Towards Human-like Virtual Beings: Simulating Human Behavior in 3D Scenes

Supplementary Material

In this document, we provide the following items that shed deeper insight on our contributions:

- §S1: Details about data generation prompts.
- §S2: Details about environment description heuristics.
- §S3: Details about motion trajectory generation and human-scene interaction.
- §S4: Details about MCTS process.
- §S5: More dataset statistics.
- §S6: More qualitative visualization and detailed goal-plan json.
- §S7: Discussion of legal/ethical considerations and limitations.

S1. Prompts for Data Generation

We give full details of the prompts used in generating linguistic goal-plan trees, including in goal-plan trees initialization Table S1, attribution of interchangeable groups in Table S2, and goal-plan tree refinement in Table S3.

S2. More Details of Environment Description Generation

In ACTOR, we employ heuristic functions to parse agent perception. They take scene geometry, which includes object segmentation, as well as the agent's state of position and action type as input (Fig. 2 Right) and gives a linguistic description of the entire scene and the agent's surroundings as output. To provide a concrete example, a linguistic environmental description could be as follows:

ENVIRONMENT: {residential interior}; OBJECTS: {bed, desk, chair, kitchen counter, sink, television, ..., sofa}; SURROUNDINGS: {sink: *empty*, faucet: *turned on*, toilet: *vacant*}

The "Environment" and "Objects" fields offer the agent a comprehensive understanding of the human behaviors that may occur in the current scene. On the other hand, the "Surroundings" field provides the agent with information about interactive objects and their respective states in the surrounding environment.

S3. More Details of Motion Trajectory Generation and Human-Scene Interaction

In the action module of ACTOR, we generate whole-body human actions in 3D scenes using off-the-shelf conditional motion generation models. Here we provide more details on how we achieve motion trajectory generation and human-scene interaction.

For motion trajectory generation, once the linguistic planning step provides us with a parsed <action, object> pair, we categorize the action into two types: still and moving. First, for still actions, such as stand up and knock, no trajectory estimation is necessary as the human remains in a fixed position. Then, as we mentioned in §4.1, for moving actions like walking, trajectory paths are pre-estimated [1]. We adapt the trajectory estimation module from [1]. The end position is sampled based on contact and collision rules, taking into account the scene and targeted object geometry. The goal is to position the human close to the target while avoiding collisions with walls. The start position is determined based on the previous step's end position. Subsequently, this module utilizes an improved A* path search algorithm, considering the start-end position and the entire scene geometry, to generate the final trajectory.

Furthermore, for achieving human-scene interaction, we construct the leaf nodes of these goals as <scene, text, motion> pairs to finetune the conditional motion generation model [2], where a scene-conditioned branch is added and implemented with a pretrained and fixed Point Transformer [3] to achieve human-scene interaction. The conditional motion generation model takes pre-estimated trajectory, text description, and scene geometry as input to generate the whole-body motion. While the grasp estimation model further refine the hand pose. During finetuning, we keep the hyperparameters consistent with the official implementation, except for using a learning rate that is half of the original value. This adjustment already yields moderate adaptation.

S4. More Details of MCTS Process

In Fig. 2, we illustrate a generalized view of the tree structure used in different algorithms such as greedy search, DFS, BFS, A* search, and also MCTS. The value function assigns values to each node, while node expansion determines the probability of transitioning from the node state. In this section, we present a more detailed description of MCTS process based on the proposed value-driven planning approach. Specifically, in MCTS, the root node represents the current state of the system being executed. Each child node corresponds to a potential action or step that can be taken from the current state. These nodes have associated values and states, which include information about the current scene and the state of the human involved. First, the initial value of a node is determined by the value function (*cf*. §4.2). This value is then updated during the backpropagation phase. Second, during the expansion phase, new child nodes are created by sampling from LLM within the Node Expansion process. Third, in the backpropagation phase, the results of a two-step rollout are summarized. This involves considering the values of the two-step children and updating the value of the parent node accordingly. Finally, the MCTS process continues to iterate until a termination condition or goal is reached, signifying that the search is finished.

S5. More Dataset Statistics

We provide lists of most frequently used motion and objects in Fig. S3-S4 and Table S4-S5. Example scene is illustrated in Fig. S1. We next give brief description of the scene dataset we incorporate for data generation: *(i)* ScanNet [4] is a widely known dataset in computer vision and 3D scene understanding. ScanNet is a large-scale RGB-D dataset containing 3D scans of indoor spaces, along with detailed semantic and instance-level annotations. It is commonly used for tasks such as 3D scene understanding, object recognition, and semantic segmentation. Researchers and developers use ScanNet to train and evaluate algorithms for various applications related to understanding the 3D structure of indoor environments. *(ii)* Habitat-Matterport 3D Research Dataset (HM3D) [5] is the largest-ever dataset of 3D indoor spaces. It consists of 1,000 high-resolution 3D scans (or digital twins) of building-scale residential, commercial, and civic spaces generated from real-world environments. Researchers can use it with FAIR's Habitat simulator to train embodied agents, such as home robots and AI assistants, at scale.

S6. More Visualization and goal-plan JSON

More illustrations of qualitative visualization and detailed goal-plan tree JSONs are given in Fig. S2.

S7. Discussion

Asset License and Consent. We build BEHAVIORHUB on top of three human motion datasets (*i.e.*, AMASS [6], BABEL [7], GRAB [8]), and two indoor scene datasets (*i.e.*, ScanNet [4], HM3D [5]), that are all publicly and freely available for academic purposes. We implement our agent with LangChain codebase using GPT-3.5 and GPT-4 models. AMASS (https://amass.is.tue.mpg.de/) is released under this License; BABEL (https://babel.is.tue.mpg.de/) is released under this License; GRAB (https://grab.is.tue.mpg.de/) is released under this License; ScanNet (http://www.scan-net.org/) is released under this License, and the code is released under the MIT license; HM3D (https://aihabitat.org/datasets/hm3d-semantics/) is released under this License; LangChain codebase (https://github.com/langchain-ai/langchain) is released under the MIT license. GPT models from OpenAI are available for academic research under this License.

Crowdsourcing Data Collection. BEHAVIORHUB is primarily collected through an automated data collection pipeline, with minimal human intervention required for verification. In addition, we conduct user studies to evaluate the quality of the human-subjective generation. All human experts involved in the annotation and evaluation process are well-informed that their contributions will be utilized for academic research, and their consent is obtained through signed agreements. To ensure privacy and equality, the annotation process strictly adheres to guidelines that prevent the disclosure of personal information about the experts and minimize data bias.

Limitation Analysis. One limitation of this work is that although the generated human motions are scene-aware, the interaction with objects is currently assumed to be static. In our future work, we aim to enhance the capabilities of BEHAVIORHUB and the ACTOR agent by incorporating interactions with interactive objects. To achieve this goal, we have developed our environment using the Habitat-Sim simulator, which offers the necessary flexibility to realistically simulate these interactions in future developments. Furthermore, we are committed to designing a more realistic benchmark and algorithm for simulating interactions, ensuring that our work aligns with future advancements in this area. To encourage broader exploration and engagement from the research community, we will also release our complete code implementation, comprising the environment simulator, dataset construction, and agent implementation.

Broader Impact. This study focuses on simulating high-level, long-horizon, abstract goal-driven human behaviors in 3D scenes. The approach has several positive implications, including advancements in Embodied AI, potential to populate virtual reality communities, and enhancement of non-player game character development. However, there are potential negative consequences to consider. The generated results could be exploited for malicious purposes, such as the creation of highly realistic and deceptive virtual characters for social engineering or online scams. While this issue falls outside the scope of this paper, we intend to release our models in a gated manner to ensure that they are solely used for academic research purposes and prevent any misuse.

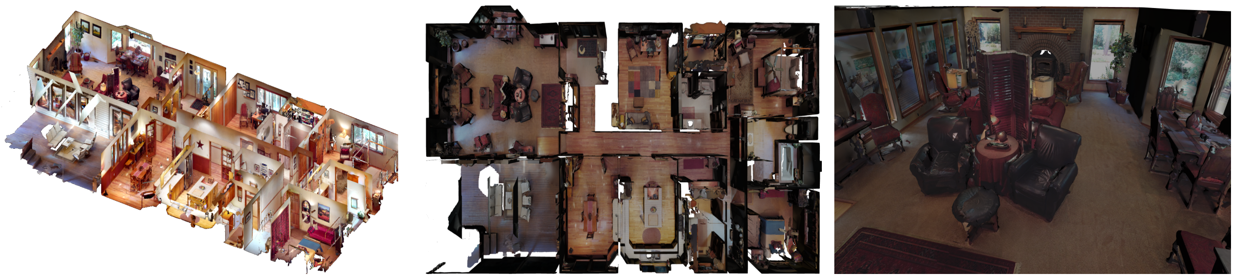
Figure S1. Example Scene. (a) Global view from a slanted perspective; (b) Global top-down view; (c) Local view of a living room.

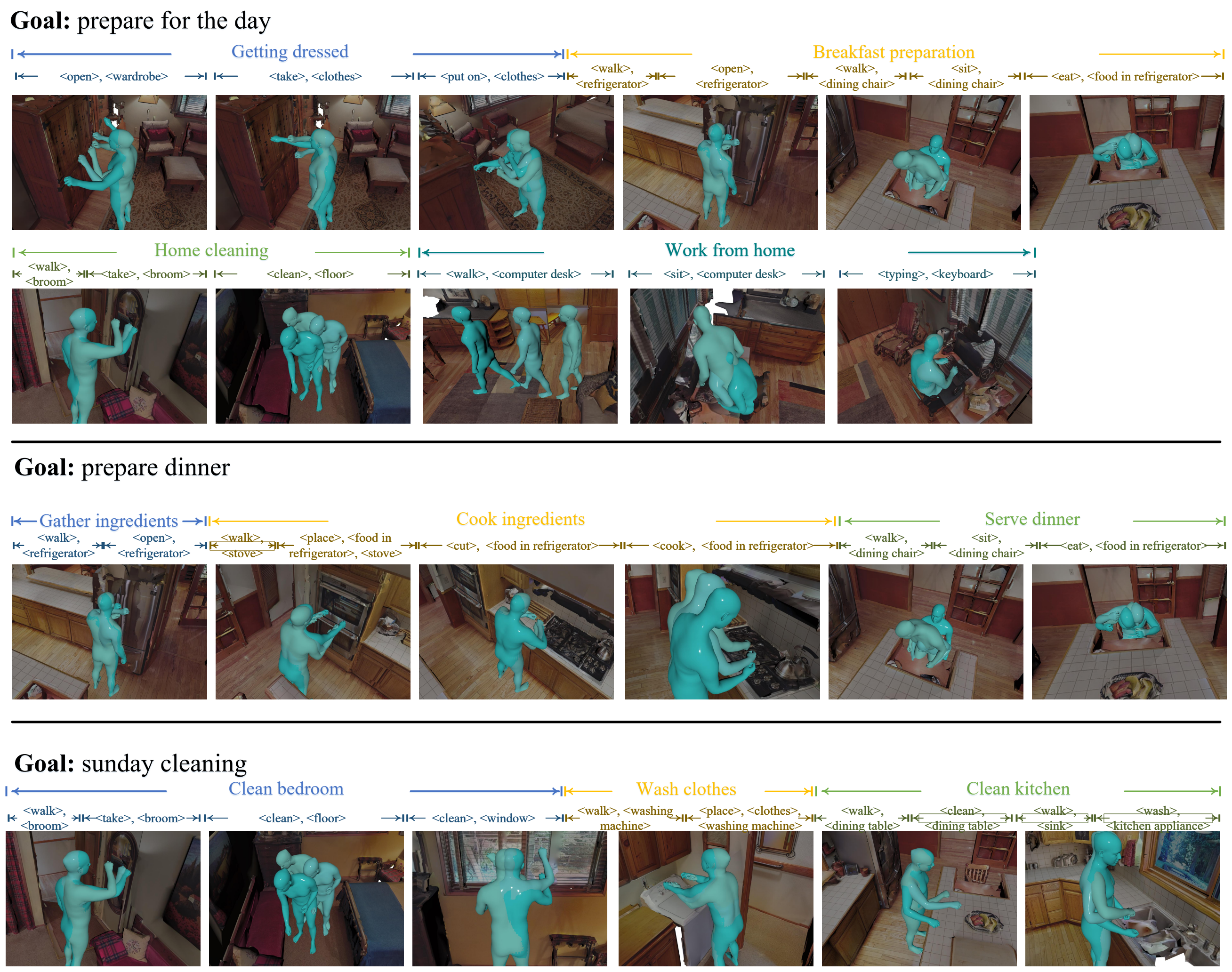

Figure S2. More qualitative visualization on our BEHAVIORHUB dataset.

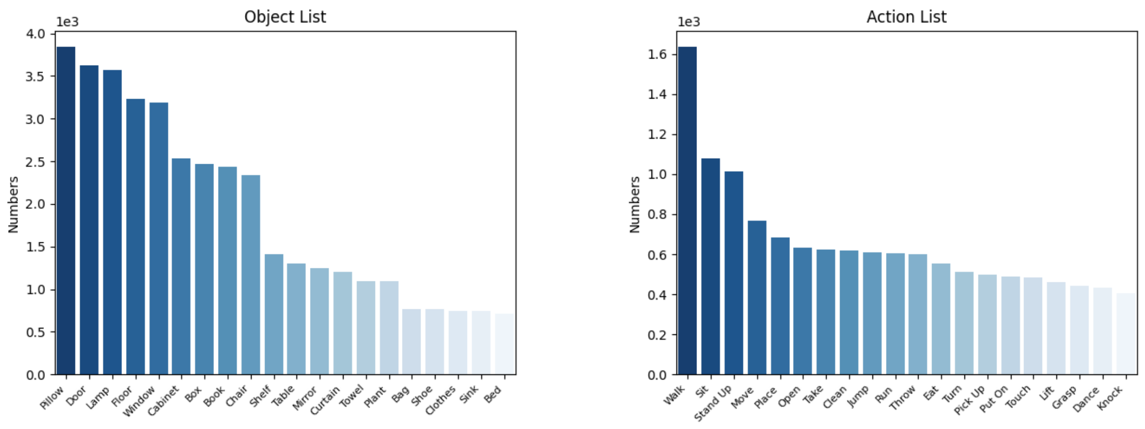

Figure S3. (a) Counts of actions in our BEHAVIORHUB dataset; (b) Object counts.

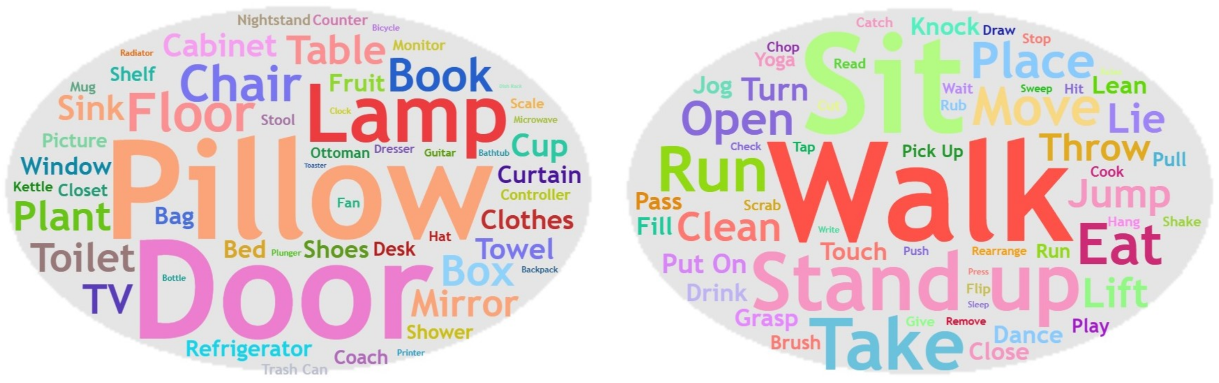

Figure S4. (a) Most frequently used objects; and (b) Most frequently used actions.

Table S1. Detailed prompt design for goal-plan trees initialization.

	Prompt
Goal-Plan Trees Initialization	#1 goal-plan trees Initialization Stage - The following is a friendly conversation between a human and an AI. The AI is professional and can generate multiple goal-plan trees with lots of specific details from its context. The AI assistant is required to using the provided “object list” and "action list" to come up with several tree-structure tasks with the following format: [{“Root”: task, “children”: [{ “node1”: subtask, “children”: [{“node1-1”: ACTION, “children”: []}, {“node1-2”: ACTION, “children”: []}]}, {“node2”: subtask, “children”: [{“node2-1”: ACTION, “children”: []}]}]}]. Note: “ACTION” must be “<action>”, “<action>, <object>” or “<action>, <object>, <object>”, non-leaf nodes must be “task” or “subtask”. Intermediate nodes must be grouped, and the order of nodes in the same group is interchangeable. The AI assistant must reply in JSON format. The “task” or “subtask” field represents high-level task such as “Read a book”, “Take a shower” or “Watch TV”. The “task” or “subtask” must be complex activities or objectives in household. The “<action >” must be selected from the “action list”, “<object>” must be selected from the “object list”, and together they achieve the corresponding “task”. Here are the “object list” and “action list” provided: {{Object List}}, {{Action List}}. To assist with goal-plan tree generation, here are several cases for your reference: {{Demonstrations}}.
	Demonstrations
	Now, please generate a tree-structure tasks:
	{“Root”: “play the toy”, “children”: [“node1”, “node2”], “interchangeable groups”: [“node”]}
	{“node1”: “walk toy”, “children”: [] }
	{“node2”: “play toy”, “children”: [] }
	Now, please generate a tree-structure tasks with more branches and more depths. Remember you should reply in JSON format and the “<action>” must be selected from the “action list”, “<object>” must be selected from the “object list”:
	{“Root”: “morning routine”, “children”: [“node1”, “node2”, “node3”]}
	{“node1”: “have breakfast”, “children”: [“node1-1”, “node1-2”, “node1-3”] }
	{“node1-1”: “<walk>”, <refrigerator>”, “children”: []}
	{“node1-2: “<open>, <refrigerator>”, “children”: []}
	{“node1-3”: “<garb>”, <food in refrigerator>”, “children”: []}
	{“node2”: “eating”, “children”: [“node2-1”, “node2-2”, “node2-3”] }
	{“node2-1”: “<walk>, <dining table>”, “children”: [] }
	{“node2-2”: “<sit>, <dining table>”, “children”: [] }
	{“node2-3”: “<eat>“, “children“: [] }
	{“node3”: “work”, “children”: [“node3-1”, “node3-2”, “node3-3”] }
	{“node3-1“: “<walk>, <computer chair>“, “children“: [] }
	{“node3-2“: “<sit>, <computer chair>“, “children“: [] }
	{“node3-3“: “<typing>, <keyboard>“, “children“: [] }

Table S2. Detailed prompt design for intermediate nodes labeling.

	Prompt
Intermediate Nodes Labeling	#2 Intermediate Nodes Labeling Stage – With the input goal-plan tree in JSON, the AI should assist in labeling the intermediate nodes in the trees using the attribute "interchangeable groups". Note: Intermediate nodes must be grouped, and the order of nodes in the same group is interchangeable. For a more comprehensive understanding of this procedural step, please refer to the corresponding demonstration {{*Demonstrations*}}. Remember you should reply in JSON format.
	Demonstrations
	Please label the intermediate nodes in the following goal-plan tree:
	Query:
	{"Root": "evening routine", "children": ["node1", "node2", "node3"]}
	{"node1": "watch TV", "children": ["node1-1", "node1-2", "node1-3"] }
	{"node1-1": "<walk>", <couch>", "children": []}
	{"node1-2: "<sit>, <couch>", "children": []}
	{"node1-3": "<press>", <remote>", "children": []}
	{"node2": "have dinner", "children": ["node2-1", "node2-2", "node2-3", "node2-4", "node2-5", "node2-6"] }
	{"node2-1": "<walk>, <refrigerator>", "children": [] }
	{"node2-2": "<open>, <refrigerator>", "children": [] }
	{"node2-3": "<take>, <food in refrigerator>", "children": [] }
	{"node2-4": "<walk>, <dining chair>", "children": [] }
	{"node2-5": "<sit>, <dining chair>", "children": [] }
	{"node2-6": "<eat>, <food in refrigerator>", "children": [] }
	{"node3": "edtime routine", "children": ["node3-1", "node3-2"] }
	{"node3-1": "<walk>, <bed>", "children": [] }
	{"node3-2": "<lie>, <bed>", "children": [] }
	Response:
	{"Root": "evening routine", "children": ["node1", "node2", "node3"], "interchangeable groups": ["group1", "group2"]}
	{"node1": "watch TV", "children": ["node1-1", "node1-2", "node1-3"] }
	{"node1-1": "<walk>", <couch>", "children": []}
	{"node1-2: "<sit>, <couch>", "children": []}
	{"node1-3": "<press>", <remote>", "children": []}
	{"node2": "have dinner", "children": ["node2-1", "node2-2", "node2-3", "node2-4", "node2-5", "node2-6"] }
	{"node2-1": "<walk>, <refrigerator>", "children": [] }
	{"node2-2": "<open>, <refrigerator>", "children": [] }
	{"node2-3": "<take>, <food in refrigerator>", "children": [] }
	{"node2-4": "<walk>, <dining chair>", "children": [] }
	{"node2-5": "<sit>, <dining chair>", "children": [] }
	{"node2-6": "<eat>, <food in refrigerator>", "children": [] }
	{"node3": "watch TV", "children": ["node3-1", "node3-2"] }
	{"node3-1": "<walk>, <bed>", "children": [] }
	{"node3-2": "<lie>, <bed>", "children": [] }
	{"group1": [{"node1"}, {"node2"}]}
	{"group2":[{"node3"}]}

Table S3. Detailed prompt design for goal-plan trees refinement.

	Prompt
goal-plan Tress Refinement	#3 goal-plan Tress Refinement Stage – Given the goal-plan tree in JSON format, the AI assistant helps improve its rationality from two aspects: 1. Completing the missing internal steps, which can often be revised on commonsense (e.g., opening the refrigerator without closing it). 2. Enhancing the non-leaf node descriptions to be more abstract (e.g., from ‘use toilet’ to ‘feel the call of nature’). Note that: You should only output the revised goal-plan tree in JSON. To facilitate goal-plan tree refinement, a set of illustrative cases is provided for reference: {{Demonstrations}}.
	Demonstrations

	Please refine the goal-plan tree:
	Query:
	{“Root”: “use toilet”, “children”: [“node1”, “node2”], “interchangeable groups”: []}
	  {“node1”: “<walk>, <toilet>”, “children”: [] }
	  {“node2”: “<sit>, <toilet>”, “children”: []}
	Response:
	{“Root”: “feel the call of nature”, “children”: [“node1”, “node2”], “interchangeable groups”: []}
	  {“node1”: “<walk>, <toilet>”, “children”: [] }
	  {“node2”: “<sit>, <toilet>”, “children”: []}

	Please refine2the goal-plan tree:
	Query:
	{“Root”: “evening routine”, “children”: [“node1”, “node2”, “node3”], “interchangeable groups”: [“group1”, “group2”]}
	  {“node1”: “watch TV”, “children”: [“node1-1”, “node1-2”, “node1-3”] }
	    {“node1-1”: “<walk>”, <couch>”, “children”: []}
	    {“node1-2: “<sit>, <couch>”, “children”: []}
	    {“node1-3”: “<press>”, <remote>”, “children”: []}
	  {“node2”: “have dinner”, “children”: [“node2-1”, “node2-2”, “node2-3”, “node2-4”, “node2-5”, “node2-6”] }
	    {“node2-1”: “<walk>, <refrigerator>”, “children”: [] }
	    {“node2-2”: “<open>, <refrigerator>”, “children”: [] }
	    {“node2-3”: “<take>, <food in refrigerator>“, “children“: [] }
	    {“node2-4“: “<walk>, <dining chair>“, “children“: [] }
	    {“node2-5“: “<sit>, <dining chair>“, “children“: [] }
	    {“node2-6“: “<eat>, <food in refrigerator>“, “children“: [] }
	  {“node3”: “watch TV”, “children”: [“node3-1”, “node3-2”] }
	    {“node3-1”: “<walk>, <bed>“, “children“: [] }
	    {“node3-2”: “<lie>, <bed>“, “children“: [] }
	  {“group1“: [{“node1“}, {“node2“}]}
	  {“group2“:[{“node3“}]}
	Response:
	{“Root”: “engage in the rituals of dusk”, “children”: [“node1”, “node2”, “node3”], “interchangeable groups”: [“group1”, “group2”]}
	  {“node1”: “indulge in the visual leisure”, “children”: [“node1-1”, “node1-2”, “node1-3”] }
	    {“node1-1“: “<walk>, <couch>“, “children“: []}
	    {“node1-2“: “<sit>, <couch>“, “children“: []}
	    {“node1-3“: “<press>, <remote>“, “children“: []}
	  {“node2”: “partake in the evening nourishment”, “children”: [“node2-1”, “node2-2”, “node2-3”, “node2-4”, “node2-5”, “node2-6”] }
	    {“node2-1“: “<walk>, <refrigerator>“, “children“: []}
	    {“node2-2“: “<open>, <refrigerator>“, “children“: []}
	    {“node2-3“: “<take>, <food in refrigerator>“, “children“: []}
	    {“node2-4“: “<walk>, <dining chair>“, “children“: []}
	    {“node2-5“: “<sit>, <dining chair>“, “children“: []}
	    {“node2-6“: “<eat>, <food in refrigerator>“, “children“: []}
	  {“node3”: “embrace the rituals preceding slumber”, “children”: [“node3-1”, “node3-2”] }
	    {“node3-1“: “<walk>, <bed>“, “children“: []}
	    {“node3-2“: “<lie>, <bed>“, “children“: []}
	  {“group1“: [{“node1“}, {“node2“}]}
	  {“group2“:[{“node3“}]}

Table S4. Top 100 objects by frequency in BEHAVIORHUB dataset.

Object List
Pillow, Door, Lamp, Floor, Window, Cabinet, Box, Book, Chair, Shelf, Table, Mirror, Curtain, Towel, Paint, Bag, Shoe, Clothes, Sink, Bed, Stairs, Toy, Tap, Cardboard Box,Rug, Toilet, Beam, Basket, Armchair, Wall Lamp, Drawer, Decoration, Shower Wall, Pipe, Wardrobe, Vase, Toilet Paper, Picture, Cushion, Bottle, TV, Carpet, Desk, Decorative Plant, Radiator, Door Knob, Ventilation, Blanket, Hanger, Blinds, Couch, Photo, Clutter, Stool, Trashcan, Container, Window Curtain, Appliance, Ornament, Flowerpot, Product, Candle, Device, Storage Box, Rack, Refrigerator, Nightstand, Dining Chair, Light Fixture, Support Beam, Basket of Something, Curtain Rod, Towel Bar, Vent, Bathroom Cabinet, Plate, Speaker, Heater, Window Glass, Kitchen Appliance, Bathroom Accessory, Faucet, Kitchen Lower Cabinet, Clock, Flower Vase, Board, Hanging Clothes, Cabinet Door, Cup, Table Lamp, Dresser, Air Vent, Case, Cloth, Bathtub, Bin, Flower, Can, Bowl, Cosmetics

Table S5. Top 50 actions by frequency in BEHAVIORHUB.

Action List
Walk, Sit, Stand Up, Move, Place, Open, Take, Clean, Jump, Run, Throw, Eat, Turn, Pick Up, Put On, Touch, Lift, Grasp, Dance, Knock, Yoga, Catch, Grab, Lie, Play, Shake, Hit, Drink, Stop, Give, Wash, Close, Relax, Remove, Rub, Check, Wait, Cut, Cook, Write, Tap, Press, Hang, Tie, Draw, Chop, Fill, Brush, Sleep, Flip

Example JSON - Prepare for the day.

```
[
  {
    Root: prepare for the
         day
    children: [
      {
        node1: getting
             dressed
        children: [
          {
            node1-1: <open>,
                  <wardrobe>
            children: []
          },
          {
            node1-2: <take>,
                  <clothes>
            children: []
          },
          {
            node1-3: <put on
                 >, <clothes
                 >
            children: []
          }
        ]
      },
      {
        node2: breakfast
             preparation
        children: [
          {
            node2-1: <walk>,
                  <kitchen>
            children: []
          },
          {
            node2-2: <open>,
                  <
                 refrigerator
                 >
            children: []
          },
          {
            node2-3: <take>,
                  <food in
                 refrigerator
                 >
            children: []
          },
          {
            node2-4: <walk>,
                  <dining
                 chair>
            children: []
          },
          {
            node2-5: <sit>,
                  <dining
                 chair>
            children: []
          },
          {
            node2-6: <eat>,
                 <food in
                 refrigerator
                 >
            children: []
          }
        ]
      },
      {
        node3: home
             cleaning
        children: [
          {
            node3-1: <walk>,
                  <bedroom>
            children: []
          },
          {
            node3-2: <take>,
                  <broom>
            children: []
          },
          {
            node3-3: <clean
                 >, <floor>
            children: []
          }
        ]
      },
      {
        node4: work from
             home
        children: [
          {
            node4-1: <walk>,
                  <computer
                 desk>
            children: []
          },
          {
            node4-2: <sit>,
                 <computer
                 chair>
            children: []
          },
          {
            node4-3: <typing
                 >, <
                 keyboard>
            children: []
          }
        ]
      }
    ],
    interchangeable
         groups: [
      {
        group1: [node1]
      },
      {
        group2: [node2,
             node3, node4]
      }
    ]
  }
]
```

Example JSON - Prepare dinner.

```
[
  {
    Root: prepare dinner
    children: [
      {
        node1: gather
             ingredients
        children: [
          {
            node1-1: <walk>,
                  <
                 refrigerator
                 >
            children: []
          },
          {
            node1-2: <open>,
                  <
                 refrigerator
                 >
            children: []
          },
          {
            node1-3: <take>,
                  <food in
                 refrigerator
                 >
            children: []
          }
        ]
      },
      {
        node2: cook
             ingredients
        children: [
          {
            node2-1: <walk>,
                  <stove>
            children: []
          },
          {
            node2-2: <place
                 >, <food in

                 refrigerator
                 >, <stove>
            children: []
          },
          {
            node2-3: <wait>
            children: []
          },
          {
            node2-4: <check
                 >, <food in

                 refrigerator
                 >
            children: []
          },
          {
            node2-5: <cut>,
                 <food in
                 refrigerator
                 >
            children: []
          },
          {
            node2-6: <cook>,
                  <food in
                 refrigerator
                 >
            children: []
          }
        ]
      },
      {
        node3: serve dinner
        children: [
          {
            node3-1: <walk>,
                  <dining
                 table>
            children: []
          },
          {
            node3-2: <sit>,
                 <dining
                 chair>
            children: []
          },
          {
            node3-3: <eat>,
                 <food in
                 refrigerator
                 >
            children: []
          }
        ]
      }
    ],
    interchangeable
         groups: [
      {
        group1: [node1]
      },
      {
        group2: [node2]
      },
      {
        group3: [node3]
      }
    ]
  }
]
```

Example JSON - Weekend cleaning.

```
[
 {
   Root: Weekend cleaning
   children: [
     {
       node1: clean
             bedroom
       children: [
         {
           node1-1: <walk>,
                   <bedroom>
           children: []
         },
         {
           node1-2: <take>,
                   <broom>
           children: []
         },
         {
           node1-3: <clean
                 >, <floor>
           children: []
         },
         {
           node1-4: <clean
                 >, <window>
           children: []
         },
         {
           node1-5: wash
                 clothes
           children: [
             {
               node1-5-1: <
                     walk>, <
                     washing
                     machine>
               children: []
             },
             {
               node1-5-2: <
                     place>,
                     <clothes
                     >, <
                     washing
                     machine>
               children: []
             }
           ]
         }
       ]
     },
     {
       node2: clean
             kitchen
       children: [
         {
           node2-1: <walk>,
                   <kitchen>
           children: []
         },
         {
           node2-2: <walk>,
                   <dining
                 table>
           children: []
         },
         {
           node2-3: <clean
                 >, <dining
                 table>
           children: []
         },
         {
           node2-4: wash
                 dishes
           children: [
             {
               node2-4-1: <
                     walk>, <
                     sink>
               children: []
             },
             {
               node2-4-2: <
                     wash>, <
                     kitchen
                     appliance
                     >
               children: []
             }
           ]
         }
       ]
     }
   ],
   interchangeable
       groups: [
     {
       group1: [node1,
             node2]
     }
   ]
 }
]
```

References

[1] Jingbo Wang, Yu Rong, Jingyuan Liu, Sijie Yan, Dahua Lin, and Bo Dai. Towards diverse and natural scene-aware 3d human motion synthesis. In *CVPR*, 2022. S1

[2] Korrawe Karunratanakul, Konpat Preechakul, Supasorn Suwajanakorn, and Siyu Tang. Guided motion diffusion for controllable human motion synthesis. In *ICCV*, 2023. S1

[3] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *ICCV*, 2021. S1

[4] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. S2

[5] Santhosh K. Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westry, Angel Xuan Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. In *NeurIPS Datasets and Benchmarks Track*, 2021. S2

[6] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. Amass: Archive of motion capture as surface shapes. In *ICCV*, 2019. S2

[7] Abhinanda R Punnakkal, Arjun Chandrasekaran, Nikos Athanasiou, Alejandra Quiros-Ramirez, and Michael J Black. Babel: Bodies, action and behavior with english labels. In *CVPR*, 2021. S2

[8] Omid Taheri, Nima Ghorbani, Michael J Black, and Dimitrios Tzionas. Grab: A dataset of whole-body human grasping of objects. In *ECCV*, 2020. S2