PianoMime: Learning a Generalist, Dexterous Piano Player from Internet Demonstrations

Cheng Qian*, Julen Urain†, Kevin Zakka‡, Jan Peters†
*Technical University of Munich
†Technical University of Darmstadt
‡University of California Berkeley

***Abstract*—In this work, we introduce PianoMime, a framework for training a piano-playing agent using internet demonstrations. The internet is a promising source of large-scale demonstrations for training our robot agents. In particular, for the case of piano-playing, Youtube is full of videos of professional pianists playing a wide myriad of songs. In our work, we leverage these demonstrations to learn a generalist piano-playing agent capable of playing any arbitrary song. Our framework is divided into three parts: a data preparation phase to extract the informative features from the Youtube videos, a policy learning phase to train song-specific expert policies from the demonstrations and a policy distillation phase to distil the policies into a single generalist agent. We explore different policy designs to represent the agent and evaluate the influence of the amount of training data on the generalization capability of the agent to novel songs not available in the dataset. We show that we are able to learn a policy with up to 56% F1 score on unseen songs. Project website: https://pianomime.github.io/**

I. Introduction

The Internet is a promising source of large-scale data for training generalist robot agents. If properly exploited, it is full of demonstrations (video, text, audio) of humans solving an infinite amount of tasks [23, 3, 5] that could inform our robot agents on how to behave. However, learning from these databases is challenging for several reasons. First, unlike teleoperation demonstrations, video data does not specify the actions applied by the robot, usually requiring the use of reinforcement learning to induce the robot actions [22, 3, 13]. Second, videos typically show a human performing the task, while the learned policy is deployed on a robot. This often requires to re-target the human motion to the robot body [13, 4, 14]. Finally, as pointed in [3], if we aim to learn a generalist agent, we must select a task for which large-scale databases are available and that allows an unlimited variety of open-ended goals.

From opening doors [4] to rope manipulation [12] or pick and place tasks [20, 11], previous works have successfully taught robot manipulation skills through observations. However, these approaches have been limited to low dexterity in the robots or to a small variety of goals.

In this work, we focus on the task of **learning a generalist piano player from Internet demonstrations**. Piano-playing is a highly dexterous open-ended task [24]. Given two dexterous robot hands and a desired song, the goal of a piano-playing agent is to press the correct keys and only the correct keys at the proper timing. Moreover, the task can be conditioned on arbitrary songs, allowing for a large, and high-dimensional goal conditioning.

To learn a generalist piano-playing agent from internet data, we introduce **PianoMime**, a framework to train a single policy capable of playing any song (See Figure 1). In its essence, the PianoMime agent is a goal-conditioned policy that generates configuration space actions given the song to be played. At each timestep, the agent receives as goal input a trajectory of the keys to be pressed. Then, the policy generates a trajectory of actions and executes them in chunk. **To learn the agent,** we combine both reinforcement learning with imitation learning. We train individual song-specific expert policies by using reinforcement learning in conjunction with Youtube demonstrations and we distill all the expert policies into a single generalist behavioral cloning policy. **To represent the agent,** we perform ablations of different architectural design strategies to model the behavioral cloning policy. We investigate the benefit of incorporating representation learning to enhance the geometric information of the goal input. Additionally, we explore the effectiveness of a hierarchical policy that combines a high-level policy generating fingertip trajectories with a learned *cross-domain inverse dynamics model* generating joint-space actions. We show that the learned agent is able to play arbitrary songs not included in the training dataset with an average 56% F1-score.

In summary, **the main contribution** of this work is a framework for training a generalist piano-playing agent using Internet demonstration data. To achieve this goal, we:

- Introduce a method to learn policies from the internet demonstrations by decoupling the human motion information from the task-related information.
- Present a reinforcement learning approach that combines residual policy learning [21, 9] with style rewards [13].
- Explore different policy designs, introducing novel strategies to learn geometrically consistent latent goal representations and conducting ablations on different designs and dataset sizes.

Finally, we are releasing the dataset and the trained models as a benchmark for testing internet-data-driven dexterous manipulation.

II. Method

The PianoMime framework is composed of three phases: data preparation, policy learning, and policy distillation. In

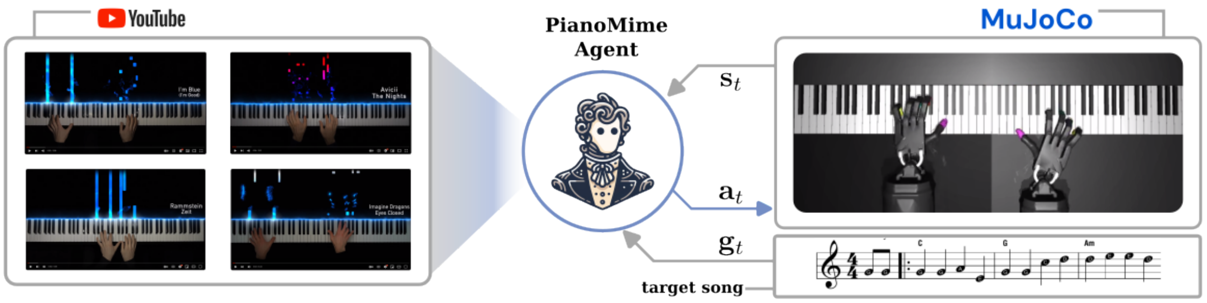

Fig. 1: The goal of this work is to train a generalist piano-playing agent (PianoMime) from Youtube videos. We collect a set of videos and accompanying MIDI files and train a single agent to play any song, combining reinforcement learning and behavioral cloning.

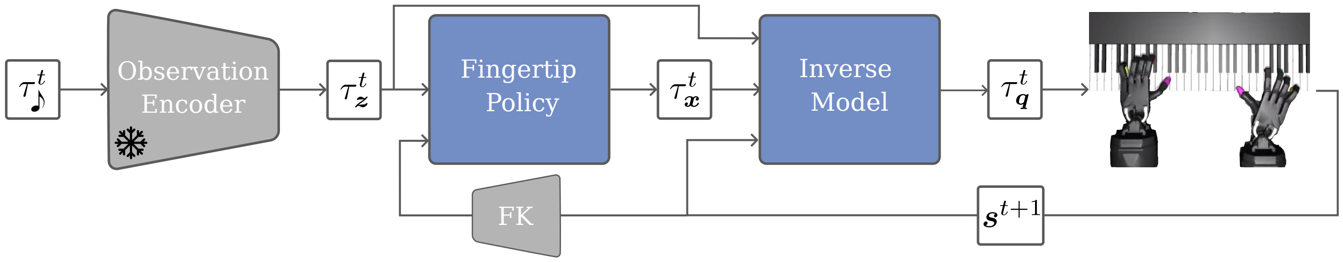

Fig. 2: Proposed distillation policy architecture. Given a L steps window of a target song $\tau^t_{♪} : (♪_{t:t+L})$ at time t, a latent representation τ^t_z is computed given a pre-trained observation encoder. Then, the policy is decoupled between a high-level fingertip predictor that generates a trajectory of fingertip positions τ^t_x and a low-level inverse dynamics model that generates a trajectory of target joint position τ^t_q.

data preparation phase, given the raw video demonstration, we extract the informative signals needed to train the policies. In **policy learning phase**, we train song-specific policies via Reinforcement Learning (RL). This step is essential for generating the robot actions that are missing in the demonstrations. The policy is trained with two reward functions: a style reward and a task reward. In **policy distillation phase**, we train a single behavioral cloning policy to mimic all song-specific policies. The goal of this phase is to train a single generalist policy capable of playing any song.

A. Data preparation: From raw data to human and piano state trajectories

We generate the training dataset by web scraping YouTube videos of professional piano artists playing various songs. We select YouTube channels that also upload MIDI files of the played songs, which represent trajectories of the piano keys state throughout the song. We use the video to extract the motion of human pianists and the MIDI file to inform about the goal state of piano during performance.

We focus on the fingertip position as the signal for the robot hand to mimic. Although some tasks might require using the palm (e.g., grasping), we find that mimicking fingertip motion is sufficient for playing the piano. This also reduces constraints on the robot, allowing for more adaptable movements.

We use MediaPipe [10] to extract fingertip motion from videos. MediaPipe outputs the hand skeleton from each video frame. We found that the typical top-view recording in piano-playing YouTube videos is particularly useful for accurately estimating fingertip positions. Notice that given the videos are RGB, we lack depth signal. Therefore, we predict the 3D fingertip positions based on the piano state. The detailed procedure is explained in Appendix A.

B. Policy learning: generating robot actions from observations

Through the data preparation phase, we extract two trajectories: a human fingertip trajectory $\boldsymbol{\tau}_x$ and a piano state trajectory $\boldsymbol{\tau}_{♪}$. The human fingertip trajectory $\boldsymbol{\tau}_x : (x_1, \ldots, x_T)$ is a T-step trajectory of two hands' 3D fingertip positions $x \in R^{3\times 10}$ (10 fingers). The piano state trajectory $\boldsymbol{\tau}_{♪} : (♪_1, \ldots, ♪_T)$ is a T-step trajectory of piano states $♪ \in B^{88}$, represented with an 88-dimensional binary variable representing which keys should be pressed.

Given the ROBOPIANIST [24] environment, **our goal** is to learn a goal-conditioned policy $\pi_{\boldsymbol{\theta}}$ that plays the song defined by $\boldsymbol{\tau}_{♪}$ while matching the fingertip motion given by $\boldsymbol{\tau}_x$ as much as possible.

Similarly to [24], we formulate the piano playing as an **Markov Decision Process (MDP)**. The state observation includes the robot's proprioception s and the goal state g_t. The goal state g_t at time t informs the desired piano key configurations $♪$ in the future $g_t = (♪_{t+1}, \ldots, ♪_{t+L})$, with L being the lookahead horizon. The action a is defined as the desired configuration for both hands $q \in R^{23\times 2+1}$, each with 23 joint angles and one dimension for the sustain pedal.

We propose solving the reinforcement learning problem by combining residual policy learning [21, 9, 4] and style mimicking rewards [13, 15].

Residual policy architecture. Given the fingertip trajectory $\boldsymbol{\tau}_x$, we solve an Inverse Kinematics (IK) [1] problem to obtain a trajectory of desired joint angles $\boldsymbol{\tau}_q^{\text{ik}} : (q_0^{\text{ik}}, \dots, q_T^{\text{ik}})$ for the robot hands. Then, we represent the policy $\pi_{\boldsymbol{\theta}}(a|s, g_t) = \pi_{\boldsymbol{\theta}}^r(a|s, g_t) + q_{t+1}^{\text{ik}}$ as a combination of a nominal behavior (given by the IK solution) and a residual policy $\pi_{\boldsymbol{\theta}}^r$. Given the goal state at time t, the nominal behavior is defined as the next desired joint angle q_{t+1}^{ik}. We then only learn the residual term around the nominal behavior.

Style-mimicking reward. We integrate a style-mimicking reward to preserve the human style in the trained robot actions. The reward function $r = r_{♪} + r_x$ is composed of a task reward $r_{♪}$ and a style-mimicking reward r_x. While the task reward $r_{♪}$ encourages the agent to press the correct keys, the style reward r_x encourages the agent to move its fingertips similar to the demonstration $\boldsymbol{\tau}_x$. We provide further details in Appendix C.

C. Policy distillation: learning a generalist piano-playing agent

Through the policy learning phase, we train song-specific policies from which we roll out state and action trajectories $\tau_s : (s_0, \dots, s_T)$ and $\tau_q : (q_0, \dots, q_T)$. Then, we generate a dataset $\mathcal{D} : (\tau_s^i, \tau_q^i, \tau_x^i, \tau_{♪}^i)_{i=1}^N$ with N being the number of learned songs. Given the dataset $\mathcal{D}$, we apply Behavioral Cloning (BC) to learn a single generalist piano-playing agent $\pi_{\boldsymbol{\theta}}(q_{t:t+L}, x_{t:t+L}|s_t, ♪_{t:t+L})$ that outputs configuration-space actions $q_{t:t+L}$ conditioned on the current state s_t and the future desired piano state $♪_{t:t+L}$. We explore different strategies to represent and learn the behavioral cloning policy and improve its generalization capabilities.

Representation Learning. We pre-train an observation encoder over the piano state ♪ to learn spatially consistent latent features. We hypothesize that two piano states that are spatially close should lead to latent features that are close. Using these latent features as goal should induce better generalization. To obtain the observation encoder, we train an autoencoder with a reconstruction loss over a Signed Distance Field (SDF) defined on the piano state. Specifically, the encoder compresses the binary vector of the goal into a latent space, while the decoder predicts the SDF function value of a randomly sampled query point (the Euclidean distance between the query point and the closest "on" piano key). We provide further details in Appendix E.

Hierarchical Policy. We represent the piano-playing agent with a hierarchical policy. The high-level policy receives a sequence of desired future piano states ♪ and outputs a trajectory of human fingertip positions x. Then, a low-level policy takes the fingertip and piano state trajectories as input and outputs a trajectory of desired joint angles q. On one hand, while fingertip trajectory data is easily available from the Internet, obtaining low-level joint trajectories requires solving a computationally expensive RL problem. On the other hand, while the high-level mapping ($♪ \mapsto x$) is complex, which involves fingerings, the low-level mapping ($x \mapsto q$) is relatively simpler, which addresses a cross-embodiment inverse dynamics problem. This decoupling allows us to train the more complex high-level mapping on large cheap datasets and the simpler low-level mapping on smaller expensive ones. We visualize the policy in Figure 2.

Expressive Generative Models. Considering that the human demonstration data of piano playing is highly multimodal, we explore using expressive generative models to better represent this multi-modality. We compare the performance of different deep generative models based policies, e.g., Diffusion Policies [2] and Behavioral Transformer [19], as well as a deterministic policy.

III. Experimental Results

All experiments are conducted on our collected dataset, which contains the notes, the corresponding demonstration videos and fingertip trajectories of 60 piano songs from a Youtube channel **PianoX** [1]. To standardize the length of each task, each song is divided into several clips, each with a duration of 30 seconds (The dataset contains totally 431 clips, 258K state-action pairs). Furthermore, we choose 12 unseen clips to investigate the generalization capability of the generalist policy. We use the same evaluation metrics from RoboPianist [24], i.e., precision, recall, and F1 score.

Our experiment setup utilizes RoboPianist simulation environment [24]. The agent predicts target joint angles at 20Hz and the targets are converted to torques using PD controllers running at 500Hz.

A. Evaluation on learning song-specific policies from demonstrations

In this section, we compare our method for training song-specific policies against two baselines: **Robopianist [24]** We use the RL method introduced in [24]. We manually label the fingering from the demonstrations videos to provide the fingering reward. **Inverse Kinematics (IK) [1]** Given a demonstration fingertip trajectory τ_x, an IK solver [1] is used to compute a target joint position trajectory and execute it open-loop. We provide the details of IK solver implementation in Appendix B. We select 10 clips with diverse levels of difficulty from the collected dataset. We train song-specific policies for each clip using both the baseline and our methods. We then compare their performance based on the F1 scores.

Results. As shown in Figure 3, our method consistently outperforms the Robopianist baseline for all 10 clips, achieving an average F1 score of 0.94 compared to the baseline's 0.75. We attribute this improvement to the incorporation of human priors, which narrows the RL search space to a favorable subspace, guiding the policy towards more optimal ones. Additionally, the IK method achieves an average F1 score of 0.72, only slightly lower than the baseline. This demonstrates

[1]https://www.youtube.com/channel/UCsR6ZEA0AbBhrF-NCeET6vQ

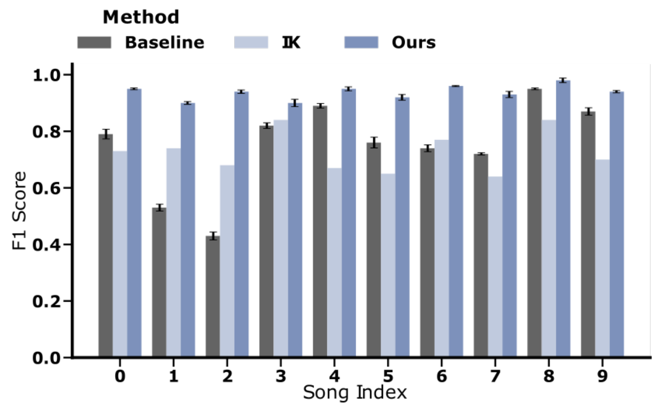

Fig. 3: The F1 score achieved by three methods for 10 clips

the effectiveness of incorporating human priors, providing a robust starting point for RL. Remarkably, we observe poor performance from the baseline, particularly in clips demanding rapid key presses across long distances.

B. Evaluation of model design strategies for policy distillation

This section focuses on evaluating policy distillation for playing different songs. We perform ablations on different policy designs. We propose a **Base** policy (Two-stage Diffusion) that incorporates both hierarchical design and goal representation learning, as described in Section II-C. Both high- and low-level policies are trained with conditional Denoising Diffusion Probabilistic Models (DDPM) [8]. The high-level policy is trained to predict the fingertip trajectory for 4 timesteps given the SDF embedding of goals over 10 timesteps, while the low-level policy predicts the robot actions for 4 timesteps given the fingertip trajectory. Note that the entire dataset is used for training the high-level policy, while only around 40 % of the collected clips (110K state-action pairs) are trained with RL and further used for training the low-level policy. The detailed network implementation is described in Appendix F.

To analyze the impact of each variable, we design four variants of the Two-stage Diffusion policy. To evaluate (**1**) the impact of integrating a pre-trained observation encoder, we train a model without the SDF embedding representation for the goal (**w/o SDF**). To evaluate (**2**) the impact of the hierarchical architecture, we train a **One-stage** Diffusion policy that directly predict the robot actions. Finally, to evaluate (**3**) the influence of using different generative models, we train a Two-stage **BeT**, that replaces Diffusion models with Behavior-Transformers [19]. We also consider as baselines a Multi-task **RL** policy and a **BC** policy with MSE Loss. We provide further details of the models in Appendix G.

Results As shown in Table I, despite that Multi-task RL has the highest precision on the test dataset (this is because it barely presses any keys), our base policy outperforms the others in all metrics on both training and test datasets. We also observe that the incorporation of SDF embedding for goal representation leads to better performance, especially on the test dataset, which demonstrates the impact of goal representation on policy generalization.

		RL	BC	**Base**	w/o SDF	One-Stage	BeT
Train	P	0.85	0.56	**0.87**	0.86	0.53	0.63
	R	0.20	0.29	**0.78**	0.76	0.34	0.42
	F1	0.12	0.30	**0.81**	0.78	0.35	0.49
Test	P	**0.95**	0.54	0.69	0.66	0.58	0.53
	R	0.18	0.22	**0.54**	0.49	0.27	0.30
	F1	0.13	0.21	**0.56**	0.51	0.26	0.31

TABLE I: Quantitative results evaluated on Training and Test Datasets. Test datasets consist of 12 clips unseen in the training dataset. We report Precision (P), Recall (R) and F1-score (F1).

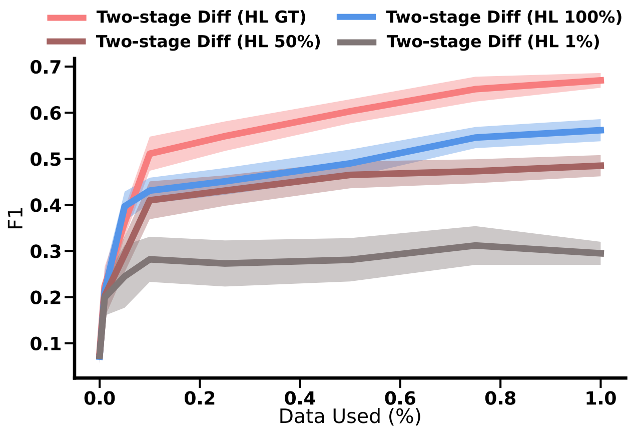

Fig. 4: F1 scores of the base policies trained with different proportions of high-level and low-level datasets. The x-axis represents the percentage of the low-level dataset utilized, while HL % indicates the percentage of the high-level dataset used.

C. Evaluations on the impact of the data in the generalization

In this section, we investigate the impact of scaling training data for high-level and low-level policies on the generalization capabilities of the agent. We employ different combinations of the high-level and low-level policies of the base policy trained with different proportions of the dataset and assess their performance. In addition, we introduce an oracle high-level policy, which outputs the ground-truth fingertip position from human demonstration videos.

Results The results (see Figure 4) demonstrate that the overall performance of policy is significantly influenced by the quality of the high-level policy. Low-level policies paired with Oracle high-level policies consistently outperform the ones paired with other high-level policies. Besides, we observe early performance convergence with increasing training data when paired with a low-quality high-level policy. Specifically, with the HL 1% policy and HL 50%, performance almost converged with around 10% and 50% low-level data, respectively.

IV. Conclusion

In this work, we present PianoMime, a framework for training a generalist robotic pianist using internet video sources. We start by training song-specific policies with residual RL, enabling the robot to master individual songs by mimicking human pianists. Subsequently, we train a single behavioral cloning policy that mimics these song-specific policies to play unseen songs. We show that goal representation learning, policy hierarchy, and expressive generative models are key factors to enhance the policy generalization. The resulting policy achieces an average F1-score of 56% on unseen songs.

Acknowledgments

We are extremely grateful to Jan Brüdigam and Stefan Sosnowski from the Chair of Information-oriented Control, Technical University of Munich for their advicing. We would also like to thank the YouTube channel PianoX for providing excellent demonstration videos on piano playing.

References

[1] Stéphane Caron, Yann De Mont-Marin, Rohan Budhiraja, and Seung Hyeon Bang. Pink: Python inverse kinematics based on Pinocchio, 2024. URL https://github.com/stephane-caron/pink.

[2] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[3] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

[4] Guillermo Garcia-Hernando, Edward Johns, and Tae-Kyun Kim. Physics-based dexterous manipulations with estimated hand poses and residual reinforcement learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9561–9568. IEEE, 2020.

[5] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012, 2022.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[7] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[9] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[10] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

[11] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022.

[12] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2146–2153. IEEE, 2017.

[13] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

[14] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *Robotics: Science and Systems (RSS)*, 2020.

[15] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

[16] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[17] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[19] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[20] Lin Shao, Toki Migimatsu, Qiang Zhang, Karen Yang, and Jeannette Bohg. Concept2robot: Learning manipulation concepts from instructions and human demonstrations. *The International Journal of Robotics Research*, 40(12-14):1419–1434, 2021.

[21] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie

Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[22] Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*, 2019.

[23] Michael Völske, Martin Potthast, Shahbaz Syed, and Benno Stein. Tl; dr: Mining reddit to learn automatic summarization. In *Proceedings of the Workshop on New Frontiers in Summarization*, pages 59–63, 2017.

[24] Kevin Zakka, Philipp Wu, Laura Smith, Nimrod Gileadi, Taylor Howell, Xue Bin Peng, Sumeet Singh, Yuval Tassa, Pete Florence, Andy Zeng, et al. Robopianist: Dexterous piano playing with deep reinforcement learning. In *Conference on Robot Learning*, pages 2975–2994. PMLR, 2023.

Appendix

A. Retargeting: From human hand to robot hand

To retarget from the human hand to robot hand, we follow a structured process.

Step 1: Homography Matrix Computation Given a top-view piano demonstration video, we firstly choose n different feature points on the piano. These points could be center points of specific keys, edges, or other identifiable parts of the keys that are easily recognizable (See Figure 5). Due to the uniform design of pianos, these points represent the same physical positions in both the video and Mujoco. Given the chosen points, we follow the Eight-point Algorithm to compute the Homography Matrix H that transforms the pixel coordinate in videos to the x-y coordinate in Mujoco (z-axis is the vertical axis).

Step 2: Transformation of Fingertip Trajectory We then obtain the human fingertip trajectory with MediaPipe [10]. We collect the fingertips positions every 0.05 seconds. Then we transform the human fingertip trajectory within pixel coordinate into the Mujoco x-y 2D coordinate using the computed homography matrix H.

Step 3: Heuristic Adjustment for Physical Alignment We found that the transformed fingertip trajectory might not physically align with the notes, which means there might be no detected fingertip that physically locates at the keys to be pressed or the detected fingertip might locate at the border of the key (normally human presses the middle point of the horizontal axis of the key). This misalignment could be due to the inaccuracy of the hand-tracking algorithm and the homography matrix. Therefore, we perform a simple heuristic adjustment on the trajectory to improve the physical alignment. Specifically, at each timestep of the video, we check whether there is any fingertip that physically locates at the key to be pressed. If there is, we adjust its y-axis value to the middle point of the corresponding key. Otherwise, we search within a small range, specifically the neighboring two keys, to find the nearest fingertip. If no fingertip is found in the range or the found fingertip has been assigned to another key to be pressed, we then leave it. Otherwise, we adjust its y-axis value to the center of the corresponding key to ensure proper physical alignment.

Step 4: Z-axis Value Assignment Lastly, we assign the z-axis value for the fingertips. For the fingertips that press keys, we set their z-axis values to 0. For other fingertips, we set their z-axis value to $2 \cdot h_{key}$, where h_{key} is the height of the keys in Mujoco.

B. Implementation of Inverse Kinematics Solver

The implementation of the IK solver is based on the approach of [1]. The solver addresses multiple tasks simultaneously by formulating an optimization problem and find the optimal joint velocities that minimize the objective function. The optimization problem is given by:

$$\min_{\dot{q}} \sum_i w_i \| J_i \dot{q} - K_i v_i \|^2, \tag{1}$$

where w_i is the weight of each task, K_i is the proportional gain and v_i is the velocity residual. We define a set of 10 tasks, each specifying the desired position of one of the robot fingertips. We do not specify the desired quaternions. All the weights w_i are set to be equal. We use quadprog [2] to solve the optimization problem with quadratic programming. The other parameters are listed in Table II.

TABLE II: The parameters of IK solver

Parameter	Value
Gain	1.0
Limit Gain	0.05
Damping	1e-6
Levenberg-Marquardt Damping	1e-6

C. Detailed MDP Formulation of Song-specific Policy

Table III, Table IV and Table V show the reward function, observation space and action space of the MDP formulation of song-specific policies, respectively.

D. Training Details of Song-specific Policy

We use PPO [18] (implemented by StableBaseline 3 [17]) to train the song-specific policy with residual RL(See Algorithm 1). All of the experiments are conducted using the same network architecture and tested using 3 different seeds. Both actor and critic networks are of the same architecture, containing 2 MLP hidden layers with 1024 and 256 nodes, respectively, and GELU [7] as activation functions. The detailed hyperparameters of the networks are listed in Table VII.

[2]https://github.com/quadprog/quadprog

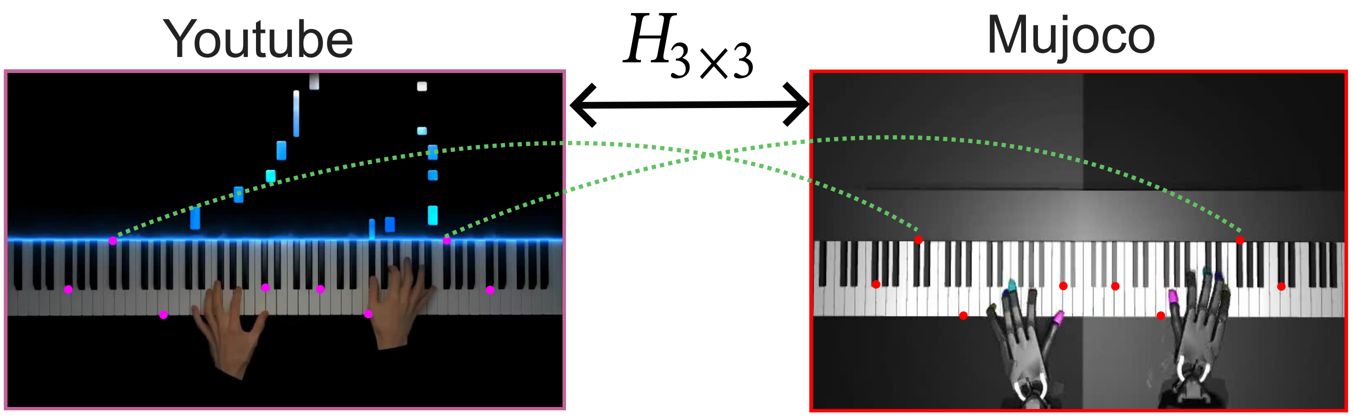

Fig. 5: Compute homography matrix given 8 correspondence feature points.

TABLE III: The detailed reward function to train the song-specific policy. The Key Press reward is the same as in [24], where k_s and k_g represent the current and the goal states of the key respectively, and g is a function that transforms the distances to rewards in the [0, 1] range. p_{df} and p_{rf} represent the fingertip positions of human demonstrator and robot respectively.

Reward	Formula	Weight	Explanation
Key Press	$0.5 \cdot g(\|k_s - k_g\|_2) + 0.5 \cdot (1 - \mathbf{1}_{\text{false positive}})$	2/3	Press the right keys and only the right keys
Mimic	$g(\|p_{df} - p_{rf}\|_2)$	1/3	Mimic the demonstrator's fingertip trajectory

TABLE IV: The observation space of song-specific agent.

Observation	Unit	Size
Hand and Forearm Joint Positions	Rad	52
Hand and forearm Joint Velocities	Rad/s	52
Piano Key Joint Positions	Rad	88
Piano key Goal State	Discrete	88
Demonstrator Forearm and Fingertips Cartesian Positions	m	36
Prior control input $\tilde{u}$ (solved by IK)	Rad	52
Sustain Pedal state	Discrete	1

TABLE V: The action space of song-specific agent.

Action	Unit	Size
Target Joint Positions	Rad	46
Sustain Pedal	Discrete	1

E. Representation Learning of Goal

We train an autoencoder to learn a geometrically continuous representation of the goal (See Figure 6 and Algorithm 2). During the training phase, the encoder $\mathcal{E}$, encodes the original 88-dimensional binary representation of a goal piano state $♪_t$ into a 16-dimensional latent code z. The positional encoding of a randomly sampled 3D query coordinate x is then concatenated with the latent code z and passed through the decoder $\mathcal{D}$. We use positional encoding here to represent the query coordinate more expressively. The decoder is trained to predict the SDF $f(x, ♪_t)$. We define the SDF value of x with respect to $♪_t$ as the Euclidean distance between the x and the nearest key that is supposed to be pressed in $♪_t$, mathematically expressed as:

$$\text{SDF}(x, ♪_t) = \min_{p \in \{p_i | ♪_{t,i}=1\}} \|x - p\|, \tag{2}$$

where pi represents the position of the i-th key on the piano. The encoder and decoder are jointly optimized to minimize the reconstruction loss:

$$L(x, , ♪_t) = (\text{SDF}(x, ♪_t) - \mathcal{D}(\mathcal{E}(v, x)))^2. \tag{3}$$

TABLE VI: The Hyperparameters of PPO

Hyperparameter	Value
Initial Learning Rate	3e-4
Learning Rate Scheduler	Exponential Decay
Decay Rate	0.999
Actor Hidden Units	1024, 256
Actor Activation	GELU
Critic Hidden Units	1024, 256
Critic Activation	GELU
Discount Factor	0.99
Steps per Update	8192
GAE Lambda	0.95
Entropy Coefficient	0.0
Maximum Gradient Norm	0.5
Batch Size	1024
Number of Epochs per Iteration	10
Clip Range	0.2
Number of Iterations	2000
Optimizer	Adam

Algorithm 1 Training of the song-specific policy with residual RL

```
1: Initialize actor network $\pi_\theta$
2: Initialize critic network $v_\phi$
3: for $i = 1 : N_{iteration}$ do
4:     # Collect trajectories
5:     for $t = 1 : T$ do
6:         Get human demonstrator fingertip position $x_t$ and observation $o_t$
7:         Compute the prior control signal that tracks $x_t$ with the IK controller $\tilde{u}_t = ik(x_t, o_t)$
8:         Run policy to get the residual term $r_t = \pi_\theta(o_t)$
9:         Compute the adapted control signal $u_t = \tilde{u}_t + r_t$
10:        Execute $u_t$ in environment and collect $s_t, u_t, r_t, s_{t+1}$
11:    end for
12:    # Update networks
13:    for $n = 1 : N$ do
14:        Sample a batch of transitions $\{(s_j, u_j, r_j, s_{j+1})\}$ from the collected trajectories
15:        Update the actor and critic network with PPO
16:    end for
17: end for
```

We pre-train the autoencoder using the **GiantMIDI** dataset [3], which contains 10K piano MIDI files of 2,786 composers. The pre-trained encoder maps the ♪$_t$ into the 16-dimensional latent code, which serves as the latent goal for behavioral cloning. The encoder network is composed of four 1D-convolutional layers, followed by a linear layer. Each successive 1D-convolutional layer has an increasing number of filters, specifically 2, 4, 8, and 16 filters, respectively. All convolutional layers utilize a kernel size of 3. The linear layer transforms the flattened output from the convolutional layers into a 16-dimensional latent code. The decoder network is a MLP with 2 hidden layers, each with 16 neurons. We train the autoencoder for 100 epochs with a learning rate of $1e-3$.

F. Training Details of Diffusion Model

All the diffusion models utilized in this work, including One-stage Diff, the high-level and low-level policies of Two-stage Diff and Two-stage Diff w/o SDF, share the same network architecture. The network architecture are the same as the U-net diffusion policy in [2] and optimized with DDPM [8], except that we use temporal convolutional networks (TCNs) as the observation encoder, taking the concatenated goals (high-level policy) or fingertip positions (low-level policy) of several timesteps as input to extract the features on temporal dimension. Each level of U-net is then conditioned by the outputs of TCNs through FiLM [16]. High-level policies take the goals over 10 timesteps and the current fingertip position as input and predict the human fingertip positions. In addition, we add a standard gaussian noise on the current fingertip position during training to facilitate generalization. We further adjust the y-axis value of the fingertips pressing the keys in the predicted high-level trajectories to the midpoint of the keys. This adjustment ensures closer alignment with the data distribution of the training dataset. Low-level policies take the predicted fingertip positions and the goals over 4 timesteps, the proprioception state as input predict the robot actions. The proprioception state includes the robot joint positions and velocities, as well as the piano joint positions. We use 100 diffusion steps during training. To achieve high-quality results during inference, we find that at least 80 diffusion steps are required for high-level policies and 50 steps for low-level policies.

[3]https://github.com/bytedance/GiantMIDI-Piano

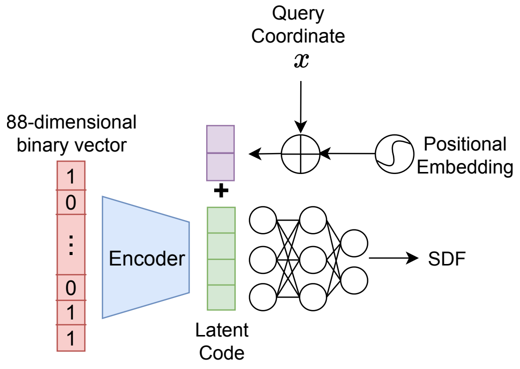

Fig. 6: 1) Encoding: The encoder compresses the binary representation of the goal into latent code. 2) Decoding: A 3D query coordinate x is randomly sampled. A neural network predicts the SDF value given the positional encoding of x and the latent code.

Algorithm 2 Training of the goal autoencoder

```
1: Initialize encoder $\mathcal{E}_\phi$
2: Initialize decoder $\mathcal{D}_\psi$
3: for $i = 1 : N_{epoch}$ do
4:     for $j = 1 : N_{batch}$ do
5:         for each goal $v$ in batch do
6:             Compute the latent code $\mathbf{z} = \mathcal{E}_\psi(♪_t)$
7:             Sample a 3D coordinate as query $\mathbf{x}$ = Sample3DCoordinate()
8:             Compute the positional encoding of query $\mathbf{pe}$ = PositionalEncoding($x$)
9:             Compute the output of the decoder conditioned by the query $\mathcal{D}_\phi(z, pe)$
10:            Compute the SDF value of query SDF($x, ♪_t$)
11:            Compute the reconstruction loss $L$
12:        end for
13:        Compute the sum of the loss
14:        Compute the gradient
15:        Update network parameter $\phi, \psi$
16:    end for
17: end for
```

TABLE VII: The Hyperparameters of DDPM

Hyperparameter	Value
Initial Learning Rate	1e-4
Learning Rate Scheduler	Cosine
U-Net Filters Number	256, 512, 1024
U-Net Kernel Size	5
TCN Filters Number	32, 64
TCN Kernel Size	3
Diffusion Steps Number	100
Batch Size	256
Number of Iterations	800
Optimizer	AdamW
EMA Exponential Factor	0.75
EMA Inverse Multiplicative Factor	1

G. *Policy Distillation Experiment*

Two-stage Diff w/o SDF We directly use the binary representation of goal instead of the SDF embedding representation to condition the high-level and low-level policies.

Two-stage BeT We train both high-level and low-level policies with Behavior Transformer [19] instead of DDPM. The hyperparameter of Bet is listed in Table VIII.

One-stage Diff We train a single diffusion model to predict the robot actions given the SDF embedding representation of goals and the proprioception state.

Multi-task RL We create a multi-task environment where for each episode a random song is sampled from the dataset. Consequently, we use Soft-Actor-Critic (SAC) [6] to train a single agent within the environment. Both the actor and critic networks are MLPs, each with 3 hidden layers, and each hidden layer contains 256 neurons. The reward function is the same as that in [24].

BC-MSE We train a feedforward network to predict the robot action of next timestep conditioned on the binary representation of goal and proprioception state with MSE loss. The feedforward network is a MLP with 3 hidden layers, each with 1024 neurons.

TABLE VIII: The Hyperparameters of Behavior Transformer

Hyperparameter	Value
Initial Learning Rate	3e-4
Learning Rate Scheduler	Cosine
Number of Discretization Bins	64
Number of Transformer Heads	8
Number of Transformer Layers	8
Embedding Dimension	120
Batch Size	256
Number of Iterations	1200
Optimizer	AdamW
EMA Exponential Factor	0.75
EMA Inverse Multiplicative Factor	1

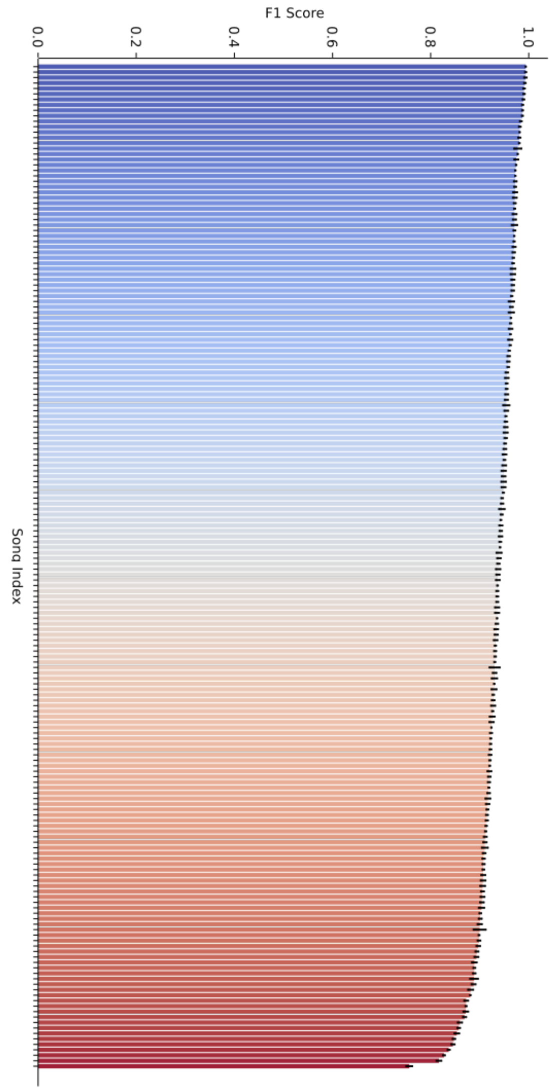

Fig. 7: F1 score of all 184 trained song-specific policies (descending order)

H. *F1 Score of All Trained Song-Specific Policies*

Figure 7 shows the F1 score of all song-specific policies we trained.

I. *Detailed Results on Test Dataset*

In Table IX and Table X, we show the Precision, Recall and F1 score of each song in our collected test dataset and the Etude-12 dataset from [24], achieved by Two-stage Diff. We observe an obvious performance degradation when testing on Etude-12 dataset. We suspect that the reason is due to out-of-distribution data, as the songs in the Etude-12 dataset are all classical, whereas our training and test dataset primarily consists of modern songs.

TABLE IX: Quantitative results of each song in the our collected test dataset

Song Name	**Precision**	**Recall**	**F1**
Forester	0.81	0.70	0.68
Wednesday	0.66	0.57	0.58
Alone	0.80	0.62	0.66
Somewhere Only We Know	0.63	0.53	0.58
Eyes Closed	0.60	0.52	0.53
Pedro	0.70	0.58	0.60
Ohne Dich	0.73	0.55	0.58
Paradise	0.66	0.42	0.43
Hope	0.74	0.55	0.57
No Time To Die	0.77	0.53	0.55
The Spectre	0.64	0.52	0.54
Numb	0.55	0.44	0.45
Mean	**0.69**	**0.54**	**0.56**

TABLE X: Quantitative results of each song in the Etude-12 dataset

Song Name	**Precision**	**Recall**	**F1**
FrenchSuiteNo1Allemande	0.45	0.31	0.34
FrenchSuiteNo5Sarabande	0.29	0.23	0.24
PianoSonataD8451StMov	0.58	0.52	0.52
PartitaNo26	0.35	0.22	0.24
WaltzOp64No1	0.44	0.31	0.33
BagatelleOp3No4	0.45	0.30	0.33
KreislerianaOp16No8	0.43	0.34	0.36
FrenchSuiteNo5Gavotte	0.34	0.29	0.33
PianoSonataNo232NdMov	0.35	0.24	0.25
GolliwoggsCakewalk	0.60	0.43	0.45
PianoSonataNo21StMov	0.32	0.22	0.25
PianoSonataK279InCMajor1StMov	0.43	0.35	0.35
Mean	**0.42**	**0.31**	**0.33**