

Figure 6: Compute homography matrix given 8 correspondence feature points.

407 A Retargeting: From human hand to robot hand

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

409 **Step 1: Homography Matrix Computation** Given a top-view piano demonstration video, we
 410 firstly choose n different feature points on the piano. These points could be center points of specific
 411 keys, edges, or other identifiable parts of the keys that are easily recognizable (See Figure 6).
 412 Due to the uniform design of pianos, these points represent the same physical positions in both
 413 the video and Mujoco. Given the chosen points, we follow the Eight-point Algorithm to compute
 414 the Homography Matrix H that transforms the pixel coordinate in videos to the x-y coordinate in
 415 Mujoco (z-axis is the vertical axis).

416

417 **Step 2: Transformation of Fingertip Trajectory** We then obtain the human fingertip tra-
 418 jectory with MediaPipe [20]. We collect the fingertips positions every 0.05 seconds. Then we
 419 transform the human fingertip trajectory within pixel coordinate into the Mujoco x-y 2D coordinate
 420 using the computed homography matrix H .

421

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

434

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

438 B Implementation of Inverse Kinematics Solver

439 The implementation of the IK solver is based on the approach of [21]. The solver addresses multiple
 440 tasks simultaneously by formulating an optimization problem and find the optimal joint velocities
 441 that minimize the objective function. The optimization problem is given by:

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

442 where w_i is the weight of each task, K_i is the proportional gain and v_i is the velocity residual. We
 443 define a set of 10 tasks, each specifying the desired position of one of the robot fingertips. We do not
 444 specify the desired quaternions. All the weights w_i are set to be equal. We use quadprog² to solve
 the optimization problem with quadratic programming. The other parameters are listed in Table 2.

Table 2: The parameters of IK solver

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

445

446 C Detailed MDP Formulation of Song-specific Policy

Table 3: The detailed reward function to train the song-specific policy. The Key Press reward is the same as in [11], 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 4: 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

447 D Training Details of Song-specific Policy

448 We use PPO [27] (implemented by StableBaseline 3 [30]) to train the song-specific policy with resid-
 449 ual RL(See Algorithm 1). All of the experiments are conducted using the same network architecture
 450 and tested using 3 different seeds. Both actor and critic networks are of the same architecture, con-
 451 taining 2 MLP hidden layers with 1024 and 256 nodes, respectively, and GELU [31] as activation
 452 functions. The detailed hyperparameters of the networks are listed in Table 7.

²<https://github.com/quadprog/quadprog>

Table 5: The action space of song-specific agent.

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

Table 6: 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

453 E Representation Learning of Goal

454 We train an autoencoder to learn a geometrically continuous representation of the goal (See Figure
 455 7 and Algorithm 2). During the training phase, the encoder \mathcal{E} , encodes the original 88-dimensional
 456 binary representation of a goal piano state \mathfrak{A}_t into a 16-dimensional latent code z . The positional
 457 encoding of a randomly sampled 3D query coordinate x is then concatenated with the latent code z
 458 and passed through the decoder \mathcal{D} . We use positional encoding here to represent the query coordinate
 459 more expressively. The decoder is trained to predict the SDF $f(x, \mathfrak{A}_t)$. We define the SDF value of
 460 x with respect to \mathfrak{A}_t as the Euclidean distance between the x and the nearest key that is supposed to
 461 be pressed in \mathfrak{A}_t , mathematically expressed as:

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

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

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

464
 465 We pre-train the autoencoder using the **GiantMIDI** dataset³, which contains 10K piano MIDI files
 466 of 2,786 composers. The pre-trained encoder maps the \mathfrak{A}_t into the 16-dimensional latent code,
 467 which serves as the latent goal for behavioral cloning. The encoder network is composed of four

³<https://github.com/bytedance/GiantMIDI-Piano>

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

```

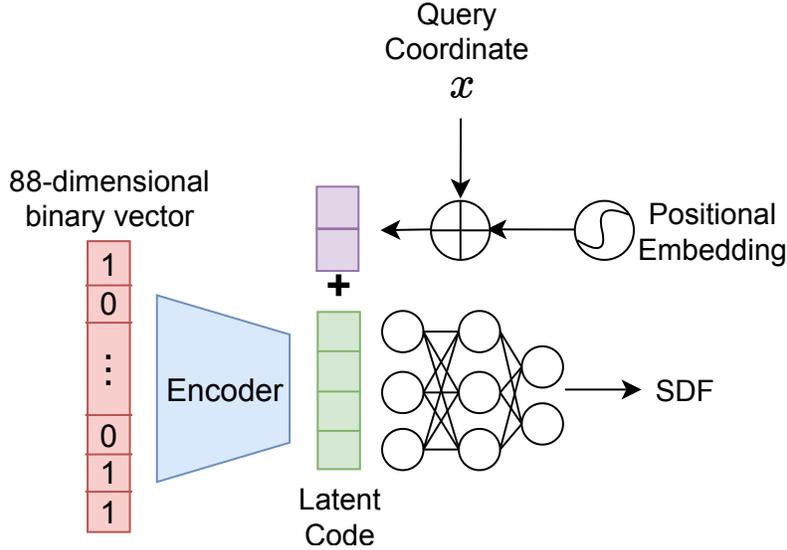


Figure 7: 1) Encoding: The encoder compresses the binary representation of the goal into latent code. 2) Decoding: A 3D query coordinate x is randomly sampled. A neural network predicts the SDF value given the positional encoding of x and the latent code.

468 1D-convolutional layers, followed by a linear layer. Each successive 1D-convolutional layer has an
 469 increasing number of filters, specifically 2, 4, 8, and 16 filters, respectively. All convolutional layers
 470 utilize a kernel size of 3. The linear layer transforms the flattened output from the convolutional
 471 layers into a 16-dimensional latent code. The decoder network is a MLP with 2 hidden layers, each
 472 with 16 neurons. We train the autoencoder for 100 epochs with a learning rate of $1e - 3$.

473 F Training Details of Diffusion Model

474 All the diffusion models utilized in this work, including One-stage Diff, the high-level and
 475 low-level policies of Two-stage Diff, Two-stage Diff-res and Two-stage Diff w/o SDF, share
 476 the same network architecture. The network architecture are the same as the U-net diffusion

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  $\mathbf{z} = \mathcal{E}_\psi(\mathcal{J}_t)$ 
7:       Sample a 3D coordinate as query  $\mathbf{x} = \text{Sample3DCoordinate}()$ 
8:       Compute the positional encoding of query  $\mathbf{pe} = \text{PositionalEncoding}(x)$ 
9:       Compute the output of the decoder conditioned by the query  $\mathcal{D}_\phi(z, \mathbf{pe})$ 
10:      Compute the SDF value of query  $\text{SDF}(x, \mathcal{J}_t)$ 
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
```

477 policy in [22] and optimized with DDPM [28], except that we use temporal convolutional net-
478 works (TCNs) as the observation encoder, taking the concatenated goals (high-level policy) or
479 fingertip positions (low-level policy) of several timesteps as input to extract the features on tem-
480 poral dimension. Each level of U-net is then conditioned by the outputs of TCNs through FiLM [32].

481

482 High-level policies take the binary representation (Two-stage Diff w/o SDF) or the SDF em-
483 bedding (Two-stage Diff, Two-stage Diff-res) of goals over 10 timesteps and the current fingertip
484 position as input and predict the fingertip positions of 4 timesteps. Besides, for training the
485 high-level policies, we add a standard gaussian noise on the current fingertip position to enhance
486 generalization. Low-level policies take the predicted fingertip positions and the goals over 4
487 timesteps, the proprioception state and the action solved by IK solver (only for Two-stage Diff-res)
488 as input and output the robot actions (One-stage Diff, Two-stage Diff, Two-stage Diff w/o SDF)
489 or the residual term of IK solver (Two-stage Diff-res). The proprioception state includes the robot
490 joint positions and velocities, as well as the piano joint positions. We use 100 diffusion steps during
491 training. To achieve high-quality results during inference, we find that at least 80 diffusion steps are
492 required for high-level policies and 50 steps for low-level policies.

Table 7: 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

493 G Policy Distillation Experiment

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

496

497 **Two-stage Diff-res** We employ an IK solver to compute the target joints given the fingertip
498 positions predicted by the high-level policy. The low-level policy predicts the residual terms of IK
499 solver instead of the robot actions.

500

501 **Two-stage BeT** We train both high-level and low-level policies with Behavior Transformer
502 [23] instead of DDPM. The hyperparameter of BeT is listed in Table 8.

503

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

506

507 **Multi-task RL** We create a multi-task environment where for each episode a random song is
508 sampled from the dataset. Consequently, we use Soft-Actor-Critic (SAC) [33] to train a single agent
509 within the environment. Both the actor and critic networks are MLPs, each with 3 hidden layers,
510 and each hidden layer contains 256 neurons. The reward function is the same as that in [11].

511

512 **BC-MSE** We train a feedforward network to predict the robot action of next timestep con-
513 ditioned 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 8: 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

514

515 H F1 Score of All Trained Song-Specific Policies

516 Figure 8 shows the F1 score of all song-specific policies we trained.

517 I Detailed Results on Test Dataset

518 **Note:** We observed that the F1 scores reported for all methods in the main part of the paper were
519 higher than their actual values (Precision and Recall are correct). This error is due to an issue with
520 scikit⁴. Here, we present the corrected F1 scores of our two methods. We will update the F1 scores

⁴<https://github.com/scikit-learn/scikit-learn/pull/27577>

521 of other baselines during the rebuttal phase. We apologize for the inconvenience. In Table 9 and
 522 Table 10, we show the Precision, Recall and F1 score of each song in our collected test dataset and
 523 the Etude-12 dataset from [11], achieved by Two-stage Diff and Two-stage Diff-res, respectively.
 524 We observe an obvious performance degradation when testing on Etude-12 dataset. We suspect that
 525 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 9: Quantitative results of each song in the our collected test dataset

Song Name	Two-stage Diff			Two-stage Diff-res		
	Precision	Recall	F1	Precision	Recall	F1
Forester	0.81	0.70	0.68	0.79	0.71	0.67
Wednesday	0.66	0.57	0.58	0.67	0.54	0.55
Alone	0.80	0.62	0.66	0.83	0.65	0.67
Somewhere Only We Know	0.63	0.53	0.58	0.67	0.57	0.59
Eyes Closed	0.60	0.52	0.53	0.61	0.45	0.50
Pedro	0.70	0.58	0.60	0.67	0.56	0.47
Ohne Dich	0.73	0.55	0.58	0.75	0.56	0.62
Paradise	0.66	0.42	0.43	0.68	0.45	0.47
Hope	0.74	0.55	0.57	0.76	0.58	0.62
No Time To Die	0.77	0.53	0.55	0.79	0.57	0.60
The Spectre	0.64	0.52	0.54	0.67	0.50	0.52
Numb	0.55	0.44	0.45	0.57	0.47	0.48
Mean	0.69	0.54	0.56	0.71	0.55	0.57

526

Table 10: Quantitative results of each song in the Etude-12 dataset

Song Name	Two-stage Diff			Two-stage Diff-res		
	Precision	Recall	F1	Precision	Recall	F1
FrenchSuiteNo1Allemande	0.45	0.31	0.34	0.39	0.27	0.30
FrenchSuiteNo5Sarabande	0.29	0.23	0.24	0.24	0.18	0.19
PianoSonataD8451StMov	0.58	0.52	0.52	0.60	0.50	0.51
PartitaNo26	0.35	0.22	0.24	0.40	0.24	0.26
WaltzOp64No1	0.44	0.31	0.33	0.43	0.28	0.31
BagatelleOp3No4	0.45	0.30	0.33	0.45	0.28	0.32
KreislerianaOp16No8	0.43	0.34	0.36	0.49	0.34	0.36
FrenchSuiteNo5Gavotte	0.34	0.29	0.33	0.41	0.31	0.33
PianoSonataNo232NdMov	0.35	0.24	0.25	0.29	0.19	0.21
GolliwoggsCakewalk	0.60	0.43	0.45	0.57	0.40	0.42
PianoSonataNo21StMov	0.32	0.22	0.25	0.36	0.23	0.25
PianoSonataK279InCMajor1StMov	0.43	0.35	0.35	0.53	0.38	0.39
Mean	0.42	0.31	0.33	0.43	0.30	0.32

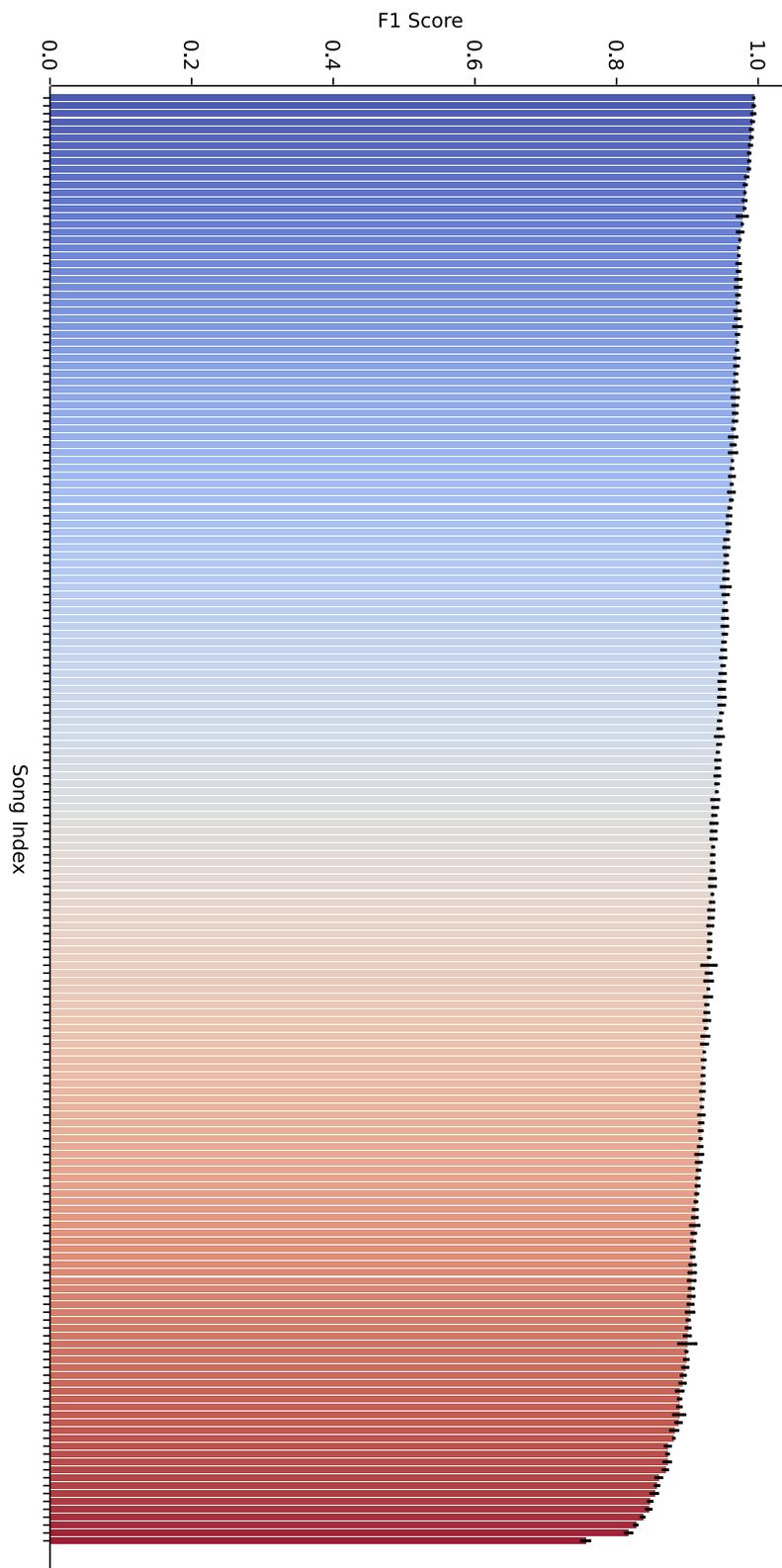


Figure 8: F1 score of all 184 trained song-specific policies (descending order)