RoboVirtuoso: Attempts at Training a Multi-Task Piano-Playing Agent

Stanford CS224R Project Final Report Spring 2023

Matt Smith
Department of Computer Science
Stanford University
mjksmith@stanford.edu

Eric Ye
Department of Electrical Engineering
Stanford University
ericye@stanford.edu

Abstract

RoboPianist [1] is a suite of robotic control tasks in which two anthropomorphic robot hands are tasked with playing a piece on a simulated piano. The tasks are challenging as they have sparse rewards and a high-dimensional action space. We aim to train a model-free agent which is able to generalize to new pieces with little-to-no additional training. To do so, we use the setup from [1] as our baseline and evaluate several modifications:

Scale Pre-training Inspired by human piano curricula, we create a set 12 environments corresponding to the types of major scales and use these environments to pre-train the agent before finetuning on the desired environment. We also explored pretraining from pieces in the PIG dataset.

Architecture changes The baseline approach uses 3-layer perceptron encoders for both the agent and the critic. The encoders are given a state representing the current goal as well as a lookahead for the next 10 time steps, representing 0.5 seconds of lookahead. To help our encoder better differentiate pieces when pretraining, we wanted to increase the size of the lookahead and make use of the temporal nature of the goal. To do so, we replace the MLP encoder with a bidirectional transformer encoder and increase the lookahead to 100 steps (5 seconds).

Hindsight Experience Replay (HER) In order to deal with the sparse reward, and to reduce our dependancy on finicky reward shaping, we apply Hindsight Experience Replay [2].

Key Reward Tiering To counter an observed mode collapse where the robot hands never touched the piano to avoid a negative reward, we modify the reward function to give the agent less reward when it doesn't press any keys.

We evaluated our approaches on the ROBOPIANIST-etudes-12 dataset from Zakka et al. [1], a collection of 12 randomly-sampled pieces from the PIG dataset [3]. The primary evaluation metric is F_1 score.

We found that pretraining on scales improved F_1 score by 13.7% vs the randomly-initialized baseline. However, the model could not learn to perform multiple pieces at once, and so simultaneous pretraining on pieces from the PIG dataset was not more effective than random initialization. We also found that replacing the MLP with a transformer encoder and applying HER were both harmful to learning. Finally, we found that Key Reward Tiering was able to improve F_1 score by 27.8% compared to the baseline reward scheme, but did not enable multi-piece learning.

We believe that pretraining the agent on multiple pieces is essential for the agent to exhibit the few-shot learning we desire. In order to help the agent learn to play multiple pieces, we are interested in investigating techniques that assign IDs or embeddings to each task, such as DREAM [4].

1 Introduction

RoboPianist is a benchmarking suite for robotic control tasks created by Zakka et al.[1] The RoboPianist suite tasks two simulated anthropomorphic robot hands with performing a piece of music on a simulated piano. See Figure 1 for a render of this environment. The suite is made up of 150 pieces of music sourced from the PIG dataset [3], which include fingering annotations. The performance tasks are challenging due to the high dimensionality of the action space and the sparse rewards. The authors explore both model-based and model-free approaches, though we focus on model-free approaches here.

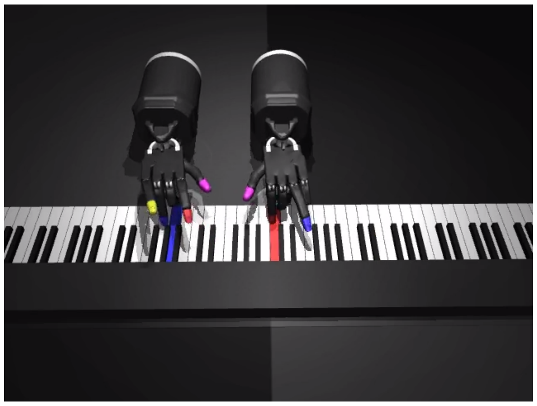

Figure 1: Render of RoboPianist playing some keys from behind the piano.

In their model-free approaches, Zakka et al.[1] train a model independently for each piece in the suite, and models trained on one piece do not generalize well to other pieces.

2 Motivation

RoboPianist is an interesting challenge to tackle because it requires a high level of precision and speed from the actuators and agent to perform at a human level. The nature of the task is challenging since there is a goal at each step in time rather than just at the end of the episode, and the agent must not only play the correct keys at the current timestep but prepare itself to play correct keys in the future. Additionally, as we'll discuss in this report, this task does not respond well to common reinforcement learning techniques such as PPO and hindsight experience relabelling, possibly due to the sparse nature of the reward. This makes it an interesting problem to try to solve with novel reinforcement learning techniques.

3 Related Work

Since RoboPianist is a new benchmark, there have not been many previous attempts to improve its performance. The model-free baseline in [1] explored several RL techniques to achieve modest performance on the classical pieces. The model was trained using DroQ, a variant of Soft-Actor-Critic algorithm. The authors of [1] found that PPO did not achieve reasonable performance even after 10x as many transitions as Soft-Actor-Critic.

The authors of [1] wrote that some modifications to the formulation of the model-free approach improved the performance of the system. These modifications were:

1. Fingering reward – Adding annotated fingering information for key key played into the model.
2. Lookahead horizon – Allowing the model to look ahead *n* steps of keys in the future, rather than just the keys at the current timestep.
3. Energy penalty – A reward term that penalizes the model based on how much energy the hand actuators need.

4. Action space reduction – reducing the flexibility of some of the hand actuators and disabling some degrees of freedom in the hand actuators.
5. Action-reward – Appending the previous action and reward to the transition tuple added to the replay buffer.

With this approach, [1] achieved an average F1 of 0.538 ± 0.122 across ROBOPIANIST-etude-12 in the dataset.

We were inspired by other work in deep learning when working on this project, notably transformers, and pretraining and fine-tuning.

Transformers [5] were a transformative model architecture that enabled modern large language models, which are known for their ability to generalize to new unseen tasks. Transformers include attention layers, which allow the model to perform a learned dictionary-style lookup on the inputs, unlocking computations that are not possible with simple multilayer perceptrons. BERT, which we use in this work, is a language model built on transformers. We hypothesized that using a transformer-based architecture would allow the model to generalize better and learn a task-agnostic model.

Pretraining and fine-tuning is another concept commonly applied to large language models introduced in [6], [7] and other works. Large language models are commonly "pre-trained" on a large corpus of unstructured text, where the language models learns grammar and other information about the structure of text, but does not perform well on a specific task. Fine-tuning is the practice of taking the model after pretraining and optimizing the model parameters for a specific task, e.g. by providing supervised examples. This avoids having to retrain a model from scratch for a variety of tasks, since what the models learns during pretraining can be applied to many tasks. We believed that the piano might be similar; that a piano-playing agent must learn similar techniques across different songs.

4 Scale Pretraining

4.1 Hypothesis

Many classical songs in the repertoire are challenging to play. Humans learning to play piano will first master simpler study pieces and finger exercises such as scales before attempting to learn these more challenging songs. We hypothesized that pretraining the model-free agent on a set of exercises would improve the ability of the agent to learn new pieces. To this end, we created a set of 12 major scale environments, one for each pitch class on the standard piano. The goal of the environments is to have the left and right hands simultaneously play one octave of the major scale, up then down, in straight quarter notes at 80 BPM.

4.2 Methods

We pretrained our agents on various environments and then recorded their performance on ROBOPIANIST-etude-12 [1], a collection of 12 randomly-selected pieces from the PIG piano pieces with fingering dataset [3]. In all of our experiments, finetuning and pretraining both last 1 million steps using the same algorithm and hyperparameters as in [1] except that we replace the ReLU activations of the MLP with GELU activations[1] [8]. In addition, when seeding the replay buffer when finetuning, we sample from the model rather than choosing actions randomly. When pretraining on multiple environments, we give each environment its own replay buffer and then cycle through the replay buffers when sampling minibatches for training.

Single-scale pretraining We pretrained the agent on the C Major scale with fingering information.

Multi-scale pretraining We pretrained the agent on all 12 major scale environments.

Multi-piece pretraining We pretrained the agent on pieces in the PIG dataset, minus the 12 pieces in the ROBOPIANIST-etude-12. We also cut out one piece that was overly similar to on of the test pieces, to avoid contamination.

[1]We didn't do this on purpose – the library we got from [1] to replicate the results uses GELUs by default and we didn't notice the discrepancy until it was too late.

To evaluate our approaches, we compared F_1 score obtained by our finetuned agents vs the F_1 score obtained by a randomly initialized agent trained for 1 million steps using the same hyperparameters. To compute F_1 score, we use the set of notes the agent played and the set of goal notes at each timestep to compute precision and recall, and report F_1 as the average per-timestep F_1:

$$F_1 = \frac{1}{T}\sum_{t=1}^{T}\frac{2 \cdot \text{precision}_t \cdot \text{recall}_t}{\text{precision}_t + \text{recall}_t}$$

4.3 Results

The single-scale and multi-scale pretrained models performed better than models without pretraining. They learned faster and averaged 13.7% higher F_1 score after 1M training steps on the 12 etudes in the test set. See Table 1 for detailed results.

Piece	**Baseline**	**Single-Scale Pretraining**	**Multi-Scale Pretraining**
Bagatelle Op3 No4	0.302	**0.380**	0.325
French Suite No 1 Allemande	0.496	**0.564**	0.543
French Suite No 5 Gavotte	0.272	0.404	**0.672**
French Suite No 5 Sarabande	0.512	0.645	**0.769**
Golliwoggs Cakewalk	0.442	0.**444**	0.379
Kreisleriana Op 16 No 8	**0.421**	0.249	0.390
Paritita No 26	0.530	**0.598**	0.463
Piano Sonata D845 1St Mov	0.320	0.242	**0.357**
Piano Sonata K279 In C Major 1St Mov	0.423	0.484	**0.609**
Piano Sonata No 2 1St Mov	0.578	**0.604**	0.581
Piano Sonata No 23 2Nd Mov	0.547	**0.646**	0.537
Waltz Op 64 No 1	**0.397**	0.374	0.343
Average	0.437 ± 0.064	0.469 ± 0.091	$\mathbf{0.497 \pm 0.091}$

Table 1: F_1 score of pieces in the ROBOPIANIST-etude-12 after 1M steps. Both single-scale pretraining and multi-scale pretraining show an improvement over the baseline.

In our initial iterations, multi-piece pretraining had significant mode collapse where it would never touch the piano. In order to address this issue, we investigated applying Hindsight Experience Relabeling [2], adjusting the environment's reward function to encourage more interaction with the piano, and making architectural changes to the actor and the critic.

5 Hindsight Experience Relabelling

5.1 Hypothesis

Inspired by one of the homework assignments and the lectures, we attempted hindsight experience relabelling (HER) [2] on the agent. We hypothesized that relabelling incorrect trajectories can help the agent learn potentially novel ways of pressing the correct keys.

5.2 Methodology

During an episode, we record the keys played at each timestep along with the closest finger to each key. We also record the sequence of actions taken during the episode. We create a new target song that is conditioned with the goal of playing these recorded keys with the recorded fingers. We play the recorded actions through an environment with this new target piece, and add all the relabelled transitions to the replay buffer.

We found that the agent would mistakenly play many keys at once before it is well-trained, and that this would lead to a different distribution of goals in the relabelled tasks compared to the original tasks. We were concerned that the goal conditioning was insufficient to elicit the agent to play differently when the goal was the original goal compared to the relabelled goal so we added an additional value in the observables vector to indicate whether an episode was a relabelled or the original vector. This way, we are treating the relabelled and original targets as two different tasks with a forced “embedding" to tell the networks which task it is training on, in addition to the baseline goal conditioning. This was inspired by the homework on task embeddings for multi-task agents.

We validated our implementation of relabelling by checking that the F_1 metrics of the agent in the relabelled episode was 1.0 (i.e. perfect), which is expected during the relabelled episode.

5.3 Results

We found that using hindsight experience relabelling made the agent significantly worse compared to training without relabelling. Adding the “relabelled" task embedding ID to the observable vector also did not help.

Piece	**Baseline**	**With HER**
Bagatelle Op3 No4	**0.302**	0.068
French Suite No 1 Allemande	**0.496**	0.057
French Suite No 5 Gavotte	**0.272**	0.020
French Suite No 5 Sarabande	**0.512**	0.038
Golliwoggs Cakewalk	**0.442**	0.258
Kreisleriana Op 16 No 8	**0.421**	0.012
Paritita No 26	**0.530**	0.078
Piano Sonata D845 1St Mov	**0.320**	0.216
Piano Sonata K279 In C Major 1St Mov	**0.423**	0.059
Piano Sonata No 2 1St Mov	**0.578**	0.024
Piano Sonata No 23 2Nd Mov	**0.547**	0.023
Waltz Op 64 No 1	**0.397**	0.024
Average	$\mathbf{0.437 \pm 0.064}$	0.0731 ± 0.051

Table 2: F_1 score of pieces in ROBOPIANIST-etude-12 after 1M steps with and without HER.

5.4 Discussion

We think that hindsight experience relabelling makes the agent worse because the distribution of notes in the relabelled episodes is different from the distribution of notes in the original pieces, and introducing these this distinct distribution of notes makes it more challenging for the agent to perform well at the original song. Even though we try giving the agent an embedding to help it distinguish between the types of goals, the difference in goals between the original and relabelled episodes might be too significant to improve performance against the original goals.

6 Tiered Rewards

6.1 Hypothesis

We noticed that in some of the relabelled training sessions, the agent would hover the hands over the keys but not actually play any of them, leading to poor accuracy. We hypothesized that this was due to the reward formulation.

The baseline reward formulation consists of five components summed together: the key press reward, the fingering reward, the sustain reward, the energy reward and the forearm reward. The fingering reward rewards the agent for having its fingers close to the correct positions on the piano. The sustain reward rewards the agent for pressing the sustain action correctly. The energy reward rewards the agent for conserving energy in its joints. The forearm reward rewards the agent for not colliding its forearms into each other.

The key press reward rewards the agent for pressing keys that should be pressed, and not pressing keys that should not be pressed.

We hypothesized that on challenging pieces, the energy and forearm reward would reward the agent for *not* moving (and therefore achieving high energy / forearm reward) and *not* pressing the keys more than the key press reward would reward it for pressing the keys correctly, especially early in training when the agent was not dexterous enough to avoid false positives.

To fix this, we introduce an idea called "**reward tiering**," where we only give the agent the reward for energy reduction and forearm collision avoidance if it achieves a high enough reward from the key press reward. This way we avoid rewarding the agent for not moving at all and hope to get it out of this local minimum.

We also noticed an odd detail of the key press reward. The key press reward consists of two parts: the reward for pressing keys that it should press correctly ("true positives", which we'll call the "recall reward" since it rewards high recall) and a reward for *not* pressing any keys that were not supposed to be pressed (which we'll call the "precision reward." These rewards are handled separately. The recall reward ignores keys that were incorrectly played, and the precision reward ignores whether the agent missed any keys. The recall reward is proportional to the number of keys that were correctly pressed, with 0.5 reward in a transition representing an agent that plays all the keys it was supposed to (and again, ignoring whether there were any keys that were played but were not supposed to be played). The precision reward is either 0 if the agent *any* keys that were not supposed to be pressed, or 0.5 otherwise.

Mathematically, we can express this as:

$$r_{krecall} = \frac{|\mathbb{K}_p \cap \mathbb{K}_g|}{|\mathbb{K}_g|}$$
$$r_{kprecision} = \begin{cases} 0, & \text{if } \mathbb{K}_p \not\subseteq \mathbb{K}_g \\ 1, & \text{otherwise} \end{cases}$$
$$r_k = 0.5 r_{krecall} + 0.5 r_{kprecision}$$

where $\mathbb{K}_p$ is the set of keys played in a given timestep, $\mathbb{K}_g$ is the set of goal keys in a given timestep, and $|s|$ is the cardinality (number of elements) in the set.

To illustrate this, consider the following examples. Assume that at one step in time, the agent's goal is to play two keys, one on each finger.

1. An agent that somehow plays all 88 keys of the piano will get a recall reward of 0.5 and a precision reward of 0, for a total key press reward of 0.5.
2. An agent that plays none of the keys will get a recall reward of 0 and a precision reward of 0.5, for a total key reward of 0.5.
3. An agent that plays both keys correctly and no other key will get a recall reward of 0.5 and a precision reward of 0.5, for a total key reward of 1.0.
4. An agent that plays one key correctly and another key incorrectly will get a recall reward of 0.25 and a precision reward of 0, for a total key reward of 0.25.

This meant that in some cases, playing no keys would result in a higher reward than playing some keys correctly and some keys incorrectly. On challenging songs, the agent might find it much easier just to take the 0.5 precision reward by not playing any keys rather than exploring how to play some keys, even if some other keys are played incorrectly.

To try to alleviate this, we devised a change to add a "reward tier" to the key reward as well, so that the agent would only get a precision reward if it achieved some amount of recall reward in a given timestep containing goal keys. Specifically, the precision reward is capped to the level of the recall reward in each timestep that contains goal keys. Mathematically, we can express this as:

$$r'_{kprecision} = \max\left(r_{krecall}, r_{kprecision}\right)$$
$$r'_k = 0.5 r_{krecall} + 0.5 r'_{kprecision}$$

In this report we call this specific modification "**Key Reward Tiering**."

6.2 Results

We found that Key Reward Tiering led to an improvement over baseline but reward tiering did not. We also found that Key Reward Tiering improved performance on hindsight relabelling significantly, although not above a no-relabelling baseline. The Key Reward Tiering seems to “unlock" better performance with hindsight relabelling, which hints that the reward might have been too tightly coupled to the baseline model. See full results in Table 3 and note how Key Reward Tiering improved relabelling performance significantly (over sixfold), but was still worse than baseline.

Relabelling	**Reward tiering**	**Baseline**	**Key Reward Tiering**
No	No	0.4958	**0.6482**
Yes	No	0.05705	**0.3508**
No	Yes	0.4777	Did not run†
Yes	Yes	$\approx 0^*$	**0.2983**

Table 3: F_1 score after 1M steps. Results related to hindsight experience relabelling and reward tiering. All results are based on French Suite No. 1 Allemande from ROBOPIANIST-etudes-12. *Training stopped early after 660k steps without achieving a nonzero F_1, which is significantly worse than any other models. † Did not run due to time constraints.

7 Architectural Changes

7.1 Hypothesis

The RoboPianist environemnts are quantized into 0.05s timesteps. The observation state includes a goal lookahead of 10 timesteps (representing 0.5s), which consists of a 10x89 binary matrix representing which keys should be active at each future timestep (88 bits at each timestep for keys + 1 bit for the sustain pedal which is unused). We hypothesized that one of the reasons multi-piece pretraining failed was because this amount of lookahead was not sufficient for the agent to differentiate between pieces, and so we could improve our multi-piece pretraining by increasing the length of the goal lookahead.

7.2 Methods

In order to scale up the lookahead buffer size without scaling the size of the model, and to make use of the temporal nature of the goal, we replaced the MLPs in the actor and critic networks with 3-layer BERT-style transformer encoders [9]. We used a hidden dimension of 128 and 2 attention heads. The environment state, current goal and goal lookahead are projected into the transformer's input dimension and arranged into a sequence to be fed into the transformer as shown in Figure 2. We use the output of the transformer at the first sequence position as the embedding of the agent's observation.

Because most lookahead timesteps are identical, we convolve the transformer inputs in the time dimension with kernel size = stride = 10 before computing the positional embeddings to reduce the input sequence to 10 vectors each representing 0.5s of lookahead and one vector representing the environment state.

We also introduced a lookahead for the fingering goal, which is a 10-dimensional binary vector representing which of the agent's 10 simulated fingers should be down at a given timestep, and include this as input to the transformer by summing the embeddings of the goal lookahead and fingering lookahead immediately before the transformer layers.

To try to disentangle the effects of the longer lookahead and the architecture changes, we increased the lookahead length for the MLP by adding in the 1D convolutions from the transformer architecture, and reported those results as MLP+Conv.

To evaluate the transformer architecture, we trained an agent on French Suite No. 1 Allemande from ROBOPIANIST-etudes-12. To test our hypothesis and see if transformer architectures were able to learn multiple pieces at once, we perform multi-piece pretraining as described in section 4.2.

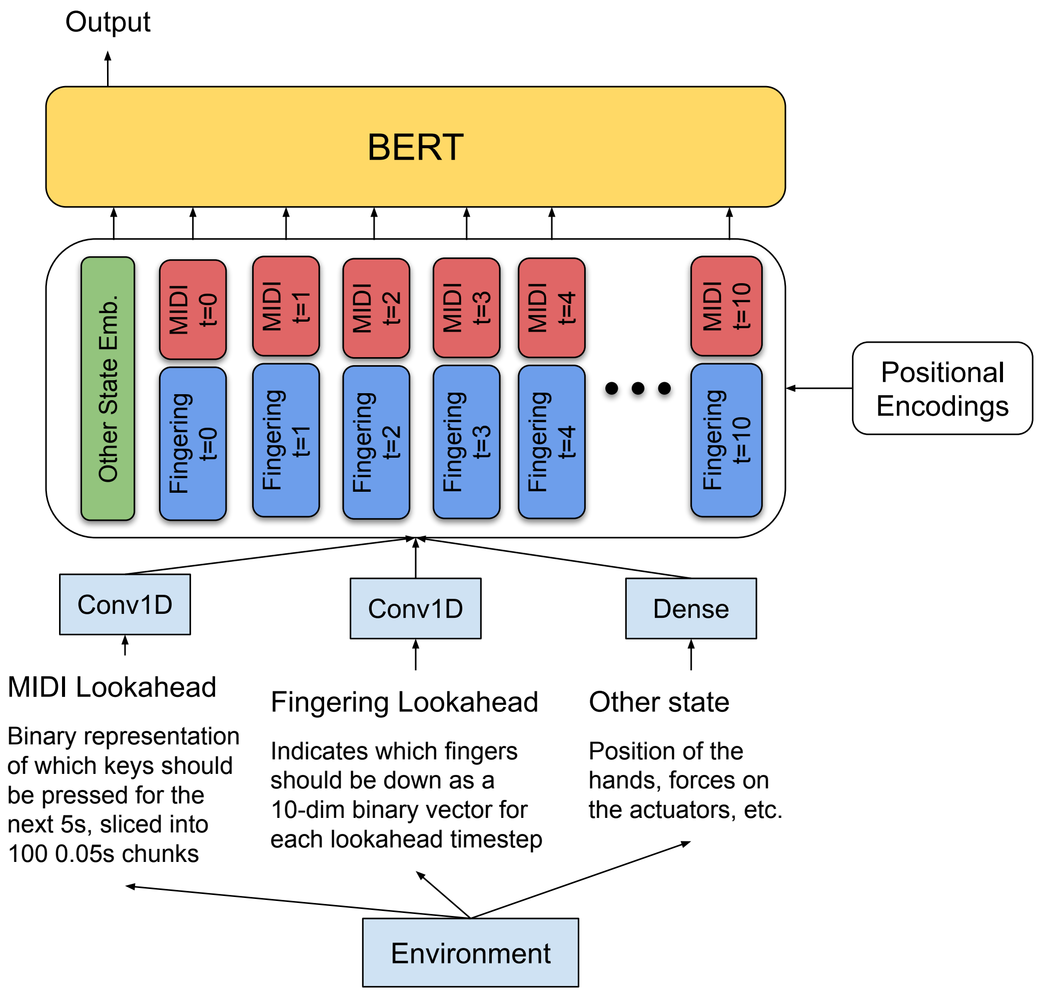

Figure 2: The transformer-based architecture for the actor & critic encoders. The first element in the sequence of transformer inputs is an embedding of the environment state (the positions of the hands, the forces on the actuators, etc.), followed by an embedding of the current goal ($t = 0$), followed by 10 embeddings each representing 0.5s of goal lookahead ($t > 0$).

7.3 Results

Architecture	Lookahead Timesteps	Key Reward Tiering	# Actor Params	F_1@100k Steps	F_1@1M Steps
MLP	10	No	477k	0.017	0.496
MLP	10	Yes	477k	0.132	**0.648**
Transformer	10	Yes	834k	0.104	0.290
MLP+Conv	10	Yes	329k	**0.187**	0.371
Transformer	100	No	834k	0.124	0.155
MLP+Conv	100	No	557k	0.085	0.120
MLP	100	Yes	2.76M	0.180	0.289
Transformer	100	Yes	834k	**0.220**	**0.372**
MLP+Conv	100	Yes	557k	0.200	0.305

Table 4: Results when evaluated on French Suite No. 1 Allemande with 10 and 100 steps of lookahead. Among the runs with 10 lookahead timesteps Transformer and MLP+Conv architectures perform better after just 100k training steps, however they stop learning more quickly than and are eventually beaten out by the MLP architecture. The Transformer is the only architecture that benefits from increased lookahead timesteps, and it beats out the MLP architecture when both are given 100 steps of lookahead. However, the MLP with 10 steps of lookahead remains our best model architecture when given the full 1M training steps.

Transformer and MLP+Conv outperform the baseline architecture after 100k steps, but are not as strong when trained for the full 1 million steps. We found that Key Reward Tiering is necessary for transformer and MLP+Conv encoders to achieve a good F1 score. Without Key Reward Tiering, the transformer network focuses too much on optimizing for the hand positioning and energy penalties and does not interact with the piano enough for learning to progress quickly.

For the multi-piece pretraining, each transformer run takes about 8 hours, so fine-tuning the transformer on each etude ended up not being possible. However we did fine-tune French Suite No 1. Allemande using the pretrained MLP and Transformers and got about the same result as a randomly-initialized model. In addition, we evaluated the MLP- and Transformer-based models in a 0-shot setting, and found that the Transformer model was able to achieve some F_1 score on each of the etudes, though the 0-shot performance is still quite poor. The results of the 0-shot evaluations are shown in Table 5. For the two pieces where MLP performed better, Golliwoggs Cakewalk and Piano Sonata D845 1St Mov, we find that the F_1 score the pretrained MLP network achieved was unchanged from its random initialization. If we remove those two pieces, the transformer network performs significantly better in the 0-shot regime. This supports our hypothesis that using a longer context length and making use of the sequential nature of the data may be important for an agent to exhibit multi-piece learning.

Piece	**MLP lookahead=10**	**Transformer lookahead=100**
Bagatelle Op3 No4	0.070	**0.091**
French Suite No 1 Allemande	0.000	**0.132**
French Suite No 5 Gavotte	0.000	**0.082**
French Suite No 5 Sarabande	0.000	**0.109**
Golliwoggs Cakewalk	**0.258**	0.045
Kreisleriana Op 16 No 8	0.001	**0.044**
Paritita No 26	0.000	**0.120**
Piano Sonata D845 1St Mov	**0.209**	0.001
Piano Sonata K279 In C Major 1St Mov	0.050	**0.074**
Piano Sonata No 2 1St Mov	0.003	**0.023**
Piano Sonata No 23 2Nd Mov	0.000	**0.084**
Waltz Op 64 No 1	0.000	**0.087**
Average	0.049 ± 0.057	$\mathbf{0.075 \pm 0.024}$
Average minus Gollwogs Cakewalk & Sonata D845	0.012 ± 0.018	$\mathbf{0.085 \pm 0.023}$

Table 5: 0-shot F_1 scores of pieces in ROBOPIANIST-etude-12 after 1M steps of multi-piece pretraining.

8 Conclusions and Future Work

We found that pretraining our agent on major scale environments gave an improvement over our baseline measurements. However our agents pretrained on the PIG dataset never learned to generalize properly. We made multiple efforts to enable pretraining on the PIG dataset, including applying Hindsight Experience Relabelling (HER), modifying the environment's reward function, and replacing the MLP in the actor and critic with a transformer-encoder-based network. We found that Key Reward Tiering yielded meaningful improvements to the agent's learning by encouraging it to interact with the piano. While our the baseline pretrained model achieved a F_1 score of 0.0 on half of the environments in ROBOPIANIST-etudes-12, combining Key Reward Tiering with the transformer architecture using a longer lookahead length yielded a small 0-shot capability. However this only addresses a tiny fraction of the gap between the agent's 0-shot performance and the fine-tuned performamce.

Improving performance on this task was much more challenging than anticipated. Our initial ambitions of implementing MT-OPT [10], a modern multi-task training system, were quickly dashed when we found that hindsight relabelling, the transformer architecture and multi-scale pretraining did not work nearly as well as expected. We then spent a significant amount of time trying to tune the system before attempting Key Reward Tiering, which did seem to "unlock" some of the other approaches. We ran out of time to explore this further, but we believe that continuing to tune the rewards could unlock other RL methods such as the ones we've seen in class. Overall, this is a very challenging task since getting the action right at each timestep is crucial (and not just at the end of the episode like for many other tasks).

9 Team Member Contributions

Eric implemented the initial setup of the robopianist environment & baseline training code. Eric also implemented HER and the tiered rewards.

Matt implemented the pretraining approaches and the transformer architecture.

We would like to thank Kevin Zakka for helping us reproduce the results from [1].

References

[1] Kevin Zakka, Laura Smith, Nimrod Gileadi, Taylor Howell, Xue Bin Peng, Sumeet Singh, Yuval Tassa, Pete Florence, Andy Zeng, and Pieter Abbeel. Robopianist: A benchmark for high-dimensional robot control, 2023.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018.

[3] Eita Nakamura, Yasuyuki Saito, and Kazuyoshi Yoshii. Statistical learning and estimation of piano fingering. *CoRR*, abs/1904.10237, 2019.

[4] Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices, 2021.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[6] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[7] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[8] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[10] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021.