《Code as Policies: Language Model Programs for Embodied Control》复现报告

摘要

在代码补全上训练的大语言模型（LLMs）已被证明能够从文档字符串中合成简单的 Python 程序。这些代码编写者 LLMs 在接收到自然语言命令的情况下可以被重新用于编写机器人策略代码。具体来说，策略代码可以表达处理感知输出（例如来自对象检测器）的函数或反馈循环，并参数化控制原语 API。当提供几个示例语言命令（格式化为注释）和相应的策略代码（通过少样本提示）作为输入时，LLMs 可以接受新的命令并自主重新组合 API 调用以生成新的策略代码。通过链接经典逻辑结构和引用第三方库进行算术运算，这种 LLMs 可以编写出具有表现出空间几何推理、泛化到新的指令、根据上下文为模糊描述指定精确值特点的机器人策略。文章提出了"代码即策略"：一种以机器人为中心的、由语言模型生成的程序（LMPs）的表述方式，可以表示反应性策略（例如阻抗控制器），以及基于航点的策略（基于视觉的抓取和放置、基于轨迹的控制）。方法的核心是提示分层代码生成（递归定义未定义的函数），可以编写更复杂的代码。

关键词：LLMs；大模型；具身智能；层次化代码生成

1 引言

近年来， LLM 在代码生成、文本生成等领域取得了突破性进展，展现出强大的自然语言理解和生成能力。而传统的机器人语言控制方法存在语义理解困难、端到端学习数据需求量很大、LLM 的能力未能得到充分利用等的局限性。为了解决这些局限问题，文章提出 Code as Policies 方法，利用代码生成 LLM 直接生成机器人策略代码，利用 LLM 的能力实现了更灵活、更通用的机器人控制方法。CaP 方法为机器人与人类之间的自然语言交互提供了新的思路，并且无需额外训练，降低了机器人开发的成本，有望推动机器人技术的普及应用和人机交互技术的发展。

2 相关工作

2.1 通过语言控制机器人

通过语言控制机器人有着悠久的历史，包括早期通过自然语言的词汇解析来展示人机交互[2]。语言不仅作为非专家与机器人交互的界面，而且作为一种将泛化能力组合扩展到新任务的方法。相关文献非常丰富，但最近的研究大致可分为以下几类：高级解释（例如，语义

解析[3])、规划和低级策略。相比之下，文章的工作重点在于大型语言模型的代码生成方面，并使用生成的程序作为一种表达方式来控制机器人。

2.2 大语言模型

LLM 展现出令人印象深刻的零样本推理能力：规划、编写数学程序、解决科学问题等。与文章最密切相关的研究是利用 LLM 能力来构建机器人代理，而无需进行额外的模型训练。例如，Huang 等人通过文本补全和语义翻译将自然语言指令分解为可执行动作序列，而 SayCan[5]通过联合解码由价值函数加权的 LLM 和技能可供性来为机器人生成可行的计划。Inner Monologue[6]通过整合来自成功检测器或其他视觉语言模型的输出，并利用其反馈进行重新规划，扩展了 LLM 的规划能力。Socratic Models[7]使用视觉语言模型将感知信息替换到生成计划的语言提示中，并使用语言条件策略。CaP 不仅利用逻辑结构来指定反馈循环，而且还参数化（并编写部分）低级控制原语。CaP 减少了收集数据和训练一组预定义技能或语言条件策略的需要，这些策略既昂贵又往往特定于领域。

2.3 代码生成

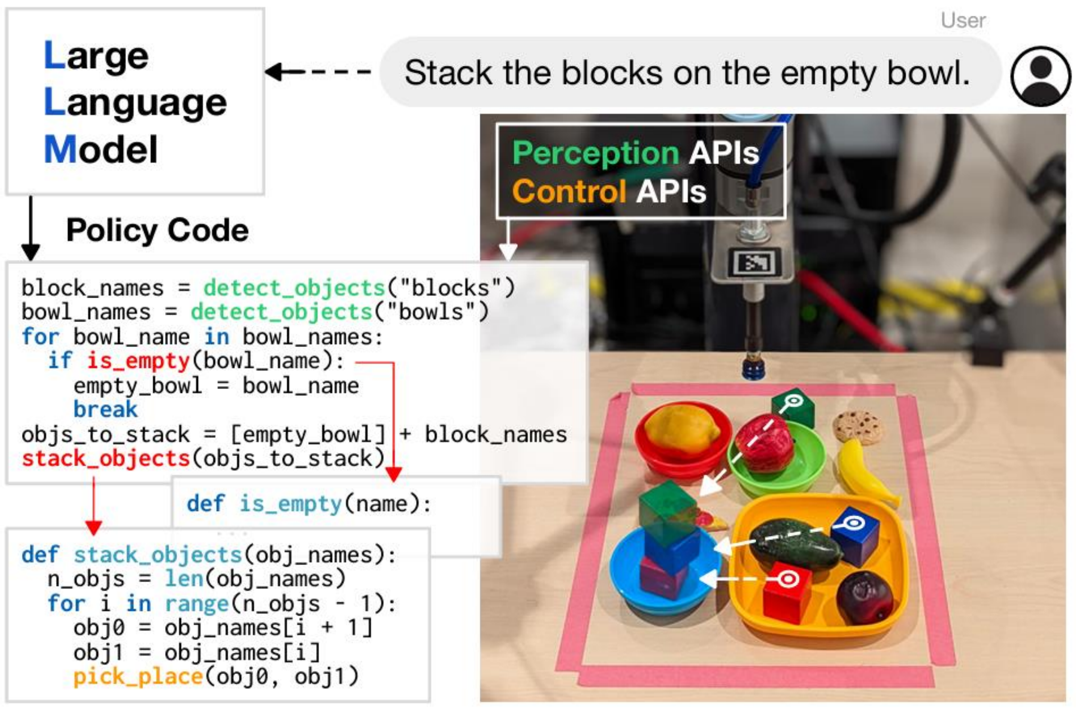
代码生成已在使用大型语言模型[8]和不使用大型语言模型[9]的情况下进行探索。程序合成已被证明能够绘制简单的图形并生成解决二维任务的策略。文章扩展了这些工作，表明代码编写大型语言模型能够实现新颖的推理能力（例如，通过依赖第三方库的熟悉程度来编码空间关系），而无需在先前工作中进行额外训练，以及分层代码编写。文章还提出了一个新的机器人主题代码生成基准，用于评估未来机器人领域的语言模型。

3 本文方法

3.1 本文方法概述

文章提出使用像 GPT-3 或 Codex 这样的 LLM 来生成可以作为控制机器人策略的程序。其思路是以代码注释的形式，使用自然语言指令提示 LLM，并提供一些示例，展示如何将指令映射到代码。然后，LLM 可以生成新的代码，处理感知输出，并调用控制原语来执行新的未见过的指令。生成的代码，被称为语言模型程序（Language Model Programs, LMPs），可以利用像循环和条件语句这样的逻辑结构创建反应策略。LMPs 还可以调用外部库，如 NumPy 进行空间推理。LMPs 是可执行的，可以直接控制真实的机器人。

文章还提出了层次化代码生成，程序解析生成的代码以找出任何未定义的函数，其中 LMPs 可以通过调用专门用于函数生成的另一个 LMPs 来生成这些函数。新创建的函数将添加到全局变量中以供将来重用。具体工作流程如图 1 所示。



```
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
  if is_empty(bowl_name):
    empty_bowl = bowl_name
    break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)
```

```
def is_empty(name):
```

```
def stack_objects(obj_names):
  n_objs = len(obj_names)
  for i in range(n_objs - 1):
    obj0 = obj_names[i + 1]
    obj1 = obj_names[i]
    pick_place(obj0, obj1)
```

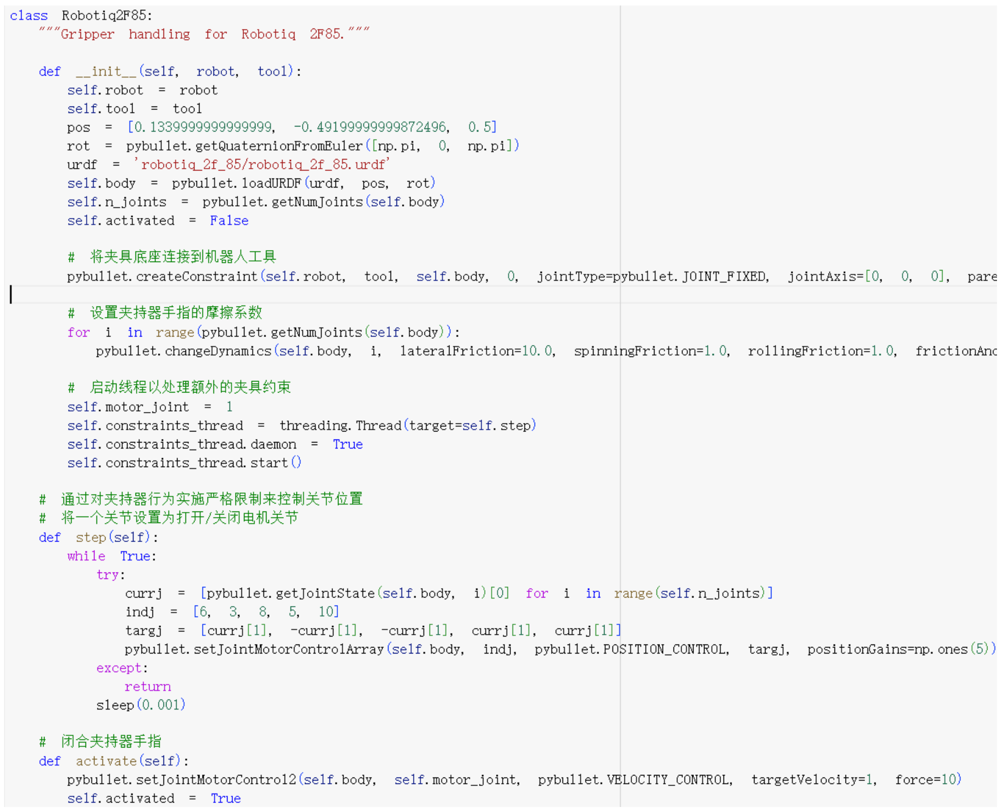
图 1 CaP 工作流程

4 复现细节

本次主要复现了论文在仿真环境进行桌面操作的实验，其代码主要包括以下部分：Robotiq2F85 类、PickPlaceEnv 类、LMP 类、LMPFGen 类、LMP_wrapper 类、LMP Prompts。

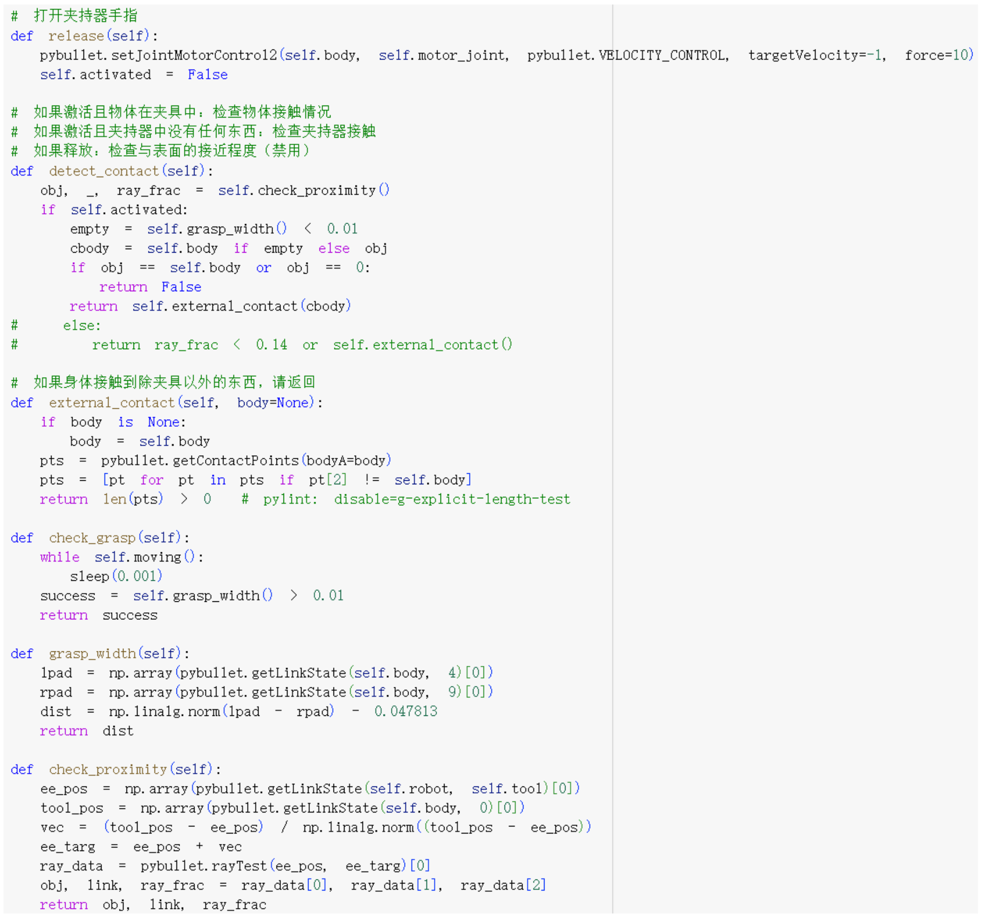
4.1 Robotiq2F85

如图 2 和图 3，Robotiq2F85 类用于定义机械臂抓取器，包含初始化、控制关节位置、闭合夹爪、打开夹爪等功能函数，用于控制抓取器动作。



```python
class Robotiq2F85:
    """Gripper handling for Robotiq 2F85."""

    def __init__(self, robot, tool):
        self.robot = robot
        self.tool = tool
        pos = [0.1339999999999999, -0.49199999999872496, 0.5]
        rot = pybullet.getQuaternionFromEuler([np.pi, 0, np.pi])
        urdf = 'robotiq_2f_85/robotiq_2f_85.urdf'
        self.body = pybullet.loadURDF(urdf, pos, rot)
        self.n_joints = pybullet.getNumJoints(self.body)
        self.activated = False

        # 将夹具底座连接到机器人工具
        pybullet.createConstraint(self.robot, tool, self.body, 0, jointType=pybullet.JOINT_FIXED, jointAxis=[0, 0, 0], pare

        # 设置夹持器手指的摩擦系数
        for i in range(pybullet.getNumJoints(self.body)):
            pybullet.changeDynamics(self.body, i, lateralFriction=10.0, spinningFriction=1.0, rollingFriction=1.0, frictionAnc

        # 启动线程以处理额外的夹具约束
        self.motor_joint = 1
        self.constraints_thread = threading.Thread(target=self.step)
        self.constraints_thread.daemon = True
        self.constraints_thread.start()

    # 通过对夹持器行为实施严格限制来控制关节位置
    # 将一个关节设置为打开/关闭电机关节
    def step(self):
        while True:
            try:
                currj = [pybullet.getJointState(self.body, i)[0] for i in range(self.n_joints)]
                indj = [6, 3, 8, 5, 10]
                targj = [currj[1], -currj[1], -currj[1], currj[1], currj[1]]
                pybullet.setJointMotorControlArray(self.body, indj, pybullet.POSITION_CONTROL, targj, positionGains=np.ones(5))
            except:
                return
            sleep(0.001)

    # 闭合夹持器手指
    def activate(self):
        pybullet.setJointMotorControl2(self.body, self.motor_joint, pybullet.VELOCITY_CONTROL, targetVelocity=1, force=10)
        self.activated = True
```

图 2 Robotiq2F85 1

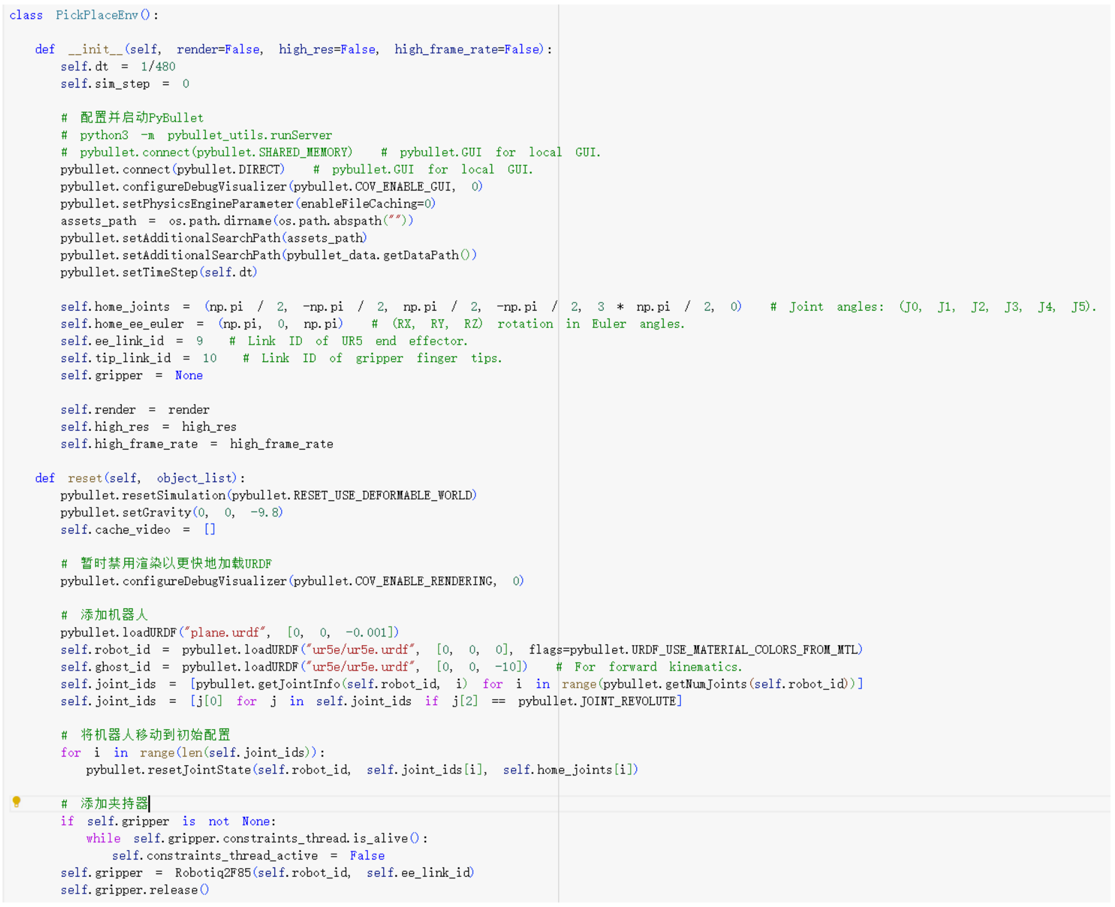


```python
# 打开夹持器手指
def release(self):
    pybullet.setJointMotorControl2(self.body, self.motor_joint, pybullet.VELOCITY_CONTROL, targetVelocity=-1, force=10)
    self.activated = False

# 如果激活且物体在夹具中：检查物体接触情况
# 如果激活且夹持器中没有任何东西：检查夹持器接触
# 如果释放：检查与表面的接近程度（禁用）
def detect_contact(self):
    obj, _, ray_frac = self.check_proximity()
    if self.activated:
        empty = self.grasp_width() < 0.01
        cbody = self.body if empty else obj
        if obj == self.body or obj == 0:
            return False
        return self.external_contact(cbody)
#    else:
#        return ray_frac < 0.14 or self.external_contact()

# 如果身体接触到除夹具以外的东西，请返回
def external_contact(self, body=None):
    if body is None:
        body = self.body
    pts = pybullet.getContactPoints(bodyA=body)
    pts = [pt for pt in pts if pt[2] != self.body]
    return len(pts) > 0  # pylint: disable=g-explicit-length-test

def check_grasp(self):
    while self.moving():
        sleep(0.001)
    success = self.grasp_width() > 0.01
    return success

def grasp_width(self):
    lpad = np.array(pybullet.getLinkState(self.body, 4)[0])
    rpad = np.array(pybullet.getLinkState(self.body, 9)[0])
    dist = np.linalg.norm(lpad - rpad) - 0.047813
    return dist

def check_proximity(self):
    ee_pos = np.array(pybullet.getLinkState(self.robot, self.tool)[0])
    tool_pos = np.array(pybullet.getLinkState(self.body, 0)[0])
    vec = (tool_pos - ee_pos) / np.linalg.norm((tool_pos - ee_pos))
    ee_targ = ee_pos + vec
    ray_data = pybullet.rayTest(ee_pos, ee_targ)[0]
    obj, link, ray_frac = ray_data[0], ray_data[1], ray_data[2]
    return obj, link, ray_frac
```

图 3 Robotiq2F85 2

4.2 PickPlaceEnv

如图 4，PickPlaceEnv 类用于定义模拟环境，该类基于 PyBullet 物理引擎，它提供了一个包含一个机器人、一些物体和一个工作空间的模拟环境，包含了初始化环境、执行动作、渲染图象等功能函数，用于模拟桌面操控任务。

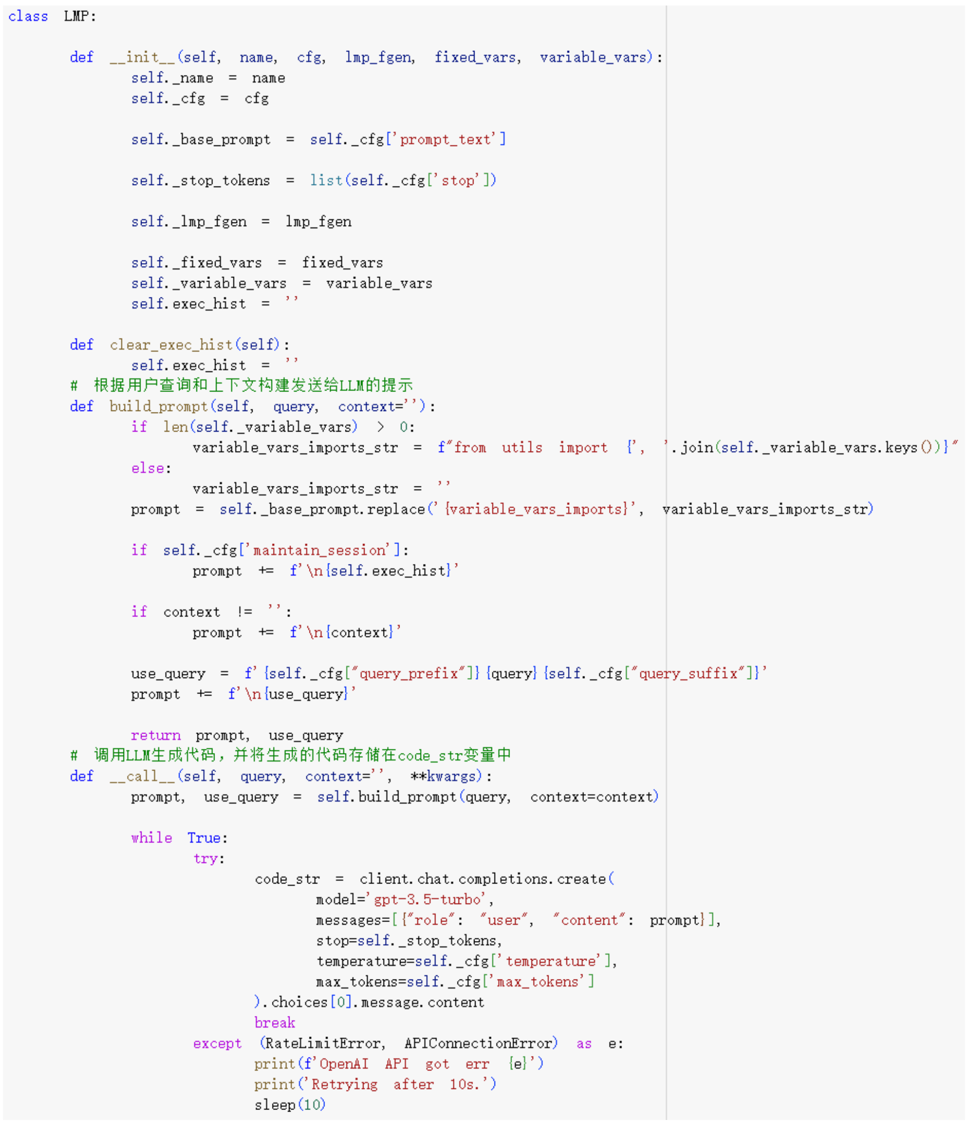


```
class PickPlaceEnv():

  def __init__(self, render=False, high_res=False, high_frame_rate=False):
    self.dt = 1/480
    self.sim_step = 0

    # 配置并启动PyBullet
    # python3 -m pybullet_utils.runServer
    # pybullet.connect(pybullet.SHARED_MEMORY)  # pybullet.GUI for local GUI.
    pybullet.connect(pybullet.DIRECT)  # pybullet.GUI for local GUI.
    pybullet.configureDebugVisualizer(pybullet.COV_ENABLE_GUI, 0)
    pybullet.setPhysicsEngineParameter(enableFileCaching=0)
    assets_path = os.path.dirname(os.path.abspath(""))
    pybullet.setAdditionalSearchPath(assets_path)
    pybullet.setAdditionalSearchPath(pybullet_data.getDataPath())
    pybullet.setTimeStep(self.dt)

    self.home_joints = (np.pi / 2, -np.pi / 2, np.pi / 2, -np.pi / 2, 3 * np.pi / 2, 0)  # Joint angles: (J0, J1, J2, J3, J4, J5).
    self.home_ee_euler = (np.pi, 0, np.pi)  # (RX, RY, RZ) rotation in Euler angles.
    self.ee_link_id = 9  # Link ID of UR5 end effector.
    self.tip_link_id = 10  # Link ID of gripper finger tips.
    self.gripper = None

    self.render = render
    self.high_res = high_res
    self.high_frame_rate = high_frame_rate

  def reset(self, object_list):
    pybullet.resetSimulation(pybullet.RESET_USE_DEFORMABLE_WORLD)
    pybullet.setGravity(0, 0, -9.8)
    self.cache_video = []

    # 暂时禁用渲染以更快地加载URDF
    pybullet.configureDebugVisualizer(pybullet.COV_ENABLE_RENDERING, 0)

    # 添加机器人
    pybullet.loadURDF("plane.urdf", [0, 0, -0.001])
    self.robot_id = pybullet.loadURDF("ur5e/ur5e.urdf", [0, 0, 0], flags=pybullet.URDF_USE_MATERIAL_COLORS_FROM_MTL)
    self.ghost_id = pybullet.loadURDF("ur5e/ur5e.urdf", [0, 0, -10])  # For forward kinematics.
    self.joint_ids = [pybullet.getJointInfo(self.robot_id, i) for i in range(pybullet.getNumJoints(self.robot_id))]
    self.joint_ids = [j[0] for j in self.joint_ids if j[2] == pybullet.JOINT_REVOLUTE]

    # 将机器人移动到初始配置
    for i in range(len(self.joint_ids)):
      pybullet.resetJointState(self.robot_id, self.joint_ids[i], self.home_joints[i])

    # 添加夹持器
    if self.gripper is not None:
      while self.gripper.constraints_thread.is_alive():
        self.constraints_thread_active = False
    self.gripper = Robotiq2F85(self.robot_id, self.ee_link_id)
    self.gripper.release()
```

图 4 部分 PickPlaceEnv

4.3 LMP

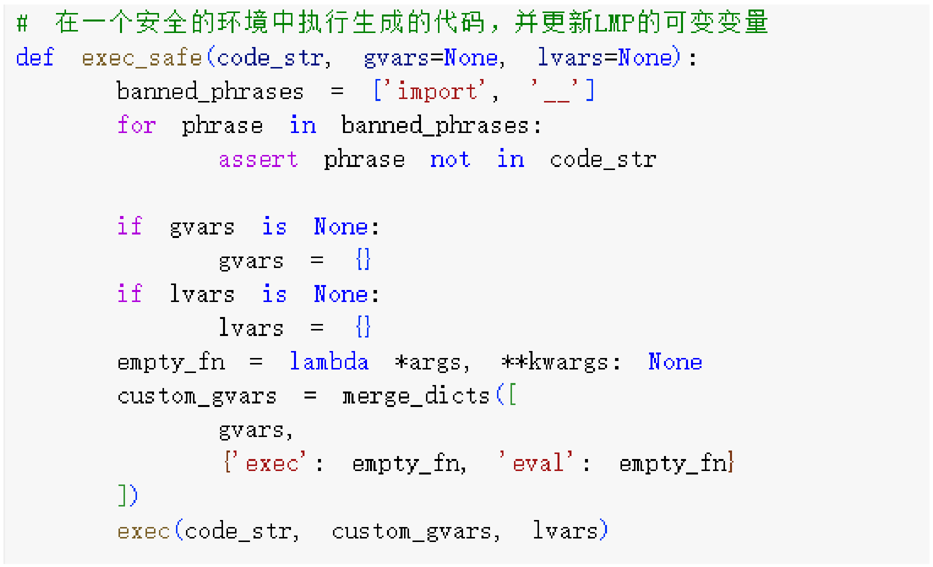
LMP 类是表示和执行语言模型程序的核心，该类封装了生成提示语、调用 LLM 生成代码和执行等功能函数。源代码使用了 codex 大模型来生成 LMP，本次复现改为使用 gpt-3.5-turbo，如图 5。



```python
class LMP:

    def __init__(self, name, cfg, lmp_fgen, fixed_vars, variable_vars):
        self._name = name
        self._cfg = cfg

        self._base_prompt = self._cfg['prompt_text']

        self._stop_tokens = list(self._cfg['stop'])

        self._lmp_fgen = lmp_fgen

        self._fixed_vars = fixed_vars
        self._variable_vars = variable_vars
        self.exec_hist = ''

    def clear_exec_hist(self):
        self.exec_hist = ''
    # 根据用户查询和上下文构建发送给LLM的提示
    def build_prompt(self, query, context=''):
        if len(self._variable_vars) > 0:
            variable_vars_imports_str = f"from utils import {', '.join(self._variable_vars.keys())}"
        else:
            variable_vars_imports_str = ''
        prompt = self._base_prompt.replace('{variable_vars_imports}', variable_vars_imports_str)

        if self._cfg['maintain_session']:
            prompt += f'\n{self.exec_hist}'

        if context != '':
            prompt += f'\n{context}'

        use_query = f'{self._cfg["query_prefix"]}{query}{self._cfg["query_suffix"]}'
        prompt += f'\n{use_query}'

        return prompt, use_query
    # 调用LLM生成代码，并将生成的代码存储在code_str变量中
    def __call__(self, query, context='', **kwargs):
        prompt, use_query = self.build_prompt(query, context=context)

        while True:
            try:
                code_str = client.chat.completions.create(
                    model='gpt-3.5-turbo',
                    messages=[{"role": "user", "content": prompt}],
                    stop=self._stop_tokens,
                    temperature=self._cfg['temperature'],
                    max_tokens=self._cfg['max_tokens']
                ).choices[0].message.content
                break
            except (RateLimitError, APIConnectionError) as e:
                print(f'OpenAI API got err {e}')
                print('Retrying after 10s.')
                sleep(10)
```

图 5 LMP

还定义了辅助函数 exec_safe 用于执行 LMP，如图 6。

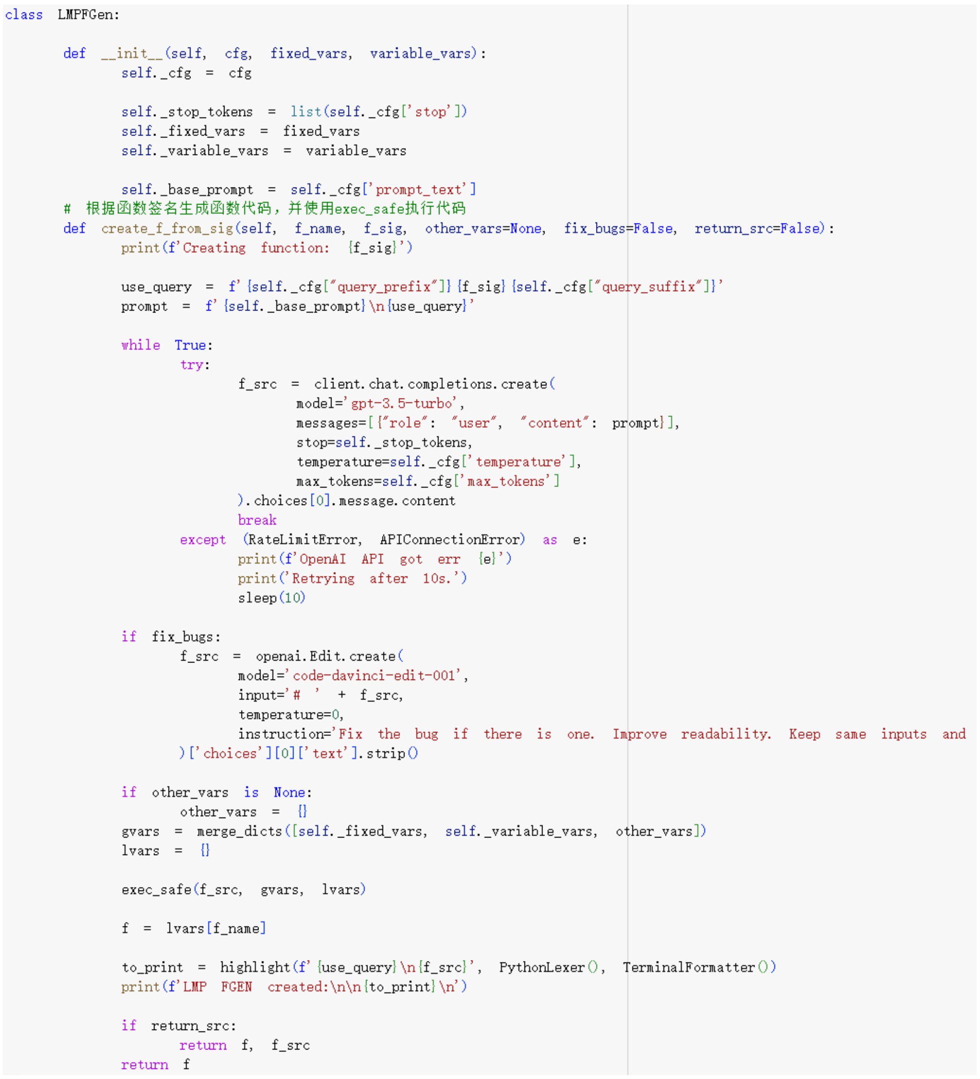


```python
# 在一个安全的环境中执行生成的代码，并更新LMP的可变变量
def exec_safe(code_str, gvars=None, lvars=None):
    banned_phrases = ['import', '__']
    for phrase in banned_phrases:
        assert phrase not in code_str

    if gvars is None:
        gvars = {}
    if lvars is None:
        lvars = {}
    empty_fn = lambda *args, **kwargs: None
    custom_gvars = merge_dicts([
        gvars,
        {'exec': empty_fn, 'eval': empty_fn}
    ])
    exec(code_str, custom_gvars, lvars)
```

图 6 辅助函数 exec_safe

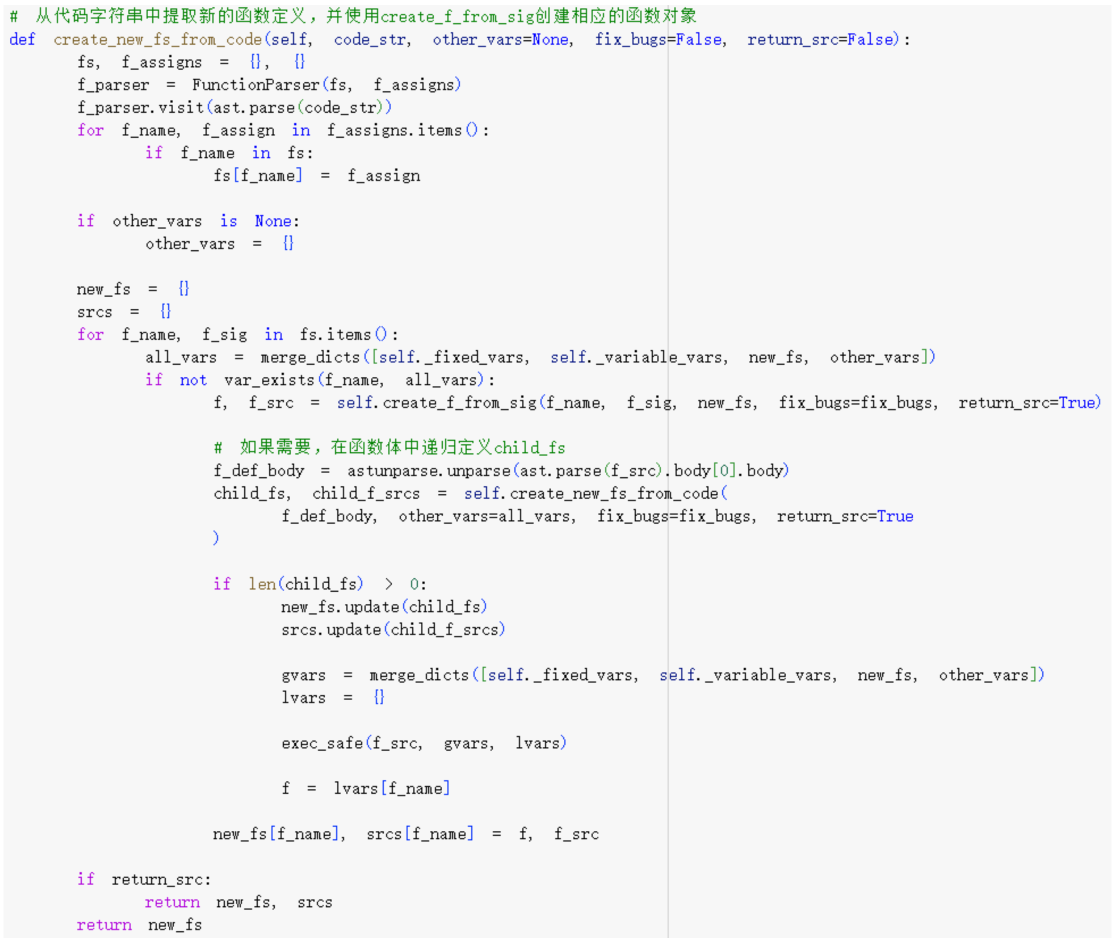
4.4 LMPFGen

LMPFGen 类的功能是从代码字符串中创建新函数，该类主要配合 LMP 类使用，在 LMP 的执行过程中动态生成新的函数。源代码使用 codex 来创建，本次复现同样改为使用 gpt-3.5-turbo，如图 7 和图 8。



```
class LMPFGen:

    def __init__(self, cfg, fixed_vars, variable_vars):
        self._cfg = cfg

        self._stop_tokens = list(self._cfg['stop'])
        self._fixed_vars = fixed_vars
        self._variable_vars = variable_vars

        self._base_prompt = self._cfg['prompt_text']
    # 根据函数签名生成函数代码，并使用exec_safe执行代码
    def create_f_from_sig(self, f_name, f_sig, other_vars=None, fix_bugs=False, return_src=False):
        print(f'Creating function: {f_sig}')

        use_query = f'{self._cfg["query_prefix"]}{f_sig}{self._cfg["query_suffix"]}'
        prompt = f'{self._base_prompt}\n{use_query}'

        while True:
            try:
                f_src = client.chat.completions.create(
                    model='gpt-3.5-turbo',
                    messages=[{"role": "user", "content": prompt}],
                    stop=self._stop_tokens,
                    temperature=self._cfg['temperature'],
                    max_tokens=self._cfg['max_tokens']
                ).choices[0].message.content
                break
            except (RateLimitError, APIConnectionError) as e:
                print(f'OpenAI API got err {e}')
                print('Retrying after 10s.')
                sleep(10)

        if fix_bugs:
            f_src = openai.Edit.create(
                model='code-davinci-edit-001',
                input='# ' + f_src,
                temperature=0,
                instruction='Fix the bug if there is one. Improve readability. Keep same inputs and
            )['choices'][0]['text'].strip()

        if other_vars is None:
            other_vars = {}
        gvars = merge_dicts([self._fixed_vars, self._variable_vars, other_vars])
        lvars = {}

        exec_safe(f_src, gvars, lvars)

        f = lvars[f_name]

        to_print = highlight(f'{use_query}\n{f_src}', PythonLexer(), TerminalFormatter())
        print(f'LMP FGEN created:\n\n{to_print}\n')

        if return_src:
            return f, f_src
        return f
```

图 7 LMPFGen 1

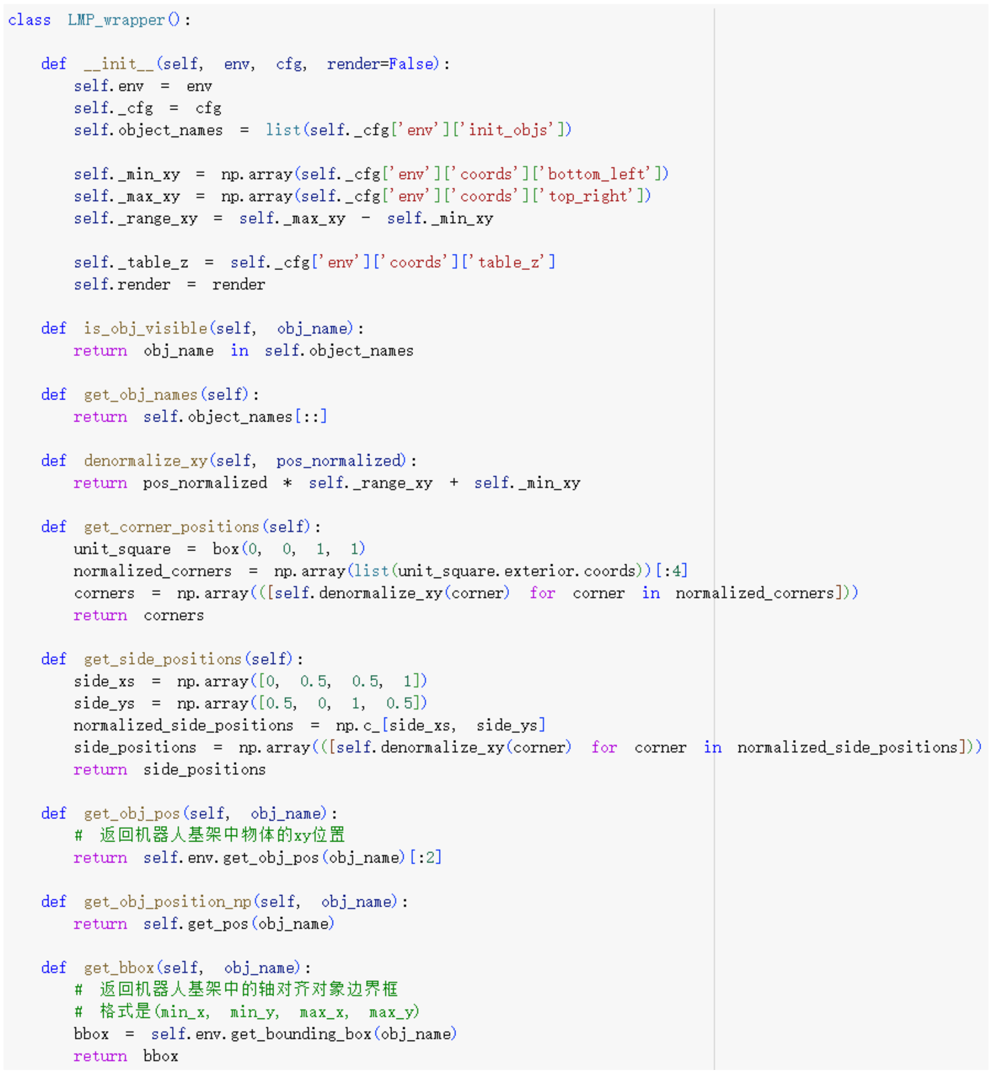


```python
# 从代码字符串中提取新的函数定义，并使用create_f_from_sig创建相应的函数对象
def create_new_fs_from_code(self, code_str, other_vars=None, fix_bugs=False, return_src=False):
    fs, f_assigns = {}, {}
    f_parser = FunctionParser(fs, f_assigns)
    f_parser.visit(ast.parse(code_str))
    for f_name, f_assign in f_assigns.items():
        if f_name in fs:
            fs[f_name] = f_assign

    if other_vars is None:
        other_vars = {}

    new_fs = {}
    srcs = {}
    for f_name, f_sig in fs.items():
        all_vars = merge_dicts([self._fixed_vars, self._variable_vars, new_fs, other_vars])
        if not var_exists(f_name, all_vars):
            f, f_src = self.create_f_from_sig(f_name, f_sig, new_fs, fix_bugs=fix_bugs, return_src=True)

            # 如果需要，在函数体中递归定义child_fs
            f_def_body = astunparse.unparse(ast.parse(f_src).body[0].body)
            child_fs, child_f_srcs = self.create_new_fs_from_code(
                f_def_body, other_vars=all_vars, fix_bugs=fix_bugs, return_src=True
            )

            if len(child_fs) > 0:
                new_fs.update(child_fs)
                srcs.update(child_f_srcs)

                gvars = merge_dicts([self._fixed_vars, self._variable_vars, new_fs, other_vars])
                lvars = {}

                exec_safe(f_src, gvars, lvars)

                f = lvars[f_name]

            new_fs[f_name], srcs[f_name] = f, f_src

    if return_src:
        return new_fs, srcs
    return new_fs
```

图 8 LMPFGen 2

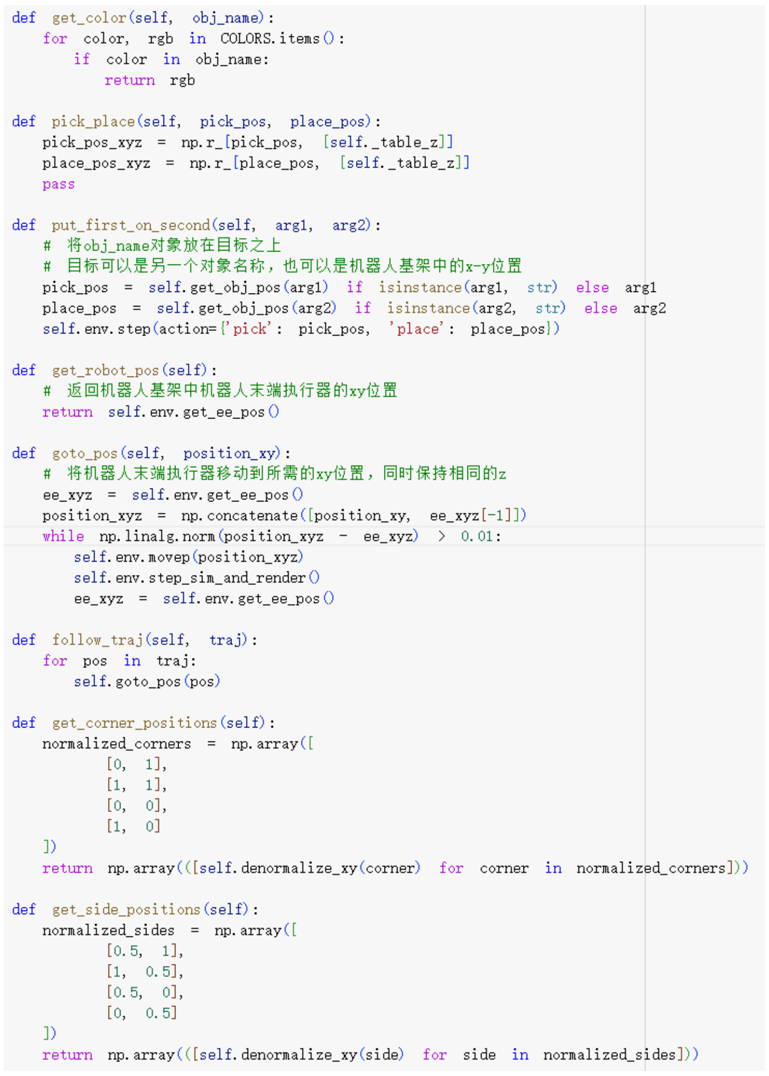
4.5 LMP_wrapper

LMP_wrapper 类用于连接 LMP 和模拟环境的适配器，该类将 LMP 生成的代码转换成模拟环境中可以理解的动作，并提供了执行动作、获取位置信息、坐标转化、获取物体名称等辅助函数，方便 LMP 和环境交互，如图 9 和图 10。



```python
class LMP_wrapper():

    def __init__(self, env, cfg, render=False):
        self.env = env
        self._cfg = cfg
        self.object_names = list(self._cfg['env']['init_objs'])

        self._min_xy = np.array(self._cfg['env']['coords']['bottom_left'])
        self._max_xy = np.array(self._cfg['env']['coords']['top_right'])
        self._range_xy = self._max_xy - self._min_xy

        self._table_z = self._cfg['env']['coords']['table_z']
        self.render = render

    def is_obj_visible(self, obj_name):
        return obj_name in self.object_names

    def get_obj_names(self):
        return self.object_names[::]

    def denormalize_xy(self, pos_normalized):
        return pos_normalized * self._range_xy + self._min_xy

    def get_corner_positions(self):
        unit_square = box(0, 0, 1, 1)
        normalized_corners = np.array(list(unit_square.exterior.coords))[:4]
        corners = np.array(([self.denormalize_xy(corner) for corner in normalized_corners]))
        return corners

    def get_side_positions(self):
        side_xs = np.array([0, 0.5, 0.5, 1])
        side_ys = np.array([0.5, 0, 1, 0.5])
        normalized_side_positions = np.c_[side_xs, side_ys]
        side_positions = np.array(([self.denormalize_xy(corner) for corner in normalized_side_positions]))
        return side_positions

    def get_obj_pos(self, obj_name):
        # 返回机器人基架中物体的xy位置
        return self.env.get_obj_pos(obj_name)[:2]

    def get_obj_position_np(self, obj_name):
        return self.get_pos(obj_name)

    def get_bbox(self, obj_name):
        # 返回机器人基架中的轴对齐对象边界框
        # 格式是(min_x, min_y, max_x, max_y)
        bbox = self.env.get_bounding_box(obj_name)
        return bbox
```

图 9 LMP_wrapper 1

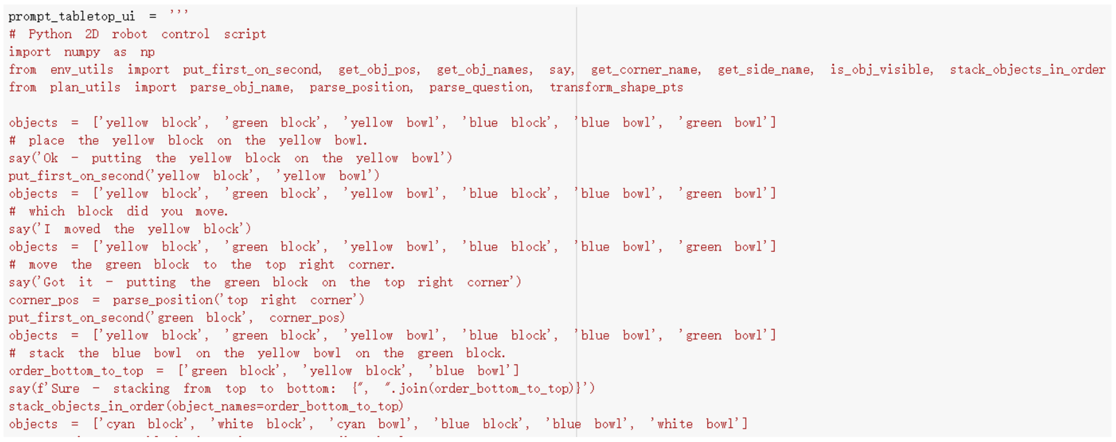


```
def  get_color(self,  obj_name):
    for  color,  rgb  in  COLORS.items():
        if  color  in  obj_name:
            return  rgb

def  pick_place(self,  pick_pos,  place_pos):
    pick_pos_xyz  =  np.r_[pick_pos,  [self._table_z]]
    place_pos_xyz  =  np.r_[place_pos,  [self._table_z]]
    pass

def  put_first_on_second(self,  arg1,  arg2):
    #  将obj_name对象放在目标之上
    #  目标可以是另一个对象名称，也可以是机器人基架中的x-y位置
    pick_pos  =  self.get_obj_pos(arg1)  if  isinstance(arg1,  str)  else  arg1
    place_pos  =  self.get_obj_pos(arg2)  if  isinstance(arg2,  str)  else  arg2
    self.env.step(action={'pick':  pick_pos,  'place':  place_pos})

def  get_robot_pos(self):
    #  返回机器人基架中机器人末端执行器的xy位置
    return  self.env.get_ee_pos()

def  goto_pos(self,  position_xy):
    #  将机器人末端执行器移动到所需的xy位置，同时保持相同的z
    ee_xyz  =  self.env.get_ee_pos()
    position_xyz  =  np.concatenate([position_xy,  ee_xyz[-1]])
    while  np.linalg.norm(position_xyz  -  ee_xyz)  >  0.01:
        self.env.movep(position_xyz)
        self.env.step_sim_and_render()
        ee_xyz  =  self.env.get_ee_pos()

def  follow_traj(self,  traj):
    for  pos  in  traj:
        self.goto_pos(pos)

def  get_corner_positions(self):
    normalized_corners  =  np.array([
            [0,  1],
            [1,  1],
            [0,  0],
            [1,  0]
    ])
    return  np.array(([self.denormalize_xy(corner)  for  corner  in  normalized_corners]))

def  get_side_positions(self):
    normalized_sides  =  np.array([
            [0.5,  1],
            [1,  0.5],
            [0.5,  0],
            [0,  0.5]
    ])
    return  np.array(([self.denormalize_xy(side)  for  side  in  normalized_sides]))
```

图 10 LMP_wrapper 2

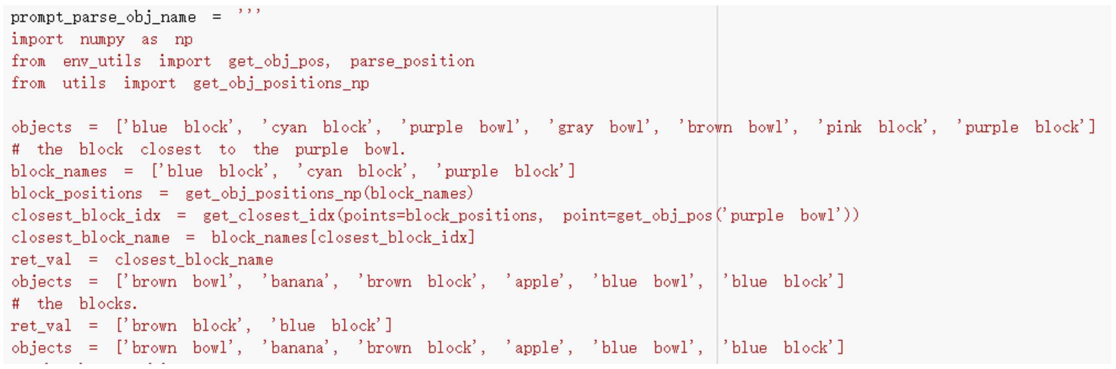
4.6 LMP Prompts

定义了 prompt_tabletop_ui、prompt_parse_obj_name、prompt_parse_position 等字符串，作为 LMP 的初始提示，帮助 LMP 更准确的理解用户指令，如图 11、图 12 和图 13。本次复现根据个人理解在源代码的提示词的基础上加了部分提示词，目的让大模型能更好的理解后续实验中未见过的指令或场景。



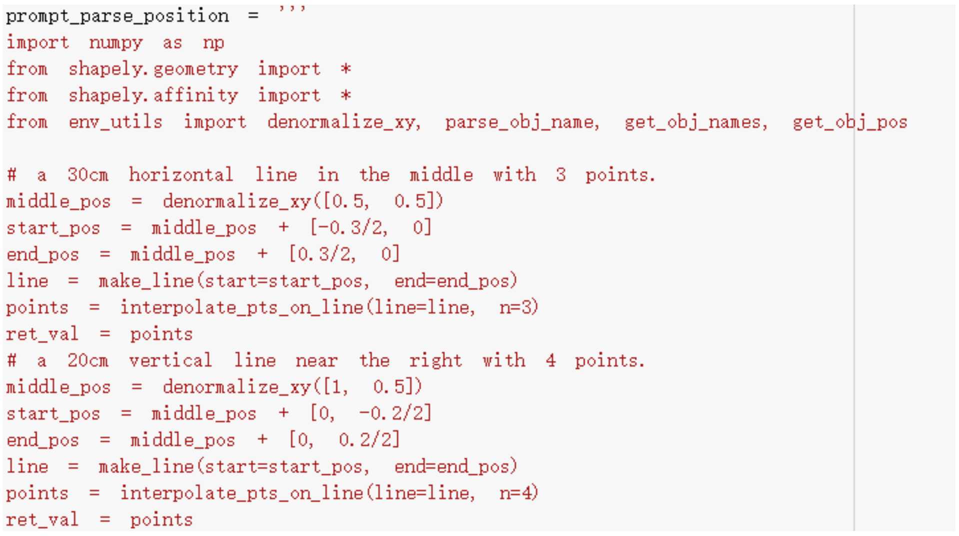
```
prompt_tabletop_ui  =  '''
#  Python  2D  robot  control  script
import  numpy  as  np
from  env_utils  import  put_first_on_second,  get_obj_pos,  get_obj_names,  say,  get_corner_name,  get_side_name,  is_obj_visible,  stack_objects_in_order
from  plan_utils  import  parse_obj_name,  parse_position,  parse_question,  transform_shape_pts

objects  =  ['yellow  block',  'green  block',  'yellow  bowl',  'blue  block',  'blue  bowl',  'green  bowl']
#  place  the  yellow  block  on  the  yellow  bowl.
say('Ok  -  putting  the  yellow  block  on  the  yellow  bowl')
put_first_on_second('yellow  block',  'yellow  bowl')
objects  =  ['yellow  block',  'green  block',  'yellow  bowl',  'blue  block',  'blue  bowl',  'green  bowl']
#  which  block  did  you  move.
say('I  moved  the  yellow  block')
objects  =  ['yellow  block',  'green  block',  'yellow  bowl',  'blue  block',  'blue  bowl',  'green  bowl']
#  move  the  green  block  to  the  top  right  corner.
say('Got  it  -  putting  the  green  block  on  the  top  right  corner')
corner_pos  =  parse_position('top  right  corner')
put_first_on_second('green  block',  corner_pos)
objects  =  ['yellow  block',  'green  block',  'yellow  bowl',  'blue  block',  'blue  bowl',  'green  bowl']
#  stack  the  blue  bowl  on  the  yellow  bowl  on  the  green  block.
order_bottom_to_top  =  ['green  block',  'yellow  block',  'blue  bowl']
say(f'Sure  -  stacking  from  top  to  bottom:  {",  ".join(order_bottom_to_top)}')
stack_objects_in_order(object_names=order_bottom_to_top)
objects  =  ['cyan  block',  'white  block',  'cyan  bowl',  'blue  block',  'blue  bowl',  'white  bowl']
```

图 11 prompt_tabletop_ui



```
prompt_parse_obj_name = '''
import numpy as np
from env_utils import get_obj_pos, parse_position
from utils import get_obj_positions_np

objects = ['blue block', 'cyan block', 'purple bowl', 'gray bowl', 'brown bowl', 'pink block', 'purple block']
# the block closest to the purple bowl.
block_names = ['blue block', 'cyan block', 'purple block']
block_positions = get_obj_positions_np(block_names)
closest_block_idx = get_closest_idx(points=block_positions, point=get_obj_pos('purple bowl'))
closest_block_name = block_names[closest_block_idx]
ret_val = closest_block_name
objects = ['brown bowl', 'banana', 'brown block', 'apple', 'blue bowl', 'blue block']
# the blocks.
ret_val = ['brown block', 'blue block']
objects = ['brown bowl', 'banana', 'brown block', 'apple', 'blue bowl', 'blue block']
```

图 12　prompt_parse_obj_name



```
prompt_parse_position = '''
import numpy as np
from shapely.geometry import *
from shapely.affinity import *
from env_utils import denormalize_xy, parse_obj_name, get_obj_names, get_obj_pos

# a 30cm horizontal line in the middle with 3 points.
middle_pos = denormalize_xy([0.5, 0.5])
start_pos = middle_pos + [-0.3/2, 0]
end_pos = middle_pos + [0.3/2, 0]
line = make_line(start=start_pos, end=end_pos)
points = interpolate_pts_on_line(line=line, n=3)
ret_val = points
# a 20cm vertical line near the right with 4 points.
middle_pos = denormalize_xy([1, 0.5])
start_pos = middle_pos + [0, -0.2/2]
end_pos = middle_pos + [0, 0.2/2]
line = make_line(start=start_pos, end=end_pos)
points = interpolate_pts_on_line(line=line, n=4)
ret_val = points
```

图 13 prompt_parse_position

4.7 与已有开源代码对比

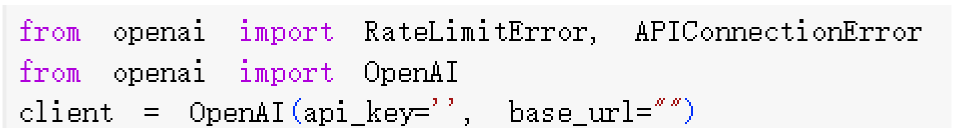
本次所复现的论文开源了它的代码，我参考了其代码并进行复现和修改，主要复现了论文在仿真环境进行桌面操作的实验。本次复现不仅完全实现了论文的功能，还对生成语言模型程序部分做了优化。在源代码中，作者使用 codex 模型来进行 LMP 程序的生成和层次生成新的 python 函数代码，而我在本次复现中改为使用 gpt-3.5-turbo 模型来完成这两步核心功能，并且在复现中添加了一些提示语以增强大模型的泛化能力，最后得出的实验结果模型也在未见过的场景和指令部分取得了更高的成功率。

4.8 实验环境搭建

本次复现只需 python 环境，安装相关包即可，可以直接在 colab 上运行。

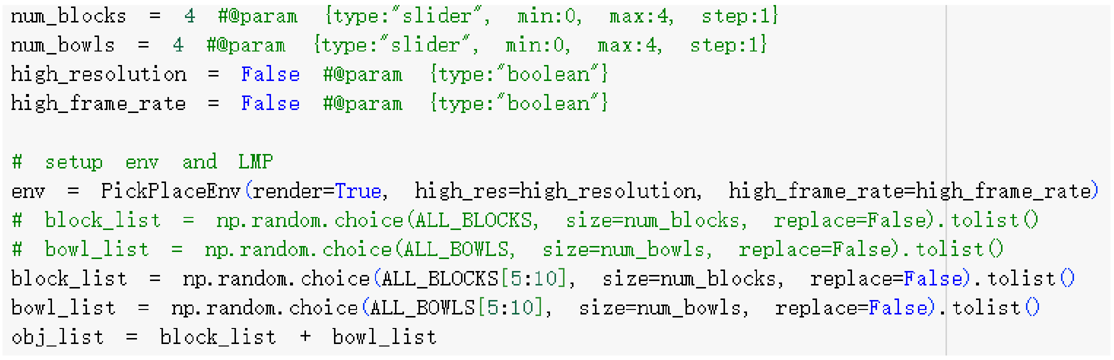
4.9 界面分析与使用说明

首先需要在环境配置中配置 openAI 的 api_key，如图 14，然后初始化环境，选择方块和碗的数量及颜色，如图 15，最后输入指令，如图 16。就可以运行代码，会自动生成机械臂的执行过程视频。



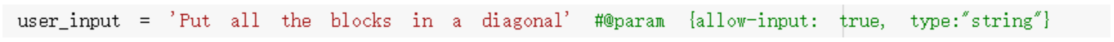
```
from  openai  import  RateLimitError,  APIConnectionError
from  openai  import  OpenAI
client  =  OpenAI(api_key='',  base_url="")
```

图 14 配置 openAI 的 api_key



```
num_blocks  =  4  #@param  {type:"slider",  min:0,  max:4,  step:1}
num_bowls  =  4  #@param  {type:"slider",  min:0,  max:4,  step:1}
high_resolution  =  False  #@param  {type:"boolean"}
high_frame_rate  =  False  #@param  {type:"boolean"}

#  setup  env  and  LMP
env  =  PickPlaceEnv(render=True,  high_res=high_resolution,  high_frame_rate=high_frame_rate)
#  block_list  =  np.random.choice(ALL_BLOCKS,  size=num_blocks,  replace=False).tolist()
#  bowl_list  =  np.random.choice(ALL_BOWLS,  size=num_bowls,  replace=False).tolist()
block_list  =  np.random.choice(ALL_BLOCKS[5:10],  size=num_blocks,  replace=False).tolist()
bowl_list  =  np.random.choice(ALL_BOWLS[5:10],  size=num_bowls,  replace=False).tolist()
obj_list  =  block_list  +  bowl_list
```

图 15 初始化环境，选择方块和碗的数量，并选择颜色，前五个是已见过的颜色，后五个是未见过的颜色



```
user_input  =  'Put  all  the  blocks  in  a  diagonal'  #@param  {allow-input:  true,  type:"string"}
```

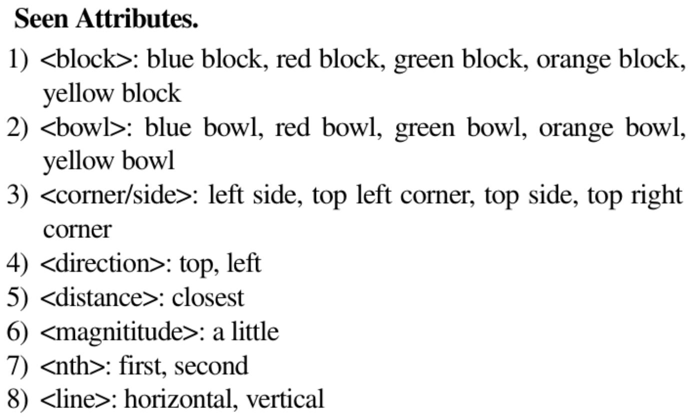
图 16 输入指令

4.10 创新点

本次复现的创新点主要在于对 LMP 程序的生成和层次生成新的 python 函数代码这两个核心功能做了优化。在源代码中，作者原本使用了 codex 模型完成这两个功能，而我本次复现调用 gpt-3.5-turbo 的接口，将此处功能交给更通用的大模型，并额外加了一些提示语，使得我们复现的代码具有更好的泛化性。

5 实验结果分析

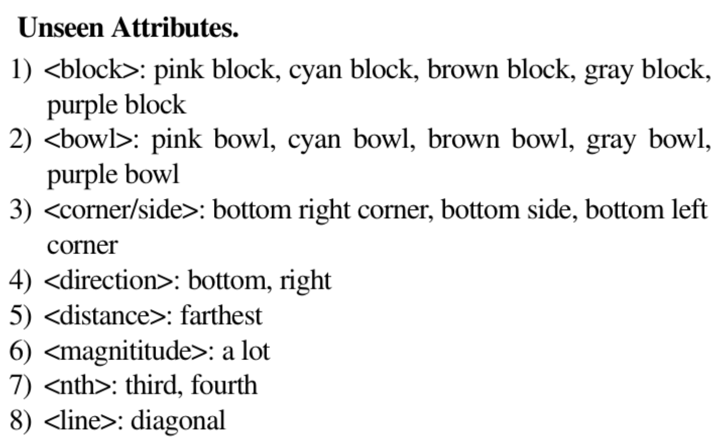
论文的桌面操作任务分为“已见”和“未见”类别的指令和属性，其中“已见”指令或属性允许出现在提示中或用于训练。并且进一步将指令分为“长视野”和“空间几何”任务系列。 “Long-Horizon”指令是已见指令中的 1-5 和未见指令中的 1-3。“Spatial-Geometric”指令在已见指令中为 5-8，在未见指令中为 4-6。已见未见属性如图 17 和图 18。文章的实验分为 SA SI、UA SI、UA UI 三个部分，各部分详细的成功率如表 1、表 2 和表 3，汇总成功率如表 4。



Seen Attributes.

1) <block>: blue block, red block, green block, orange block, yellow block
2) <bowl>: blue bowl, red bowl, green bowl, orange bowl, yellow bowl
3) <corner/side>: left side, top left corner, top side, top right corner
4) <direction>: top, left
5) <distance>: closest
6) <magnititude>: a little
7) <nth>: first, second
8) <line>: horizontal, vertical

图 17 已见属性



Unseen Attributes.

1) <block>: pink block, cyan block, brown block, gray block, purple block
2) <bowl>: pink bowl, cyan bowl, brown bowl, gray bowl, purple bowl
3) <corner/side>: bottom right corner, bottom side, bottom left corner
4) <direction>: bottom, right
5) <distance>: farthest
6) <magnititude>: a lot
7) <nth>: third, fourth
8) <line>: diagonal

图 18 未见属性

表 1 SA SI 操作成功率对比

Seen Attributes, Seen Instructions	Cap	Recurrence
Pick up the <object1> and place it on the (<object2>or<recepticle-bowl>)	100	100
Stack all the blocks	94	100
Put all the blocks on the <corner/side>	92	90
Put all the blocks in the <receptacle-bowl>	100	100
Put all the blocks in the bowls with matching colors	100	100
Pick up the block to the <direction> of the <recepticle-bowl> and place it on the <corner/side>	72	70
Pick up the block <distance> to the <receptacle-bowl> and place it on the <corner/side>	98	100
Pick up the <nth> block from the <direction> and place it on the <corner/side>	98	90
Total	94.3	93.75
Long-Horizon Total	97.2	98.0
Spatial-Geometric Total	89.3	86.7

表 2 UA SI 操作成功率对比

Unseen Attributes, Seen Instructions	Cap	Recurrence
Pick up the <object1> and place it on the (<object2>or<recepticle-bowl>)	100	100
Stack all the blocks	100	100
Put all the blocks on the <corner/side>	100	80
Put all the blocks in the <receptacle-bowl>	96	90
Put all the blocks in the bowls with matching colors	92	100
Pick up the block to the <direction> of the <recepticle-bowl> and place it on the <corner/side>	60	80
Pick up the block <distance> to the <receptacle-bowl> and place it on the <corner/side>	100	90
Pick up the <nth> block from the <direction> and place it on the <corner/side>	60	90
Total	88.5	91.25
Long-Horizon Total	97.6	94.0
Spatial-Geometric Total	73.3	90.0

表 3 UA UI 操作成功率对比

Unseen Attributes, Unseen Instructions	Cap	Recurrence
Put all the blocks in different corners	98	100
Put the blocks in the bowls with mismatched colors	60	80
Stack all the blocks on the <corner/side>	82	80
Pick up the <object1> and place it <magnitude> to the <direction> of the <receptacle-bowl>	38	60
Pick up the <object1> and place it in the corner <distance> to the <receptacle-bowl>	58	80
Put all the blocks in a <line>	90	100
Total	71.0	83.3
Long-Horizon Total	80.0	86.7
Spatial-Geometric Total	62.0	80.0

表 4 总成功率

Train/Test	Task Family	Cap	Recurrence
SA SI	Long-Horizon	97.2	**98.0**
SA SI	Spatial-Geometric	**89.3**	86.7
UA SI	Long-Horizon	**97.6**	94.0
UA SI	Spatial-Geometric	73.3	**90.0**
UA UI	Long-Horizon	80.0	**86.7**
UA UI	Spatial-Geometric	62.0	**80.0**

结果表明在已见过的指令和属性上的操作复现的成功率与原文相差不大，但是在未见过的指令和属性上的操作复现的成功率要高于原文，具有更强大的泛化能力。

6 总结与展望

总之，我们介绍了一种新的用大模型去直接将自然语言指令转换生成代码控制机器人的策略 CaP，提出了一个解决机器人操作领域如何将抽象的语言指令转化为具体的机器人动作的关键问题的可行方案，并且方法不需要额外的模型训练，生成的机器人策略代码结构清晰，易于理解维护。此方法在机器人操作领域也具有一定的应用前景和潜力，未来随着大模型的功能的更加强大或在此方法基础上再配合 VLM 等先进的模型相信此方法也能更具泛化性和通用性。

参考文献

[1] Liang J, Huang W, Xia F, et al. Code as policies: Language model programs for embodied control[C]//2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023: 9493-9500.

[2] Winograd T. Procedures as a representation for data in a computer program for understanding natural language[J]. 1971.

[3] Thomas K, Stefanie T, Deb R, Nicholas R, et al. Toward Understanding Natural Language Directions[C], Human-Robot Interaction, 2010.

[4] Wenlong H, Pieter A, Deepak P, Igor M, et al. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents.[J], Computing Research Repository, 2022: 9118-9147.

[5] Michael A, Anthony B, Noah B, Yevgen C, Omar C, Byron D, Chelsea F, Chuyuan F, Keerthana G, Karol H, Alex H, Daniel H, Jasmine H, Julian I, Brian I, Alex I, Eric J, Rosario J R, Kyle J, Sally J, Nikhil J J, Ryan J, Dmitry K, Yuheng K, Kuang-Huei L, Sergey L, Yao L, Linda L, Carolina P, Peter P, Jornell Q, Kanishka R, Jarek R, Diego R, Pierre S, Nicolas S, Clayton T, Alexander T, Vincent V, Fei X, Ted X, Peng X, Sichun X, Mengyuan Y, Andy Z, et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances[J], CONFERENCE ON ROBOT LEARNING, VOL 205, 2022, 205: 287-318.

[6] Wenlong H, Fei X, Ted X, Harris C, Jacky L, Pete F, Andy Z, Jonathan T, Igor M, Yevgen C, Pierre S, Noah B, Tomas J, Linda L, Sergey L, Karol H, Brian I, et al. Inner Monologue: Embodied Reasoning Through Planning with Language Models[C], Conference on Robot Learning, 2022, 205: 1769-1782.

[7] Andy Z, Maria A, Brian I, Krzysztof C, Adrian W, Stefan W, Federico T, Aveek P, Michael R, Vikas S, Johnny L, Vincent V, Pete F, et al. Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language.[J], Computing Research Repository, 2022, abs/2204.00598.

[8] Mark C, Jerry T, Heewoo J, Qiming Y, Henrique P D O P, Jared K, Harri E, Yuri B, Nicholas J, Greg B, Alex R, Raul P, Gretchen K, Michael P, Heidy K, Girish S, Pamela M, Brooke C, Scott G, Nick R, Mikhail P, Alethea P, Lukasz K, Mohammad B, Clemens W, Philippe T, Felipe P S, Dave C, Matthias P, Fotios C, Elizabeth B, Ariel H, William H G, Alex N, Alex P, Nikolas T, Jie T, Igor B, Suchir B, Shantanu J, William S, Christopher H, Andrew N C, Jan L, Josh A, Vedant M, Evan M, Alec R, Matthew K, Miles B, Mira M, Katie M, Peter W, Bob M, Dario A, Sam M, Ilya S, Wojciech Z, et al. Evaluating Large Language Models Trained on Code[J], arXivorg, 2021, abs/2107.03374.

[9] Kevin E, Lionel W, Maxwell N, Mathias S, Luc C, Lore A P, Luke H, Armando S, Joshua B T, et al. DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning[J], Philosophical Transactions of the Royal Society A Mathematical Physical and Engineering Sciences, 2023, 381(2251).