Promptable Closed-loop Traffic Simulation Supplementary Material

Anonymous Author(s)
Affiliation
Address
email

A Demo Video

In our submitted supplementary video, we include the rollout videos of all the rollout outputs of `ProSim` in the main paper. We highly recommend interested readers to go through the video demos to observe the promptable and closed-loop properties of `ProSim`.

B `ProSim`

B.1 `Encoder`: Position-aware Attention Details

To model relationships between scene tokens and aggregate features, we pass F^0_{ma} through multiple Transformer layers. In most existing methods, each layer follows $F^l_{ma} = \texttt{MHSA}(F^{l-1}_{ma})$, where `MHSA` denotes multi-head self-attention and l is the layer index. However, since each token in F^0_{ma} is normalized to its local coordinate system, basic `MHSA` cannot infer the relative positional relationship between tokens. Instead, we explicitly model the relative positions between tokens with a position-aware attention mechanism. For scene token i, we compute its relative positional relationship with scene token j with:

$$p_{ma[i,j]} = \text{Rot}(p_{ma[j]} - p_{ma[i]}, -h_{ma[i]}), \quad h_{ma[i,j]} = h_{ma[j]} - h_{ma[i]}, \tag{1}$$

where $p_{ma[i,j]} \in \mathbb{R}^2$ and $h_{ma[i,j]} \in \mathbb{R}$ are the relative position and heading of token j in token i's coordinate system, $\text{Rot}(\cdot)$ is the vector rotation function. We denote this paired relative position as $r_{ma[i,j]} = [p_{ma[i,j]}, h_{ma[i,j]}]$. Then, we perform position-aware attention for token i with:

$$\begin{aligned} f^l_{ma[i]} = \texttt{MHSA}(\text{Q} : &[f^{l-1}_{ma[i]}, \text{PE}(r_{ma[i,i]})], \\ \text{K} : &\{[f^{l-1}_{ma[j]}, \text{PE}(r_{ma[i,j]})]\}_{j\in\Omega(i)}, \\ \text{V} : &\{[f^{l-1}_{ma[j]} + \text{PE}(r_{ma[i,j]})]\}_{j\in\Omega(i)}), \end{aligned} \tag{2}$$

where PE denotes positional encoding and $\Omega(i)$ is the scene token index of the neighboring tokens of i. In our experiments, we set $\Omega(i)$ to contain the nearest 32 tokens of i according to their positions. Note that the above result remains the same regardless of which global coordinate system we use for the scene input $\sigma = (M, A)$. With this formulation, we model the relative position relationship between different scene tokens symmetrically. At each layer, we apply Equation 2 to all scene tokens in parallel. We denote this position-aware attention module as:

$$F^l_{ma} = \texttt{MHSA}'(F^{l-1}_{ma}, P_{ma}, H_{ma}) \tag{3}$$

Note that this position-aware modification can be similarly applied to multi-headed cross-attention `MHCA'`, which we will use later in the `Generator` and `Policy` modules. Finally, we obtain the last-layer token features as scene tokens $F = [F_m, F_a]$.

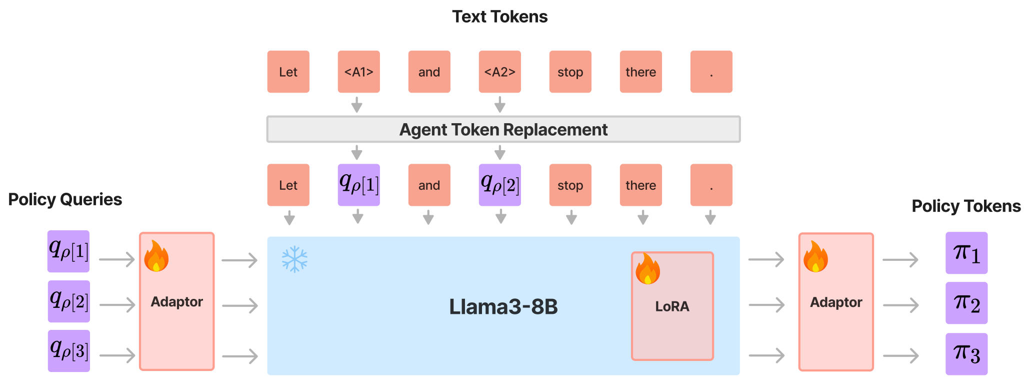


Figure A1: Language condition encoder for `Generator`.

B.2 `Generator`: Language Prompting Details

For each scene, we have an optional user-input text prompt L that contains multiple sentences that could describe agent behaviors, interactions and scenario properties. To make it easy to refer to different agents in the scene, we ask the user to use a specific format "<A[i]>" when mentioning the i-th agent. For example, to instruct agent a_1 to stop, the user would say *"let <A1> stop"*.

To process the scene-level text prompt L and condition all the agents, we use an LLM to comprehend the natural language prompt and policy features, and generate language-conditioned policy features for all agents. To do this, we use a LLaMA3-8B model finetuned with LoRA as backbone, as well as two adaptors to bridge the latent spaces of LLM and policy tokens. We show an overview of our model in Figure A1.

Specifically, we use a two-layer MLP adaptor to convert all the policy features to in the LLM's feature space $\{t_{a[1]}, ..., t_{a[N]}\} = \text{MLP}(\{q_{\rho[1]}, ..., q_{\rho[N]}\})$. This operation enables the LLM to take and comprehend agent policy features in the text space. Next, we use LLM's text tokenizer and input embedding to obtain text tokens from the natural language prompt L $\{t_{L[1]}, ..., t_{L[O]}\}$, where O is the number of text tokens in L.

Note that some tokens of L specifically mentions indexed agents (e.g., "<A1>"). To help LLM understanding the correspondence between agent reference and their policy tokens, we directly replace these reference text tokens with agent policy tokens. Specifically, for an agent-reference text token $t_{L[j]}$, we replace it with the corresponding agent policy token $t_{L[j]} \leftarrow t_{a[i]}$ given $t_{L[j]}$ corresponds to "<A[i]>" To make sure each "<A[i]>" is tokenized to a single token, we add them as new tokens to the text tokenizer. We call this step "Agent Token Replacement" in Figure A1.

After getting all the text and agent token features, we concatenate them to create the complete input sequence $\{t_{L[1]}, ..., t_{L[O]}, t_{a[1]}, ..., t_{a[N]}\}$ and feed it to the LLM. We then extract the hidden features for the last N tokens from the last LLM layer $\{t'_{a[1]}, ..., t'_{a[N]}\}$, which contains text-contextualized policy features for each agent. Next, we use an MLP adaptor to convert these features back to get the text-conditional policy features $\{q_{L[1]}, ..., q_{L[N]}\} = \text{MLP}(\{t'_{a[1]}, ..., t'_{a[N]}\})$.

Finally, for each agent, we obtain its policy token π_i by adding the prompt and text conditional policy tokens together, $\pi_i = q_{\rho[i]} + q_{L[i]}$.

B.3 Training

LLM Pretraining. As described above, we train the LLM in `Generator` with LoRA to comprehend and generate policy token features. However, we found that directly training the LLM with the closed-loop imitation loss leads to inferior text-prompting performance. We conjecture this is because at the early training stage the LLM is not prepared to interact with the policy tokens. Meanwhile, the rollout loss can be decently optimized without using the LLM's output, giving little signal for LLM to learn.

To deal with this issue, we propose to pretrain the LLM to have the capacity to interact with policy token features. To this end, we first train a `ProSim` model without using text prompts and the LLM with the rollout loss $\mathcal{L}$. Next, we add a simple MLP layer after the `Generator` to predict the goal point of each agent given its policy token π. We supervise this task with an MSE loss $\mathcal{L}_{\text{goal}}$ using GT goal points. This task is much simpler than the full task while enforces the LLM to interact with policy tokens. To pretrain the LLM, we fix all other modules and only train the LLM and this new MLP with $\mathcal{L}_{\text{goal}}$. Here we only use text as the prompt in inputs. After pretraining, the LLM learns to predict the goal intention of each agent from text prompt and add that information to π, making it already useful for the `Policy`. Finally, we discard the goal-prediction MLP and train the full `ProSim` module with all types of prompts and the complete loss $\mathcal{L}$. In our experiments, we pretrain the LLM on `ProSim-Instruct-520k` for 5 epoches before using it in the full `Generator` module.

Collision loss. For collision loss $\mathcal{L}_{\text{coll}}$, we aim to compute the overlapping area between each pair of agent using their bounding boxes through the full rollout trajectory. To this end, at each timestep, for each agent we compute their occupation polygon using their size, position and heading at this timestep. Then, we measure the overlapping area of each agent polygon pair by computing the signed distance between these polygons, where positive distance indicate no collision while negative distance collision indicate a collision. Specifically, we compute the signed distance between polygons A and B with the distance between the origin point and the Minkowski sum A + (-B). For all the signed distance, we compute the loss value by first setting the positive distance to 0 (no collision), and then taking the negative of the rest of signed distances to penalize collisions. We compute the average over all agents through all timesteps to obtain the final $\mathcal{L}_{\text{coll}}$. We implement this loss function by referring to the Waymo Open Sim Agent Challenge collision metric.

Offroad loss. For offroad loss $\mathcal{L}_{\text{off}}$, we aim to compute the overlapping area of each agent and the offroad areas through the full rollout trajectory. To this end, we use the similar strategy as in the collision loss. Specifically, at each timestep we obtain all the agent polygons in the same way as in the collision loss. Then, we compute the signed distance between the four corners of each polygones to a densely sampled set of road edges. Similarly, positive distance indicate no offroad while negative distance indicate being offroad. Finally, We average over all negative distances for all agents through all timesteps to obtain the final offroad loss. We implement this loss function by referring to the Waymo Open Sim Agent Challenge offroad metric.

C `ProSim-Instruct-520k`

C.1 Route Sketch Labeling Details

Compared with the real trajectory, points in the route sketch are sparse, noisy, and incomplete. We simulate these effects with the following steps. First, we extract each agent's complete trajectory from τ. Then, we process this point set by 1) uniformly subsampling the points for sparsity; 2) adding random noise to each point; 3) randomly sampling a consecutive subset. For each agent, its route sketch is a set of ordered 2D points representing sketch points on the map. The number of points in route sketch could be different across agents. In our experiment, we use a uniform subsample rate of 5. We than add random noise with standard variation of 0.1 meter. Finally, we ensure the randomly sampled consecutive subset contains at least 5 points.

C.2 Action Tag Labeling Details

For each action type, we carefully design a heuristic function that takes an agent's trajectory $\mathbf{s}_{t_1:t_2}$ from step t_1 to t_2, and outputs a binary label whether $\mathbf{s}_{t_1:t_2}$ satisfies the condition of this action. Then, we can run this function across the full rollout with sliding window and temporal aggregation to obtain the full duration $[t_s, t_e]$ that this motion tag is valid. For each agent, we run all the motion tags with this method we obtain a set of motion tags. Finally, we post-process the labels to remove conflicting motion tags and temporal noises. Please refer to our codebase for the heuristic function implementation details upon release.

C.3 Natural Language Labeling Details

For each scenario, we provide the LLM model with agent properties (name and type) as well as all of their agent tags (action type and duration). We then prompt LLM to output 20 different sentences, each describing the agent behavior or scenario properties in natural language. To obtain interesting and diverse language description of the scenario, in the system prompt we instruct the LLM to 1) describe temporal transition of agent behavior (e.g., "Let <A1> change to the left lane and then make a left turn."); 2) describe scenario properties (e.g., "This is a busy scene with most agents accelerating"); 3) describe relationships of different agents (e.g., "Let <A1>, <A2>, <A3> keep their own lanes simultaneously"). We concatenate these sentences together to form the prompt L for each scenario.

Here we show full prompt we used for the LLM labeling:

Prompt 1: Full prompt for LLM labeling

```
Example input:
  Vehicle Agents:
      ['<ego>', '<71f1c>', '<df6a1>', '<dad99>']
  Pedestrian Agents:
      ['<a261a>', '<191e8>']
  Motorcycle Agents:
      ['<d3ddc>', '<8cc93>', '<73c13>', '<d6a9e>']

  Agent to Agent:
    ParallelDriving - Agent (Left):<d6a9e>, Agent (Right):<dad99>, Start:50, End:80
    ByPassingRight - Agent (Right, Faster, Overtaking):<ego>, Agent (Left, Slower, Overtaken
     ):<dad99>, Start:30, End:65

  Agent Behavior:
    Decelerate - Agent:<d3ddc>, Start:0, End:5
    Decelerate - Agent:<ego>, Start:30, End:55
    Decelerate - Agent:<dad99>, Start:35, End:45
    Decelerate - Agent:<d6a9e>, Start:45, End:80
    KeepLane - Agent:<d3ddc>, Start:0, End:60
    KeepLane - Agent:<dad99>, Start:30, End:80
    KeepLane - Agent:<d6a9e>, Start:40, End:80
    KeepSpeed - Agent:<191e8>, Start:0, End:20
    KeepSpeed - Agent:<ego>, Start:0, End:30
    KeepSpeed - Agent:<d6a9e>, Start:40, End:45
    Parked - Agent:<a261a>, Start:0, End:25
    Parked - Agent:<df6a1>, Start:0, End:80
    RightLaneChange - Agent:<ego>, Start:0, End:15
    Straight - Agent:<191e8>, Start:0, End:20
    Straight - Agent:<d3ddc>, Start:0, End:70
    Straight - Agent:<ego>, Start:0, End:80
    Straight - Agent:<dad99>, Start:30, End:80
    Straight - Agent:<d6a9e>, Start:40, End:80
    Stopping - Agent:<d3ddc>, Start:5, End:70
    Stopping - Agent:<dad99>, Start:45, End:80
    LeftLaneChange - Agent:<ego>, Start:15, End:65

Example output:

Here are the 20 commands for the simulation:
  "<ego> bypasses <dad99> from the right lane side when <dad99> is driving slower and
     finally stopping."
  "Do right lane change <ego> at the start of the simulation."
  "Motorcycle <d3ddc> decelerates early on."
  "After finishing the initial maneuver, redirect <ego> to occupy the left lane."
  "Make <dad99> and <d6a9e> to drive parallel, with <d6a9e> on the left."
  "<ego> slows down following the sequence of lane changes.."
  "All vehicles, except for <ego>, <dad99>, and <d6a9e>, remain parked."
  "Let <ego> maintain a steady speed after decelerating."
  "Command <d3ddc> cyclist to come to a complete stop after its initial slowdown."
  "Instruct <dad99> to decelerate and then stop towards the end of the scenario."
  "Keep <d6a9e> in its lane after it finishes driving parallel."
  "Walking person <191e8> should keep a steady pace before stopping."
  "<d3ddc> resumes moving straight after stopping."
  "Keep all parked vehicles stationary to represent a low-activity scene."
  "Direct <dad99> car to travel straight for an extended period after decelerating."
  "Ensure <ego> car moves straight throughout the simulation."
  "After slowing, instruct <d6a9e> to continue on a direct trajectory."
  "Let the pedestrian <a261a> standstill in the scene."
  "Emphasize the limited activity within the scene, highlighting agents either stopping or
    staying within their lanes."
  "Pedestrian <191e8> walks straight."
```

We also show two LLM output examples for two scenarios:

Prompt 2: Text prompt labeling example 1

```
Here are the 20 commands for the simulation:

1. "Have <43> accelerate and make a left turn throughout the entire simulation."
2. "Initially, <58> accelerates, but then slows down and comes to a stop."
3. "Keep <10> moving straight while it decelerates early on."
4. "Make <48> decelerate and then turn left towards the middle of the simulation."
5. "Command <39> to accelerate and drive straight after 40 seconds."
6. "Make <7> accelerate rapidly towards the end of the simulation."
7. "<ego> remains parked and stationary throughout the entire scenario."
8. "Direct <11> to stay parked for the entire duration of the simulation."
9. "Most vehicles, except for a few, are parked and stationary at the start of the
     simulation."
10. "After 40 seconds, <14> starts moving after being parked initially."
11. "Make <4> change direction with a right turn after being parked for a while."
12. "Let <48> decelerate and come to a stop before making a left turn."
13. "Have <58> accelerate initially, but then slow down and stop."
14. "Command <45> and <56> to stop after 40 seconds."
15. "Instruct <32> to stop after 15 seconds and then stay stationary."
16. "Make <15> stop and stay stationary throughout the entire simulation."
17. "Have <50>, <52>, and <58> accelerate and drive straight throughout the simulation."
18. "Keep <24> moving straight after 65 seconds."
19. "Make <39> accelerate and drive straight after being parked initially."
20. "Emphasize the dynamic nature of the scene, with agents accelerating, decelerating, and
     changing directions."
```

Prompt 3: Text prompt labeling example 2

```
Here are the 20 commands for the simulation:

1. "<653> accelerates while making a left turn throughout the entire simulation."
2. "Have <654> maintain a steady acceleration from start to finish."
3. "<661> gradually slows down and drives straight throughout the scenario."
4. "<666> decelerates and moves straight without any turns or stops."
5. "Keep <1465> cycling in a straight line for the entire 8 seconds."
6. "Make <651> drive straight without any changes in speed or direction."
7. "Ensure <659>, <660>, <662>, <664>, and <ego> remain parked and stationary throughout the
      simulation."
8. "After accelerating, have <653> continue moving in a straight line."
9. "Instruct <661> to decelerate and then maintain a steady speed."
10. "Make <654> overtake <661> from the left lane."
11. "<663> follows <661> at a steady pace, maintaining a safe distance."
12. "Command <653> to merge into the lane where <661> is driving."
13. "Direct <1465> to pass <651> on the right side."
14. "Have <661> change lanes to the left and then continue driving straight."
15. "Make <654> drive parallel to <653> on the right side."
16. "<ego> remains stationary, observing the surrounding traffic."
```

```
17. "After accelerating, have <653> change lanes to the right."
18. "Ensure <660> and <664> remain parked, blocking the left and right lanes respectively."
19. "Instruct <663> to decelerate and then stop behind <661>."
20. "The bicycle <1465> cycles past the parked vehicles, maintaining a steady pace."
```

C.4 Metric Formulation

In our paper, we provide two metrics of promptable closed-loop traffic simulation to measure *realism* and *controllability*. Here we link these metrics to the problem formulation in Section 3. Recall that we formulate promptable closed-loop traffic simulation as

$$p(\mathbf{s}_{1:T}|\sigma,\rho)=\prod_{t=1}^{T}\prod_{i=1}^{N}p(s_t^i|\mathbf{s}_{1:t-1},\sigma,\rho). \tag{4}$$

Given GT data (τ,σ,ρ), *realism* measures the probability of the real rollout under the model distribution $p(\tau|\sigma,\rho)$. In our main paper, we implement this metric with **ADE**$(\hat{\tau},\tau)$.

On the other hand, *controllability* measures how well the model follows the prompt ρ. We quantify this by comparing the model's realism gain against the unconditional model rollout $p(\tau|\sigma,\rho)-p(\tau|\sigma)$. In our main paper, we implement this metric with relative improvement (**% Gain**) in realism of the model's output with and without prompts. We compute % Gain by comparing rollout ADE with and without prompt conditioning. : % Gain $=\frac{\text{ADE}(\overline{\tau},\tau)-\text{ADE}(\hat{\tau},\tau)}{\text{ADE}(\overline{\tau},\tau)}\times 100\%$.

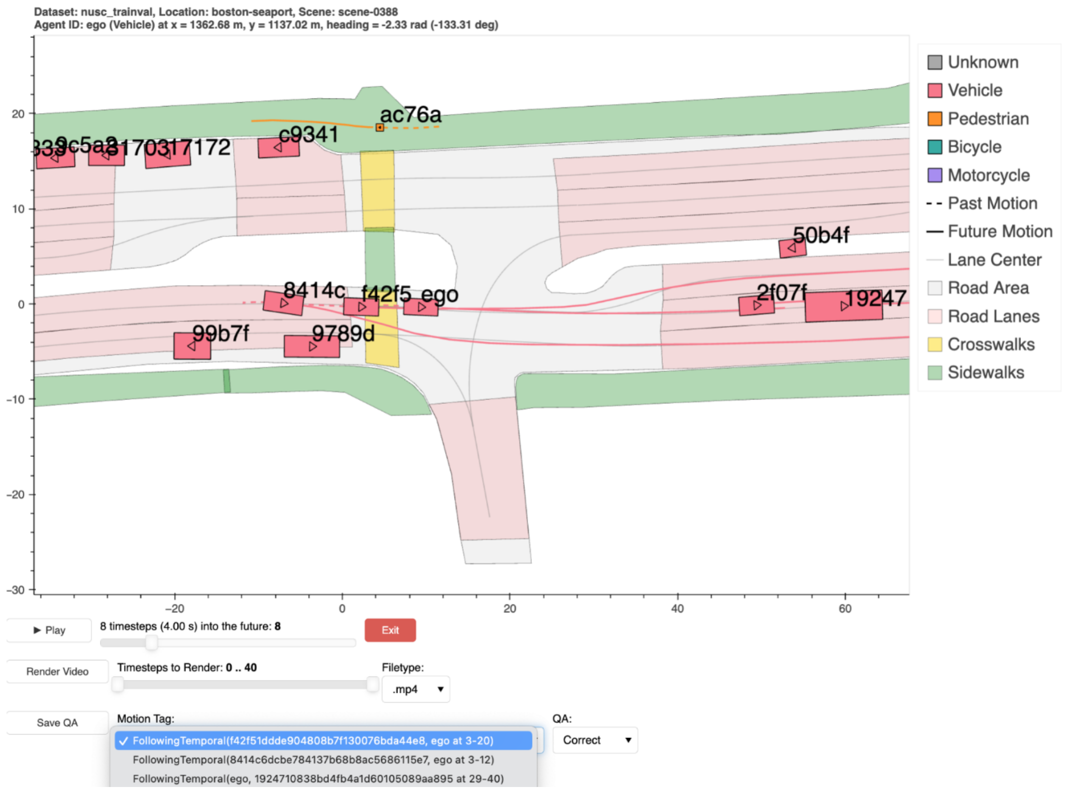
C.5 Quality Assurance

To ensure the prompts we generate faithfully reflect the agent behaviors in the scenario, we conduct a careful quality assurance process with human effort. As Goal Point and Route Sketch are directly modified from real trajectories, there is no need to check their accuracy. On the other hand, the accuracy of Text is largely dependent on the accuracy of the Action Tag as it stems from Action Tags of all agents in a scene. Therefore, we focus on checking the quality of Action Tag. For each action types, we ask human labelers to manually check whether the labeled action tag is accurate both semantically and temporally by viewing the rollout videos. At each round, we ask human labelers to check 100 motion tag examples for each action type. If the qualification rate of a certain action type is below 85%, we rewrite the heuristic function of this action type according to the human feedback, relabel the motion tags of this type and ask for another round of human checking. We continue this process until all the action types pass the quality threshold.

We show the interface we developed for human labelers in Figure A2. This interface allow human labelers to go through and rewind the scenario easily with the interactive progress bar. For each scenario with multiple action tags, the interface let the labeler to go through all the action tags all together. This allows the human labeler to QA multiple motion tags of the same scenario very efficiently. In average, we found human labelers take around 10 seconds to give the QA output for each motion tag. Additionally, we ask human labelers to give a QA output for each tag, choosen from Correct, Wrong Action, Wrong Time, Wrong Agent, and Need Attention (unsure or other types). These different types of QA error tags provide us useful feedback to improve our heuristic functions.

D Additional experiment results

In our paper, we show benchmark results of 5 different prompt combinations. Aside from these combinations shown in the paper, we also show the % Gain results of other prompt combinations in Table A1. We can see from the Table A1 that ProSim achieves consistent gain with different kinds of prompt modality combinations. These results show that ProSim allows users to freely combine different prompt modalities with high controllability.



Dataset: nusc_trainval, Location: boston-seaport, Scene: scene-0388
Agent ID: ego (Vehicle) at x = 1362.68 m, y = 1137.02 m, heading = -2.33 rad (-133.31 deg)

8 timesteps (4.00 s) into the future: **8**

Timesteps to Render: **0 .. 40**

Filetype: .mp4

Motion Tag:
✓ FollowingTemporal(f42f51ddde904808b7f130076bda44e8, ego at 3-20)
FollowingTemporal(8414c6dcbe784137b68b8ac5686115e7, ego at 3-12)
FollowingTemporal(ego, 1924710838bd4fb4a1d60105089aa895 at 29-40)

QA: Correct

Figure A2: Interface used by human labeler for Quality Assurance.

Metric	ADE ↓	Gain ↑	ADE ↓	Gain ↑	ADE ↓	Gain ↑	ADE ↓	Gain ↑
Prompt	Goal + Sketch		Goal + Text		Sketch + Action		Sketch + Text	
`ProSim`	0.4845	48.98%	0.5983	37.00%	0.5588	41.16%	0.5698	40.00%
Prompt	Goal + Action + Sketch		Goal + Action + Text		Text + Sketch + Action		All Types	
`ProSim`	0.3635	61.72%	0.5663	40.37%	0.5311	44.08%	0.2877	69.71%

Table A1: Controllability evaluation of `ProSim`