s_{t+N}) \tag{1}$$

This propagates reward credit back 3 timesteps, reaching the pre-press positioning actions. The buffer stores $\gamma_{\text{eff}} = \gamma^n$ (or $\gamma^{n'}$ for $n' < N$ when an episode ends early) to keep the discount correct at boundaries:

Listing 3: N-step buffer with episode-boundary handling (droq.py)

```
buf = NStepReplayBuffer(
   capacity=300_000, obs_dim=OBS_DIM,
       act_dim=ACT_DIM,
   n_steps=3, gamma=0.88
)
# Each stored transition carries its own
    effective gamma:
# eff_gamma = gammaˆN, or
    gammaˆ(episode_end_step+1) if done early
obs, actions, rewards, next_obs, dones,
    eff_gammas = buf.sample(batch)
target_q = rewards + eff_gammas * (1 - dones) *
    (tq - alpha * next_log_prob)
```

### D. Minibatch Size and Memory Bandwidth

We increase the minibatch from 256 to 1024. On a replay buffer of $3 \times 10^5$ transitions, the larger batch reduces gradient variance and better saturates GPU memory bandwidth. We also allocate replay buffer arrays in pinned (page-locked) CPU memory and use non-blocking GPU transfers to overlap data movement with the previous step's compute:

Listing 4: Pinned memory for async CPU-to-GPU transfer (droq.py)

```
def sample(self, batch_size: int):
   idx = np.random.randint(0, self.size,
       size=batch_size)
   to = lambda x: (torch.FloatTensor(x[idx])
                .pin_memory()
                .to(DEVICE, non_blocking=True))
   return to(self.obs), to(self.actions),
       to(self.rewards), \
        to(self.next_obs), to(self.dones),
            to(self.eff_gammas)
```

### E. Mixed-Precision Training

We evaluated Automatic Mixed Precision (AMP, fp16) but found it *reduces* throughput on this architecture: 368 updates/sec with AMP versus 528 without. The 3-layer, 256-hidden-unit MLPs used here are below the matrix tile size at which fp16 GEMM on Tensor Cores outweighs the overhead of fp16 casts. AMP is therefore disabled in all DroQ experiments.

### F. Behavioral Cloning Pre-training from RP1M

Before DroQ fine-tuning, we initialize the actor via behavioral cloning on the RP1M dataset. The actor is supervised with MSE loss against expert action trajectories for the 39 finger DOFs present in RP1M; the 6 wrist DOFs not covered remain at Xavier initialization and are discovered by RL. This gives the policy a motor prior (plausible finger kinematics) rather than requiring RL to search from a random starting point.

## III. Reward Function Exploration

Dexterous hands manipulation tasks are challenging due to the high dimensionality of control spaces and sparse reward signals. Robotic piano playing stretch the complexity due to

the agent must precisely control and timing multiple fingers across two hands.

Reinforcement learning (RL) approaches have demonstrated the ability to learn complex motor behaviors in simulated environments. However, sparse reward signals often lead to unstable exploration and slow convergence. In piano-playing tasks, agents frequently exhibit random key pressing rather than learning structured musical sequences.

This reward exploration introduces additional reward shaping techniques that guide the agent toward meaningful exploration.

### A. Original Reward Structure

The original baseline reward functions consist of several components:

| Reward Component | Purpose |
|---|---|
| Key Press Reward | Correctly pressing target piano keys |
| Sustain Reward | Correct sustain pedal timing |
| Energy Reward | Penalize excessive actuator power |
| Fingering Reward | Minimize finger distance to keys |
| Forearm Reward | Prevent collisions between arms |

TABLE II: Original reward components in the piano RL environment

Although these rewards are intended to guide the agent in playing notes correctly, they do not provide sufficient guidance at the beginning of the exploration stages.

### B. Proposed Reward Augmentations

There are three additional reward components that are introduced to help improve training more efficiency.

*1) Key Proximity Reward:* This reward motivates the agent to move fingers toward the keys that should be pressed, which reduce delay in timing.

$$R_{proximity} = \frac{1}{N}\sum_{i=1}^{N} e^{-d_i} \tag{2}$$

where $d_i$ represents the Euclidean distance between a fingertip and the closest target key.

The benefit of this reward provides a feedback loop even before a key presses.

*2) Smooth Motion Reward:* Smooth motion is encouraged by penalizing high joint velocities, which reduces jerky movements from the hands.

$$R_{smooth} = e^{-\sum_j v_j^2} \tag{3}$$

where $v_j$ denotes the velocity of the joint $j$. Lower joint velocities correspond to smoother and more stable movements of the hands.

*3) Temporal Anticipation Reward:* The reward encouraged the agent to prepare for upcoming notes, which improves musical timing.

$$R_{anticipation} = \frac{1}{M}\sum_{k=1}^{M} e^{-0.5 d_k} \tag{4}$$

where $d_k$ is the distance between the fingertips and the keys that will be pressed in future times-teps within a fixed look-ahead window.

### C. Combined Reward Function

The final reward function is the sum of all reward components:

$$\begin{aligned} R_{total} =& R_{key\_press} + R_{sustain} + R_{fingering} \\ &+ R_{forearm} + R_{energy} \\ &+ R_{proximity} + R_{smooth} + R_{anticipation} \end{aligned} \tag{5}$$

This is the reward scaling used during training:

| Reward Component | Weight |
|---|---|
| Key Press Reward | 4.0 |
| Fingering Reward | 2.0 |
| Energy Reward | 0.2 |
| Smooth Motion Reward | 0.3 |
| Anticipation Reward | 0.2 |

TABLE III: reward weighting used during training

Correct weighting is very important because it ensures that the additional reward components guide the learning process smoothly without dominating the original baseline.

## IV. Sonata 3 - A Novel Architecture

Sonata 3 intends to combine state of the art robot learning concepts into one deep model in order to learn the most generalized bimanual control model for robopiano, yet. Using the RP1M dataset, we use GMM clustering as proposed by citation to learn generalized primitives that are useful across various piano pieces. These should fit to represent common techniques. Previously, the MDP enabled probabalistic reward for upcoming key presses to produce a sequence of movements, however after employing this strategy to great depth, we have found its limitations. Instead we choose to train a transformer for primitive selection based on lookahead attention to keys. Finally, a diffusion model denoises to smooth the trajectory, finding the greatest fit from the combined primitive paths for playing back the goal notes. While all these techniques have shown promise in prior literature, Sonata is the first model to use them in combination, optimizing for intentional bimanual control with a high complexity of action.

### A. Primitive Learning

As a foundation for low level control, we generalize across many piano pieces from the RP1M 300 dataset by segmenting them with a Gaussian Mixture Model. With this, we learn a discrete vocabulary of motion primitives that are each then fit by a phase-conditioned GMR prior. These low level actions are tokenized inputs that can be fed to a high level learning model for handling intention.

### B. Attention Transformer

Sonata uses a causal sequence model over primitives similarly to an LLM. This has been introduced as a Decision Transformer, applying attention to determine high level abstract actions. With this, we predict the next states directly from inputs rather than following conditional reward as an MDP might. Our autoregressive model receives an embedded input of both previously selected primitives as well as the goal positions, in order to produce a near sighted prediction of upcoming actions based on short term context. This approach attempts to reproduce that of the pianist reading a score as they play.

### C. Diffusion Refinement

Furthermore, with low level control defined loosely by the primitives, the developed trajectory sequences still require fine tuning. The results of RP1M show great promise for the technique of diffusion refinement. Such a self supervised model enables a generative sequence of actions with excellent scalability into high dimension. The denoising diffusion process with stochastic Langevin steps sequentially moves towards a probabile space by adding noise for exploration. The time-series diffusion transformer has produced the highest complexity and rate of actions, but for simplicity we chose to explore and MLP based denoising approach, making this stage an area for further development.

## V. Comparing Results

### A. Accelerating Robopianist

TABLE IV: RoboPianist Training Throughput Benchmark (RTX 5090 FE)

| Configuration | Batch | AMP | Updates/sec |
|---|---|---|---|
| Environment (MuJoCo CPU) | – | – | 384.9 steps/sec |
| DroQ UTD=1 [2] | 256 | No | 533 |
| DroQ UTD=20 (ours) | 1024 | No | 525 |
| DroQ UTD=20 (ours) | 1024 | Yes | 394 |

TABLE V: Effective Gradient Updates per Wall-Clock Second

| Configuration | Eff. updates/sec | Speedup | Bottleneck |
|---|---|---|---|
| DroQ UTD=1 [2] | 533 | 1× | ENV-bound |
| DroQ UTD=20 (ours) | 10,492 | **19.7**× | GPU-bound |
| Original: 5M env steps → 5M grad updates (≈5 hrs, 4×K80) | | | |
| Ours: 0.25M env steps → 5M grad updates (≈0.2 hrs est.) | | | |

TABLE VI: Convergence Probe: Critic TD-Loss After 2,000 Environment Steps

| Configuration | Batch | Grad Updates | Critic TD-Loss |
|---|---|---|---|
| DroQ UTD=1 [2] | 256 | 1,500 | 14.72 |
| DroQ UTD=20 (ours) | 1024 | 19,500 | **0.076** |
| Improvement | | | **99% lower TD-loss** |

Table IV shows that AMP reduces throughput on our small MLP architecture (394 vs. 525 updates/sec), so it is disabled in all experiments. The per-update speed is nearly identical between the original and our configuration (533 vs. 525), confirming the 19.7× effective speedup in Table V comes entirely from the UTD ratio change. Table VI provides empirical confirmation: given the same 2,000 environment steps, DroQ UTD=20 accumulates 13× more gradient updates and converges to a critic TD-loss of 0.076 vs. 14.72 for the original achieving a 99% reduction.

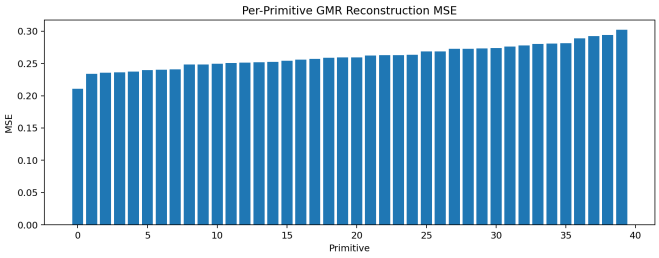
### B. Reward Function

The proposed reward shaping strategy provides several benefits:

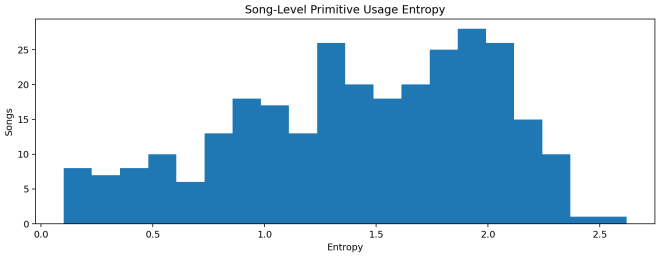
- Decreased random key presses during exploration
- Increased finger-to-key alignment
- Smoother hand trajectories and avoid hand collisions
- Faster convergence of RL.

These improvements help the agent learn meaningful piano-playing behaviors more efficiently.

### C. Sonata




(a) MSE Per Primitive




(b) Primitive Usage Entropy

Fig. 1: Sonata Stage 1 Results

Fig 1.a shows how well each learned primitive can reconstruct the trajectory segments it represents. The errors

fall within a relatively tight band ( 0.21 to  0.30) with no extreme outliers, indicating that all primitives are modeling their assigned motions in a stable and consistent way. The gradual increase in MSE across primitives suggests that some primitives correspond to more complex or less frequent motion patterns, which are naturally harder to model. Importantly, there are no signs of failed clusters or degenerate primitives, meaning the clustering and GMR fitting process in Stage 1 successfully produced a coherent set of motion primitives.

The song-level primitive usage entropy histogram in Fig 1.b measures how diverse the primitive usage is within each song. The distribution, centered roughly between 1.2 and 2.2, indicates that most songs use a mix of multiple primitives rather than collapsing to a single dominant one. This is a strong signal that the learned primitive vocabulary is expressive and being meaningfully reused across different musical sequences. Entropy is not excessively high, which would suggest randomness or lack of structure. Instead, the balance observed here implies that primitives capture reusable motion patterns while still allowing structured composition.

Successful training of Stage 2 produced an attention transformer that selected primitives with an 86% top-k accuracy, strong family accuracy, and a dynamics accuracy within an acceptable range. Stage 3 is now undergoing training with promising initial results showing diffusion loss making gradual improvements over epochs. We hope to refine this model and conclude that we can truly generalize a robot learning policy to play the piano better than ever before.

## ACKNOWLEDGMENT

The Introduction, Abstract, and Comparison sections were written collectively with equal contributions from all authors. Anthony Angeles explored accelerating robopianist and is the author of Section 2, showing successful results running locally on his device. Tin Nguyen Developed a new Reward function, authored Section 3, and showed results improving on the current state of the field. Jack Landers developed the Sonata 3 architecture, employing strategies built on the findings of Anthony and Tin, which is still training on Santa Clara's WAVE HPC Cluster, but has shown promising results in initial stages thus far. The codebase for Sonata 3 was written using GPT Codex 5.4 Extra High.

## REFERENCES

[1] Y. Zhao, L. Chen, J. Schneider, Q. Gao, J. Kannala, B. Schölkopf, J. Pajarinen, and D. Büchler, "RP1M: A Large-Scale Motion Dataset for Piano Playing with Bimanual Dexterous Robot Hands," in *Proc. Conf. on Robot Learning (CoRL)*, 2025.

[2] K. Zakka, P. Wu, L. Smith, N. Gileadi, T. Howell, X. B. Peng, S. Singh, Y. Tassa, P. Florence, and A. Zeng, "RoboPianist: Dexterous Piano Playing with Deep Reinforcement Learning," in *Proc. Conf. on Robot Learning (CoRL)*, 2023.

[3] J. Mao et al., "Robot Learning from a Physical World Model," *arXiv preprint arXiv:2511.07416*, 2025.

[4] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision Transformer: Reinforcement Learning via Sequence Modeling," *arXiv preprint arXiv:2106.01345*, 2021.

[5] S. Sudhakaran and S. Risi, "Skill Decision Transformer," *arXiv preprint arXiv:2301.13573*, 2023.

[6] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," *arXiv preprint arXiv:2303.04137*, 2023.

[7] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning Latent Plans from Play," *arXiv preprint arXiv:1903.01973*, 2019.

[8] K. Pertsch, Y. Lee, and J. J. Lim, "SPiRL: Skill Priors for Reinforcement Learning," *arXiv preprint arXiv:2010.02893*, 2021.

[9] T. Shu et al., "Skill Transformer: A Transformer-Based Framework for Skill Modeling," *arXiv preprint*, 2023.

[10] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[12] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, 1998.

[13] R. Veerapaneni et al., "Learning Parameterized Action Primitives for Robotic Reinforcement Learning," *arXiv preprint*, 2019.

[14] "Single-Song GRPO Fine-Tuning," Unpublished / internal work, 2025.

[15] "GPU-Parallel Episode Collection via MJX," Unpublished system description, 2025.

[16] "Multi-Song Shared LoRA Training (v8/v9)," Unpublished method, 2025.

[17] "Multi-Song Shared Training (v8/v9)," Unpublished method, 2025.

[18] J. Landers, 'https://github.com/JacktheLander/robopiano/tree/feat/sonata-online-stage1'

[19] A. Angeles, 'https://github.com/aneangel/dexterous-robotics'