2024 年第二届大湾区杯科技竞赛

题　　号	**CG2409**		
标　　题	使用强化学习和有监督学习研究动作视觉任务		
成员信息	姓名：	单位：	邮箱：
	王仪瑾	华南师范大学	yjin_wang@163.com
	姓名：	单位：	邮箱：
	赵雨晴	华南师范大学	zhaoyuqing98@163.com
	姓名：	单位：	邮箱：
	王金泉	华南师范大学	904323192@qq.com

签名（可电子签名）：王仪瑾、赵雨晴、王金泉

摘要

本研究旨在探索深度学习方法在模拟人类认知行为过程中的应用效果。我们首先采用强化学习模型对经典的 Stroop 任务进行建模。在该任务中，模型需要学习在词义与颜色不一致的条件下做出正确反应。实验结果表明，强化学习模型能够成功复现人类在 Stroop 任务中表现出的行为模式，产生的数据与真实人类实验数据具有高度相似性。这一发现验证了强化学习在模拟基础认知过程中的有效性，为后续研究奠定了基础。

在此基础上，我们进一步设计了一个更复杂的动作-视觉判断任务，以探究动作系统与视觉系统之间的相互作用机制。在该任务中，被试需要首先完成一个手指点击动作，随后对呈现的光栅朝向（偏左或偏右）进行迫选。为了探究动作系统是否会向视觉系统传递信息，我们构建了两种理论模型：第一种模型基于感觉系统与运动系统分离的假设，认为视觉判断不受先前动作的影响；第二种模型则基于感觉系统与运动系统整合的假设，预测视觉判断会受到动作系统传出信息的调制。

我们分别使用强化学习和有监督学习方法构建的计算模型，其行为表现均未能支持上述任何一种理论假设，模型预测的行为模式与两种理论预期的结果都存在差异。这一结果表明，尽管深度学习方法在模拟简单认知任务（如 Stroop 任务）时表现出色，但在处理涉及多个认知系统交互的复杂任务时仍面临挑战。这可能反映了当前深度学习模型在模拟跨模态信息整合过程时的局限性，也提示我们需要开发新的模型架构或学习算法来更好地捕捉人类认知系统的复杂交互模式，尤其是来自于运动与视觉的交互。

一、问题分析

1.1 研究背景

在当代心理学研究领域，计算建模主要采用传统机器学习方法和贝叶斯推理框架。然而，随着深度学习技术的迅速发展和广泛应用，将这一前沿方法引入心理学实验范式的计算建模已成为一个极具前景的研究方向。深度学习不仅能够通过其强大的特征提取和模式识别能力来更精确地理解和预测人类行为，还可以借助其多层次的信息处理架构来揭示传统实验方法难以捕捉的潜在心理机制。这种新型的建模方法为心理学研究开辟了新的视角，有望带来方法论上的突破性进展。

1.2 研究目标

本项目旨在通过两个不同的实验范式，探索深度学习在心理学计算建模中的应用：

1.通过 Stroop 任务验证强化学习模型在模拟人类认知过程中的有效性

2.通过运动-视觉判断任务探索动作系统对视觉加工的影响机制

二、开发环境

编程语言：Python

深度学习框架：PyTorch

NumPy：用于数学运算

Matplotlib：用于数据可视化

Gym：用于创建和管理强化学习环境

三、Stroop 任务模型

3.1 Stroop 任务介绍

3.1.1 任务描述

Stroop 任务由美国心理学家 Stroop, J. R.在 1935 年首次提出 (Stroop, 1935)，是研究注意力和干扰效应的重要实验范式。其核心是探究人类在处理冲突信息时的认知

过程，特别是自动化过程与控制过程之间的相互作用。

在经典的 Stroop 任务中，实验材料为具有颜色意义的词语（如“Red”、“Green”、“Blue”），这些文字以不同的颜色呈现。实验包含以下两种主要条件：a. 一致性条件：词的意义与字体颜色一致。例如，红色的单词“Red”；b. 不一致条件：词的意义与字体颜色不一致。例如，绿色的单词“Red”。被试的任务是尽可能快速准确地命名呈现词语的墨水颜色，而忽略词语的语义内容。被试的任务是尽可能快速且准确地命名所呈现的词语的颜色，忽略词语的语义内容。

Stroop 效应指的是在不一致条件下，被试命名墨水颜色的反应时间显著长于一致性条件，且错误率更高。这一效应表明，自动化的阅读过程干扰了对墨水颜色的命名，需要额外的认知控制来抑制干扰。

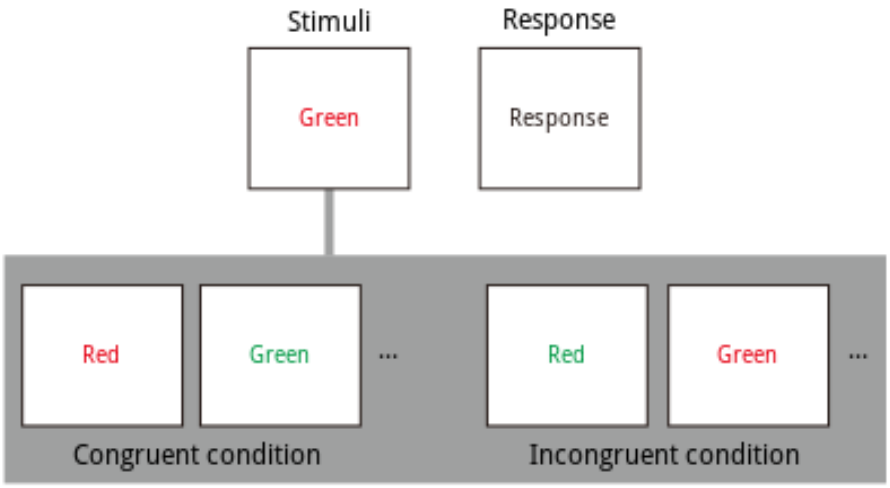

图 1 Stroop 任务流程图。在一致条件（Congruent condition）下，单词语义和单词颜色保持一致，如红色的“Red”和绿色的“Green”；在不一致条件（Incongruent condition）下，单词予以和单词颜色不一致，如绿色的“Red”和红色的“Green”。要求被试忽略单词语义，又快又准地报告单词颜色。

3.1.2 经典 Stroop 任务的心理学理论模型

(1) 加工速度理论（Processing Speed Theory）

加工速度理论始于 J.R. Stroop 最初对实验结果的解释 (Stroop, 1935)，并由后续的研究者进一步发展 (MacLeod, 1991)。根据该理论的观点，Stroop 效应是由于颜色命名和词汇阅读的加工速度不同引起的。对于人脑来说，词汇阅读是自动化程度较高的加工过程，因此阅读词语的速度比颜色命名更快。在 Stroop 任务中，要求被试命名颜色时，因为词汇阅读的自动化程度更高，加工速度更快，所以会对颜色命名产生干扰。

这个理论强调了不同信息处理过程的速度差异在 Stroop 效应中的关键作用。

(2) 自动化加工理论（Automaticity Theory）

自动化加工理论从词汇加工和颜色加工的自动化程度差异角度分析 Stroop 效应，提出阅读的自动化特性会对 Stroop 任务产生干扰。Posner 和 Snyder 等人认为，词汇阅读是一种自动化过程，不需要有意识的控制即可完成，而颜色命名则需要更高的认知控制，因此，在 Stroop 任务中，自动化的词汇阅读会干扰需要控制的颜色命名任务，从而导致反应时间增加和错误率提高（Solso, 1975; Logan, 1980）。

(3) 双重加工理论（Dual Process Theory）

双重加工理论始于 Dehaene 等人的研究（Dehaene & Changeux, 1989），该理论结合了自动化加工系统和控制加工系统来解释 Stroop 效应。理论认为 Stroop 任务的完成依赖于两个并行的加工过程：自动化的词汇阅读（自动化加工系统）和需要认知控制的颜色命名（控制加工系统）。当词汇与颜色不一致时，自动化加工系统会对控制加工系统造成干扰，需要控制加工系统加强认知控制来抑制自动化加工的干扰（Botvinick et al., 2001）。因此，任务的难度会增加，具体表现为 Stroop 任务中反应时间和错误率的显著提升。

3.2 强化学习实现

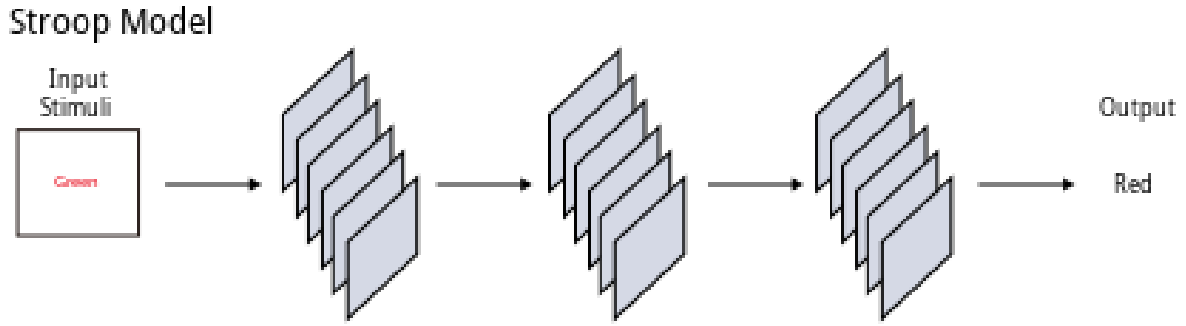

图 2 Stroop Model（DQN 网络）结构示意图。输入层包括颜色词和墨水颜色的特征编码；中间层包括多层神经网络处理单元；输出层包括反应时间预测和反应正确性判断；强化学习机制为通过奖惩信号优化决策过程。

3.2.1 任务环境模块

在这里，我们设计了 StroopEnvironment 类来模拟 Stroop 效应实验的环境。在这

个类中有三个变量：colors：一个包含四种颜色名称的列表，用于生成实验中的单词和墨水颜色。color_to_idx：一个字典，将颜色名称映射到它们在 colors 列表中的索引。state：当前环境的状态，由单词状态和颜色状态组成，用于外部观察。

在这个类中，我们设计了三个主要的函数：在 reset 函数中，重置环境到初始状态。随机选择一个单词和一个墨水颜色，然后根据这些选择生成环境的状态。状态是一个包含单词和颜色索引的数组，其中索引为 1 表示相应的颜色是当前的单词或墨水颜色。

在 step 函数中，模拟实验中的一步操作。接受一个动作（即用户对墨水颜色的响应），计算奖励（如果响应正确，则为 1，否则为-1），模拟反应时间，并返回新的环境状态、奖励、是否结束以及反应时间。

在_simulate_reaction_time 函数中，模拟反应时间。基于是否一致（单词颜色与墨水颜色相同）和用户的动作来计算反应时间。一致条件下的反应时间较短，不一致条件下的反应时间较长。如果用户的动作错误，反应时间会进一步减少。确保反应时间不会小于 0.2 秒。

```
class StroopEnvironment:
    def __init__(self):
        self.colors = ['RED', 'BLUE', 'GREEN', 'YELLOW']
        self.color_to_idx = {color: idx for idx, color in enumerate(self.colors)}
        self.reset()

    def reset(self):
        self.word = random.choice(self.colors)
        self.ink_color = random.choice(self.colors)
        word_state = np.zeros(len(self.colors))
        color_state = np.zeros(len(self.colors))
        word_state[self.color_to_idx[self.word]] = 1
        color_state[self.color_to_idx[self.ink_color]] = 1
        self.state = np.concatenate([word_state, color_state])
        return self.state

    def step(self, action):
        reward = 1 if action == self.color_to_idx[self.ink_color] else -1
        rt = self._simulate_reaction_time(action)
        done = True
        return self.state, reward, done, {'rt': rt}
```

```python
def step(self, action):
    reward = 1 if action == self.color_to_idx[self.ink_color] else -1
    rt = self._simulate_reaction_time(action)
    done = True
    return self.state, reward, done, {'rt': rt}

def _simulate_reaction_time(self, action):
    base_rt = 0.4
    congruent = self.word == self.ink_color

    if congruent:
        rt = base_rt + np.random.normal(0, 0.1)
    else:
        rt = base_rt + 0.2 + np.random.normal(0, 0.15)  # 不一致条件下反应时间更长

    # 错误反应通常更快
    if action != self.color_to_idx[self.ink_color]:
        rt *= 0.9

    return max(0.2, rt)  # 确保反应时间为正
```

图 3 任务环境模块

3.2.2 基础模型结构

我们选择了最基本的 DQN 网络作为基础（如图 2 所示），由于在 Stroop 实验中，任务比较简单，反应都是离散的变量，所以我们使用了 DQN 网络。DQN 网络中有三个全连接层，最后输出一个变量用于预测当前任务的颜色。

```python
class DQN(nn.Module):
    def __init__(self, state_size, action_size):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(state_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, action_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
```

图 4 DQN 网络结构

3.2.3 智能体模块实现

StroopAgent 类是一个基于深度强化学习的智能体，用于在 Stroop 效应实验环境中进行决策。该智能体使用 DQN 网络来学习最优策略。在这个类中有 12 个主要的变量：

device：智能体将使用的设备，优先使用 GPU（如果可用）。state_size：环境状态的大小。action_size：可能的动作数量。base_dir：保存模型的目录。policy_net：智能体的策略网络，用于预测动作的 Q 值。target_net：目标网络，用于稳定学习过程。optimizer：优化器，用于更新策略网络的权重。memory：经验回放缓冲区，存储智能体的经验。batch_size：训练时的批次大小。gamma：折扣因子，用于计算未来奖励的现值。epsilon：ε-greedy 策略的初始值，用于平衡探索和利用。epsilon_decay：ε-greedy 策略的衰减率。psilon_min：ε-greedy 策略的最小值。

在这个类中，我们设计了三个主要的函数：act 函数：智能体的决策方法。根据当前状态和 ε-greedy 策略选择一个动作。如果随机数大于 ε，则选择 Q 值最高的动作；否则随机选择一个动作。remember 函数：存储经验的方法。将当前状态、动作、奖励、下一个状态和是否结束的信息存储到经验回放缓冲区。train 函数：训练方法。如果经验回放缓冲区中的样本数量大于批次大小，则从缓冲区中随机抽取一批样本进行训练。计算损失并更新策略网络的权重。同时更新 ε 的值。update_target_network 函数：更新目标网络的方法。将策略网络的权重复制到目标网络，以稳定学习过程。

```
class StroopAgent:
    def __init__(self, state_size, action_size, base_dir='stroop_saved_models'):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.state_size = state_size
        self.action_size = action_size
        self.base_dir = base_dir

        if not os.path.exists(base_dir):
            os.makedirs(base_dir)

        self.policy_net = DQN(state_size, action_size).to(self.device)
        self.target_net = DQN(state_size, action_size).to(self.device)
        self.target_net.load_state_dict(self.policy_net.state_dict())

        self.optimizer = optim.Adam(self.policy_net.parameters())
        self.memory = deque(maxlen=10000)

        self.batch_size = 64
        self.gamma = 0.99
        self.epsilon = 0.9
        self.epsilon_decay = 0.995
        self.epsilon_min = 0.01
```

```
    def act(self, state):
        if random.random() > self.epsilon:
            with torch.no_grad():
                state = torch.FloatTensor(state).unsqueeze(0).to(self.device)
                q_values = self.policy_net(state)
                return q_values.argmax().item()
        return random.randrange(4)

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def train(self):
        if len(self.memory) < self.batch_size:
            return

        batch = random.sample(self.memory, self.batch_size)
        states, actions, rewards, next_states, dones = zip(*batch)

        states = torch.FloatTensor(states).to(self.device)
        actions = torch.LongTensor(actions).to(self.device)
        rewards = torch.FloatTensor(rewards).to(self.device)
        next_states = torch.FloatTensor(next_states).to(self.device)
        dones = torch.FloatTensor(dones).to(self.device)

        current_q_values = self.policy_net(states).gather(1, actions.unsqueeze(1))
        next_q_values = self.target_net(next_states).max(1)[0].detach()
        target_q_values = rewards + (1 - dones) * self.gamma * next_q_values

        loss = nn.MSELoss()(current_q_values.squeeze(), target_q_values)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        self.epsilon = max(self.epsilon_min, self.epsilon * self.epsilon_decay)

    def update_target_network(self):
        self.target_net.load_state_dict(self.policy_net.state_dict())
```

图 5 Stroop 任务智能体结构

3.3 模型评估

模型经过训练后输出模型在 Stroop 任务中的反应正确率，结果发现正确率随着模型的训练轮数而不断提升。前 400 轮的训练对模型正确率的提升效果显著，模型正确率达到 90%后保持稳定，随训练轮数的增加缓慢提升，约 600 轮次后达到 100%正确率。

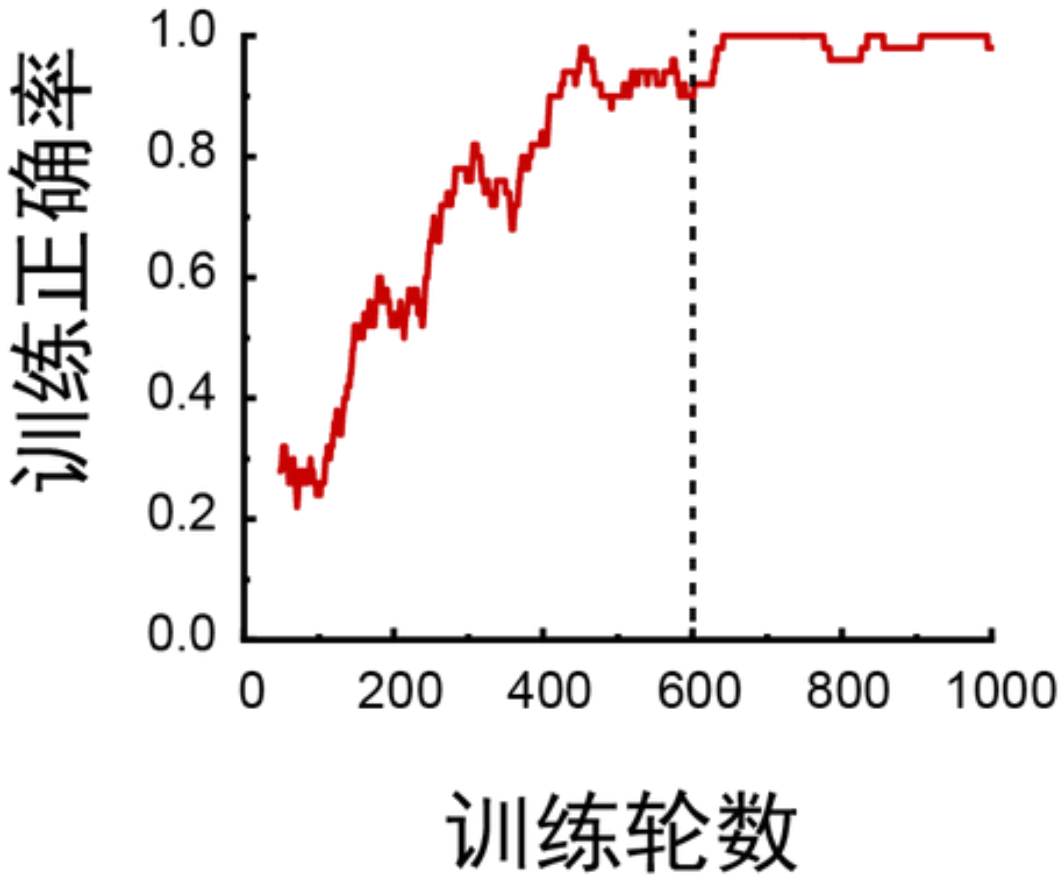

图 6 Stroop Model 学习曲线。图中 x 轴为训练轮数，y 轴为模型正确率。

模型训练后预测不同条件下 Stroop 任务的反应时。结果发现在不一致条件下，模型预测的整体反应时显著长于一致条件，表现出较强的 Stroop 效应，该效应在正确反应与错误反应中均存在。上述结果表明模型在预测 Stroop 任务反应时中模拟了人类的行为反应，在不一致条件下的颜色命名受到了语义加工的干扰，从而导致反应时延长。

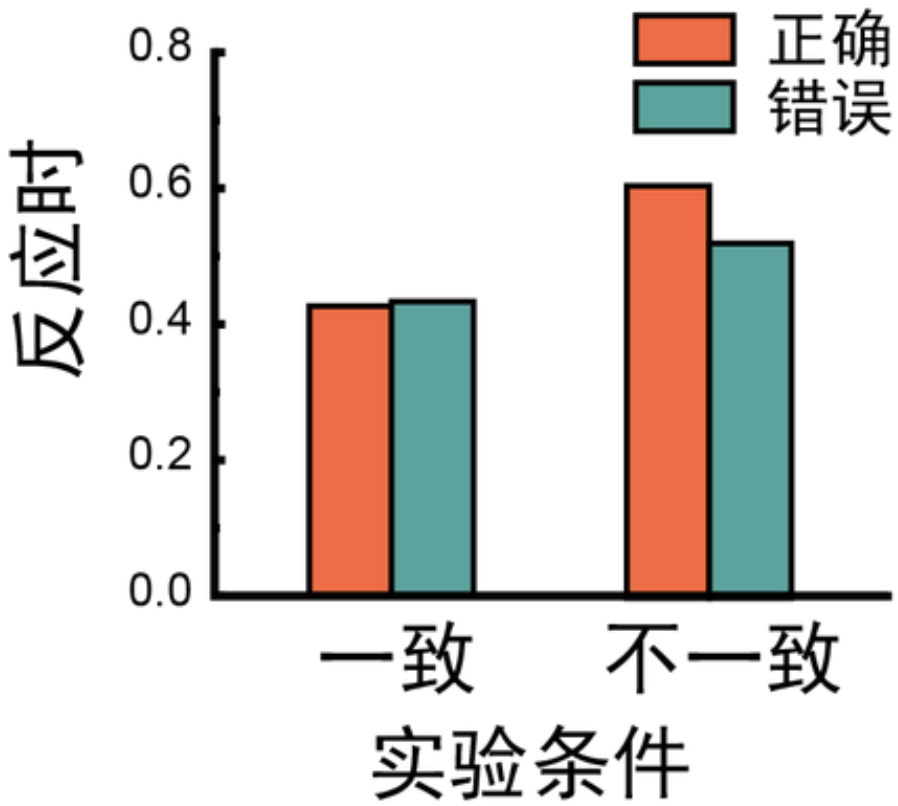

图 7 Stroop Model 反应时分布柱状图。图中 x 轴为任务条件，分别为不一致条件（Incongruent)及一致条件（Congruent）；y 轴为模型输出的反应时。

模型在不同条件下的反应时分布均呈现正态分布，但不一致条件分布的均值显著大于一致条件，再次表明模型能够成功地模拟 Stroop 效应中的人类反应模式。

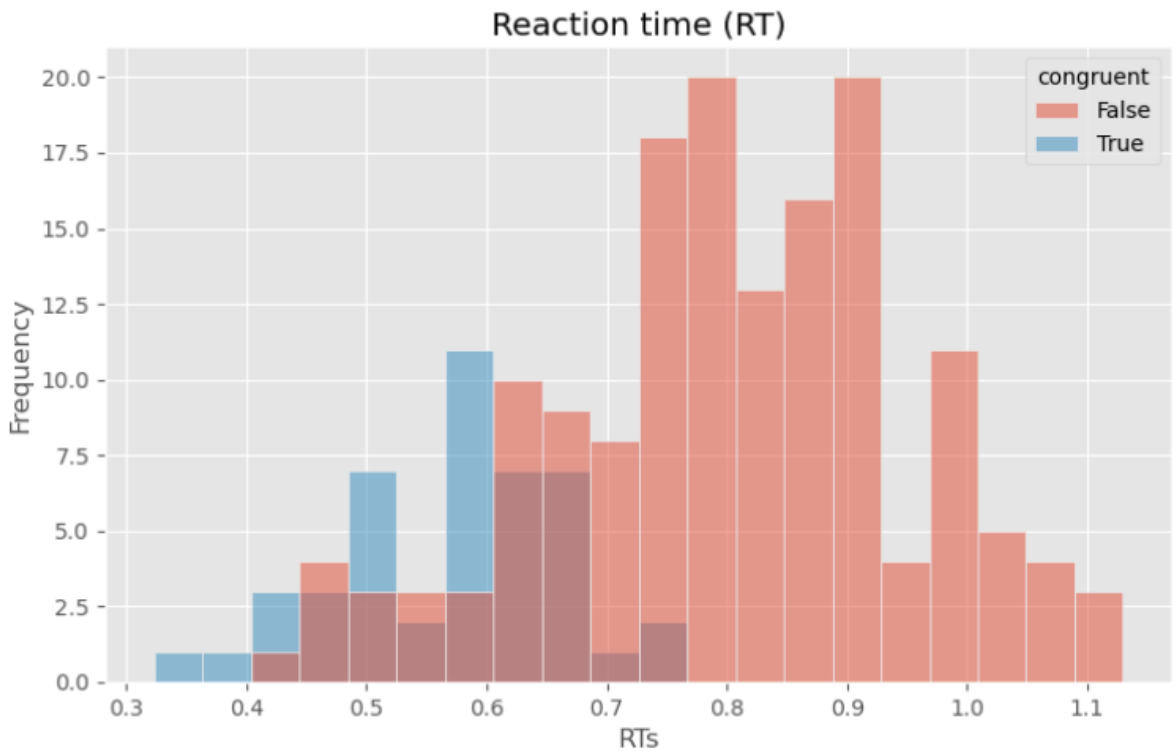

图 8 Stroop Model 反应时分布频数直方图。图中 x 轴为反应时；y 轴为频数。红色部分直方图为不一致条件，蓝色部分直方图为一致条件。

模型输出的正确率预测结果显示，在不一致条件下 Stroop 任务的反应正确率显著低于一致条件，表明模型能够成功模拟 Stroop 效应中人类的反应模式。

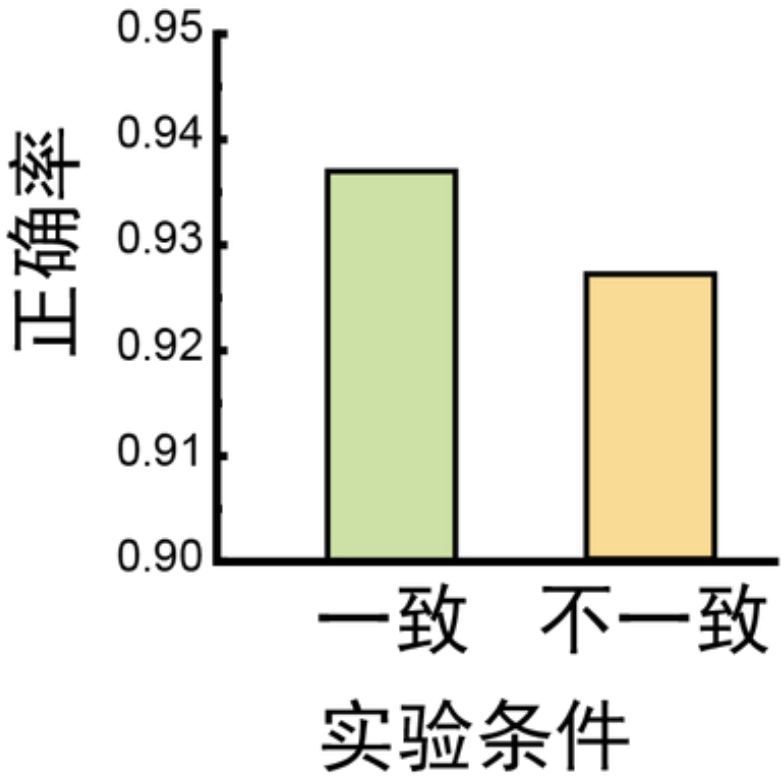

图 9 Stroop Model 正确率柱状图。图中 x 轴为实验条件，左侧为一致条件，右侧为不一致条件；y 轴为反应正确率。

3.4 评估结论

模型的损失随着学习的 epoch 逐渐降低，直到变得平稳，这说明在本研究中模型的训练轮次是足够的。模型生成的反应时模式符合经典 Stroop 效应，即在不一致的情况下反应时大于一致情况下的反应时，这说明了模型和人类的相似性。模型生成的错误率分布与人类数据相似，即在不一致情况下的错误率大于一致情况下的错误率。

本部分的结果说明强化学习训练的模型能够达到和人类类似的成绩，可以用于后续的研究。

四、运动-视觉判断任务模型

4.1 运动-视觉判断任务的介绍和理论

在视觉感知领域，经典的前馈加工模型一般将视觉看做是被动地接收自下而上信息

的过程，感觉通路中的神经元仅加工前馈的输入信号，从而形成对外部世界的表征(Keller & Mrsic-Flogel, 2018)。但主动感知领域的研究者认为，人们通过动作认识世界并改造世界，这一过程不仅涉及感觉系统与运动系统相互独立的信息处理，同时也包含二者的交互作用，而前馈加工模型则忽略了来自于运动系统的自上而下信息对于输入信号的主动建构与调节。基于此，研究者们提出输入信号可以进一步划分为“外传入信号”与“自传入信号”(Straka, Simmers, & Chagnaud, 2018)，前者是指由外部环境改变而造成的感觉输入，后者则是指由机体自身活动而造成的感觉输入。例如视网膜接收到移动光点的输入信号，但大脑需要区分光点的移动是由自身的眼动导致（自传入信号），还是光点本身在移动（外传入信号）。正确区分这两种信号对于动物的生存至关重要，例如猴子在丛林中活动时，需要分辨树叶的摩擦声是来自于自己的动作，还是来自于捕食者(Crapse & Sommer, 2008b)。但仅仅包含外传入信号的前馈加工模型无法说明大脑如何区分两种输入信号，缺乏一定的解释力。因此研究者们提出预期编码理论(Straka et al., 2018)，认为应将感知过程看做是动作过程的结果（consequence），动作系统在发出运动指令传输至效应器完成动作的同时，也会自上而下地发送传出副本（或伴随放电）信号(Ford & Mathalon, 2019)，传递至感觉系统以完成对于动作感知结果的预测。该信号中包含了相应的动作参数，感觉皮层能够在整合自上而下信息和自下而上信息的基础上实现较为准确的感知。例如在灵长类动物中，来自上丘（SC）的眼动信号能够传递至前眼场（FEF），使神经元的感受野提前转移至眼动目标位置(Rao et al., 2016)；在鸣禽中，发声运动能够抑制听觉皮层对自主鸣叫声音的反应，从而提高对外部环境音的感知；在仅有低级反射的线虫中，也存在运动抑制通路，调节感觉系统对外传入信号的反应强度(Crapse & Sommer, 2008a)。

不难发现，大脑在根据运动反馈信号调节感觉输入信号时，需要结合具体的动作参数对感知后果进行预测，这一过程一般被认为发生在初级感觉皮层，相应的运动信号也在视觉系统中得以表征，包括头动方向(Guitchounts, Masís, Wolff, & Cox, 2020)、眼动方向、肢体运动方向(Bola, Vetter, Wenger, & Amedi, 2023)等。方向作为基本的视觉特征(Jin, Dragoi, Sur, & Seung, 2005)，能够在早期的视觉处理过程中就被提取出，并且持续的方向刺激能够诱发神经元的适应性调整能力(Webster, 2015)。因此本项目将视觉皮层神经元对方向的适应性反应作为探索动作-感知交互的窗口，在预期编码理论的基础上，探讨包含动作方向的运动反馈信号是否能够调节初级视觉反应。

项目假设机体在执行某个方向的上肢运动时，运动系统会向感觉系统发送传出副本，其中包含了动作的方向信息，该方向信息随动作的重复执行反复传递至视觉皮层，能够引发视觉神经元的适应性调整能力，从而产生适应后效。

项目中分别构建两种模型进行对比，模型一将感觉系统与运动系统分离，感觉系统的预测结果不受运动系统输入的影响；模型二将感觉系统与运动系统整合，感觉系统的预测结果受到运动系统的影响，在自上而下的动作方向信息重复输入后，感觉系统会产生适应后效，使其在预测中性角度时更多偏向与动作方向相反的角度。

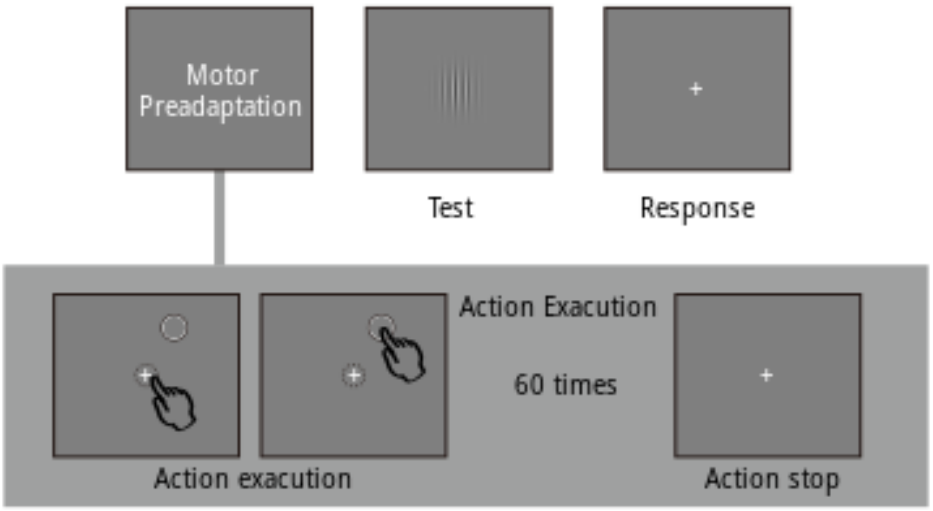

图 10 运动-视觉判断任务流程图。实验任务要求被试在完成目标方向的点触动作后再进行视觉光栅方向判断任务。要求被试首先重复 60 次相同方向的点触动作（动作方向为 80°，90°，100°），之后对视觉光栅刺激的倾斜角度进行判断（光栅倾斜角度为 88°，89°，90°，91°，92°）。

4.2 强化学习实现

在本项目中，我们设计了两个深度学习模型，即 IndependentAgent 和 CombinedAgent。这两个模型都基于相同的架构，但它们在处理输入数据的方式上有所不同。具体来说，IndependentAgent 中两个模型是平行加工的，CombinedAgent 中的两个模型具有交互，模拟了运动对于视觉的副本操作（如图所示）。

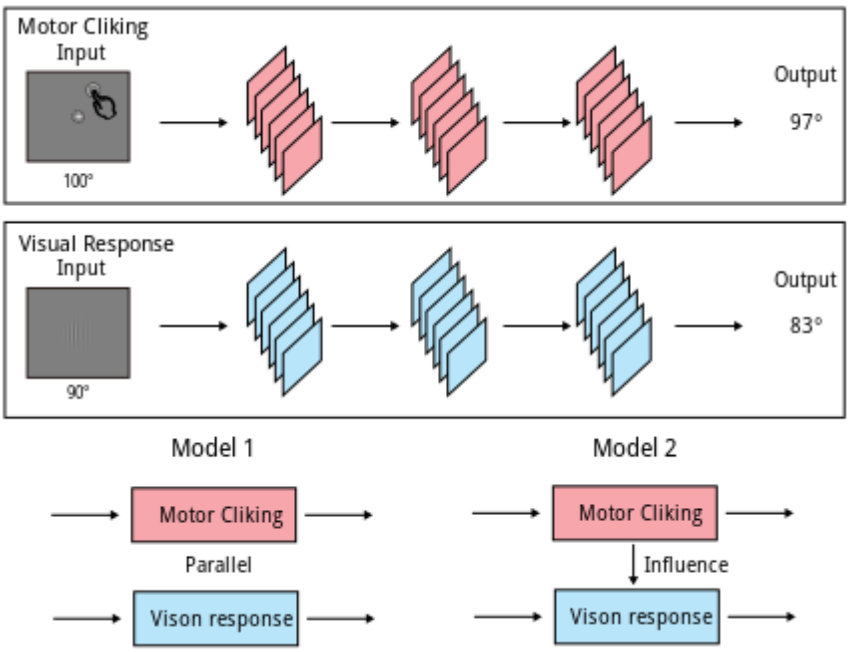

图 11 运动-视觉判断任务模型结构示意图。包括两个基本模型，即 Motor Clicking 和 Visual Response 模块，前者用于让模拟被试的点击，后者用于模拟被试对光栅朝向的感知。根据两个基本模型我们设计了两个核心模型，即 Model1（IndependentAgent）和 Model2（CombinedAgent），Model1 中两个模型是平行加工的，Model2 中的两个模型具有交互，模拟了运动对于视觉的副本操作。

4.2.1 任务环境模块

基于这个任务，我们设计了一个基于 Gym 库的环境类 ClickPerceptionEnv 类，用于模拟点击感知任务和线段估计任务。该环境旨在研究用户如何通过点击操作来感知和估计角度。

该类中有 4 个变量：target_angles: 点击任务中目标角度的数组，表示可能的目标角度，即 80°，90° 和 100°。line_angles: 感知线条角度的数组，表示可能的感知线段角度，即 88°，89°，90°，91°，92°。observation_space: 观测空间，表示环境状态的范围，这里是两个角度值。action_space: 动作空间，表示可能的点击角度和线段预测的范围，在这里点击角度可能是 0°-180° 的连续角度，线段预测角度则为左（小于 90°）或右（大于 90°）的二分迫选。

该类中有 2 个核心函数：reset 函数用于重置环境到初始状态，随机选择一个点击角度和一个线段角度，返回这两个角度作为初始状态。step 函数用于执行一步操作。接

受两个动作参数：点击角度和线段预测。计算点击任务的奖励（点击角度越接近目标角度，奖励越高），以及线段估计任务的奖励（预测方向与实际方向一致时获得奖励）。返回新的状态、奖励、是否结束等信息。

```
class ClickPerceptionEnv(gym.Env):
    metadata = {'render.modes': ['human']}

    def __init__(self):
        super(ClickPerceptionEnv, self).__init__()
        self.target_angles = np.array([80, 90, 100])
        self.line_angles = np.array([88, 89, 90, 91, 92])
        self.observation_space = spaces.Box(low=0, high=180, shape=(2,), dtype=np.float32)
        self.action_space = spaces.Box(low=0, high=180, shape=(1,), dtype=np.float32)  # 连续动作空间

    def reset(self):
        self.target_angle = np.random.choice(self.target_angles)
        self.line_angle = np.random.choice(self.line_angles)
        return np.array([self.target_angle, self.line_angle])

    def step(self, action_clicking_angle, action_line_pred):
        # 点击任务
        clicking_angle = action_clicking_angle
        clicking_reward = -abs(clicking_angle - self.target_angle)  # 越接近目标角度，奖励越高

        # 线段估计任务
        if self.line_angle > 90:
            direction = 1  # 右
        else:
            direction = 0  # 左

        if action_line_pred[0][0] < action_line_pred[0][1]:
            pred_direction = 1
        else:
            pred_direction = 0

        if pred_direction == 0 and direction == 0:
            line_reward = 10
        elif pred_direction == 1 and direction == 1:
            line_reward = 10
        else:
            line_reward = 0

        reward = clicking_reward + line_reward
        done = True
        return np.array([self.target_angle, self.line_angle]), reward, done, {}

    def render(self, mode='human', close=False):
        pass
```

图 12 任务环境模块

4.2.2 基础模型结构

实验设计了两种模型架构(ClickingModule 和 LineOrientationModule)来模拟本实验中的点击行为和线条方向判断任务。

4.2.2.1 点击模块(ClickingModule)

该类别用于模拟被试的点击行为，具有三个全连接层，输出是模拟的点击的角度。

```
class ClickingModule(nn.Module):
    def __init__(self, hidden_size=64):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(1, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 1),
            nn.Tanh()
        )

    def forward(self, target_angle):
        output = self.network(target_angle)
        clicking_angle = output
        return clicking_angle
```

图 13 点击模块

4.2.2.2 线条朝向模块(LineOrientationModule)

该类别用于模拟被试的对线条角度的预测，具有三个全连接层，输出是模拟的二分类预测（左或右）。

```python
class LineOrientationModule(nn.Module):
    def __init__(self, hidden_size=64, use_clicking=False):
        super().__init__()
        input_size = 1 if not use_clicking else 2
        self.network = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 2)
        )

    def forward(self, line_angle, clicking_angle=None):
        if clicking_angle is not None:
            x = torch.cat((line_angle, clicking_angle), dim=-1)
        else:
            x = line_angle
        return self.network(x)
```

图 14 线条朝向模块

4.2.3 智能体模块实现

4.2.3.1 IndependentAgent 类

该类别用于模拟在运动不给视觉提供输出副本情况下的模型的预测。

```python
class IndependentAgent(nn.Module):
    def __init__(self, hidden_size=64):
        super(IndependentAgent, self).__init__()
        self.clicking = ClickingModule(hidden_size)
        self.line_orientation = LineOrientationModule(hidden_size, use_clicking=False)

    def forward(self, state):
        target_angle = state[:, 0].unsqueeze(-1)
        line_angle = state[:, 1].unsqueeze(-1)
        clicking_angle = self.clicking(target_angle)
        line_pred = self.line_orientation(line_angle)
        return clicking_angle, line_pred
```

图 15 IndependentAgent 类实现

4.2.3.2 CombinedAgent 类

该类别用于模拟在运动给视觉提供输出副本情况下的模型的预测。

```python
class CombinedAgent(nn.Module):
    def __init__(self, hidden_size=64):
        super(CombinedAgent, self).__init__()
        self.clicking = ClickingModule(hidden_size)
        self.line_orientation = LineOrientationModule(hidden_size, use_clicking=True)

    def forward(self, state):
        target_angle = state[:, 0].unsqueeze(-1)
        line_angle = state[:, 1].unsqueeze(-1)
        clicking_angle = self.clicking(target_angle)
        clicking_angle = clicking_angle.squeeze(-1)

        if clicking_angle.dim() < line_angle.dim():
            clicking_angle = clicking_angle.unsqueeze(0)
        elif clicking_angle.dim() > line_angle.dim():
            line_angle = line_angle.unsqueeze(0)

        line_pred = self.line_orientation(line_angle, clicking_angle)
        return clicking_angle, line_pred
```

图 16 CombineAgent 类实现

4.2.4 模型训练

该函数使用了强化学习中的策略梯度方法，并通过 Adam 优化器来更新智能体的参数。

```python
def train_agent(env, agent, n_episodes=1000):
    optimizer = optim.Adam(agent.parameters(), lr=0.001)
    clicking_losses = []
    line_losses = []

    for episode in range(n_episodes):
        state = torch.tensor(env.reset(), dtype=torch.float32).unsqueeze(0)
        done = False
        episode_clicking_loss = 0
        episode_line_loss = 0

        while not done:
            action_clicking_angle, line_pred = agent(state)
            action_clicking_angle = action_clicking_angle.squeeze().detach()
            next_state, reward, done, _ = env.step(action_clicking_angle.numpy(), line_pred.detach().numpy())
            next_state = torch.tensor(next_state, dtype=torch.float32).unsqueeze(0)
            reward = torch.tensor(reward, dtype=torch.float32).unsqueeze(0)

            clicking_loss = -reward.mean()
            episode_clicking_loss += clicking_loss.item()

            line_loss = nn.CrossEntropyLoss()(line_pred, torch.tensor([1 if r.item() > 0 else 0 for r in reward]))
            episode_line_loss += line_loss.item()

            state = next_state
```

```
        clicking_losses.append(episode_clicking_loss / (episode + 1))
        line_losses.append(episode_line_loss / (episode + 1))

        optimizer.zero_grad()

        total_loss = torch.tensor(clicking_losses[-1] + line_losses[-1], requires_grad=True)
        total_loss.backward()
        optimizer.step()

    return clicking_losses, line_losses
```

图 17 模型训练实现

4.2.5 模型评估

运动-视觉模型 1 经过训练后输出模型在运动-视觉任务中的损失，结果发现运动任务及视觉任务的损失均随着模型的训练轮数而不断下降。前 10 轮的训练对模型损失的降低效果显著，模型损失达到 10%后保持稳定，随训练轮数的增加缓慢下降，约 20 轮次后达到 0%损失。

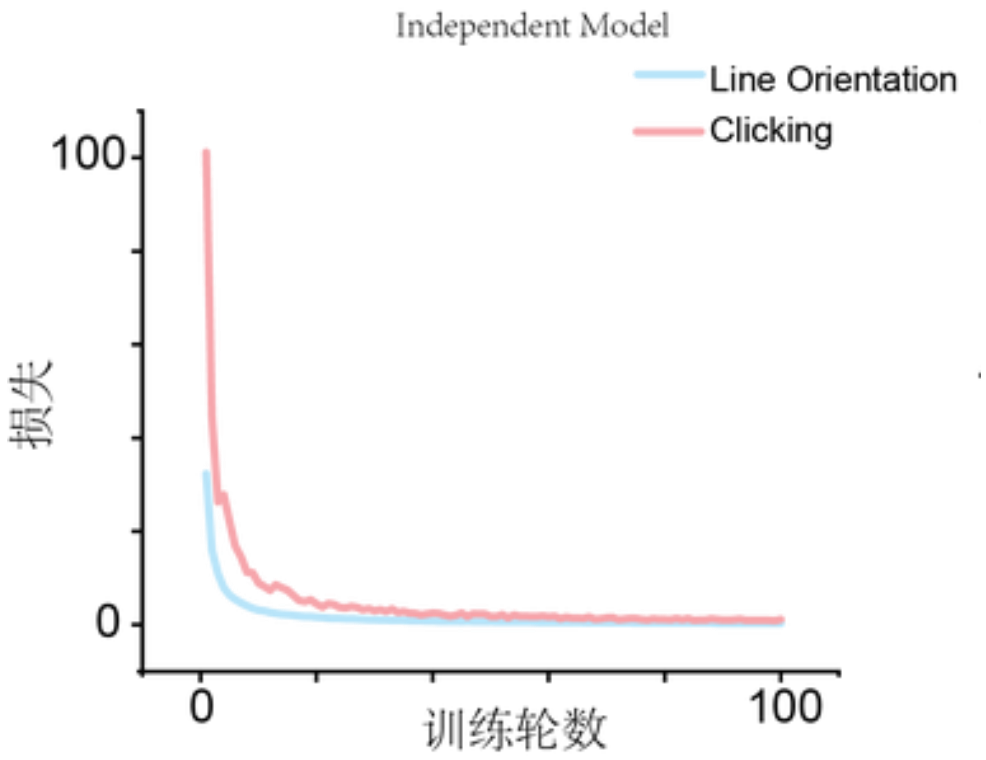

图 18 运动-视觉 Model 1 损失曲线。图中 x 轴代表训练轮次，y 轴代表模型损失。

同样地，运动-视觉模型 2 经过训练后输出模型在运动-视觉任务中的损失，结果

发现在运动任务和视觉任务中模型损失均随着模型的训练轮数而不断下降。前 10 轮的训练对运动模型损失的降低效果显著，模型损失达到 10%后保持稳定，随训练轮数的增加缓慢下降，约 20 轮次后达到 0%损失。而视觉模型的损失在模型训练初期便保持在较低水平，训练轮次给其损失带来的降低效果并不显著。

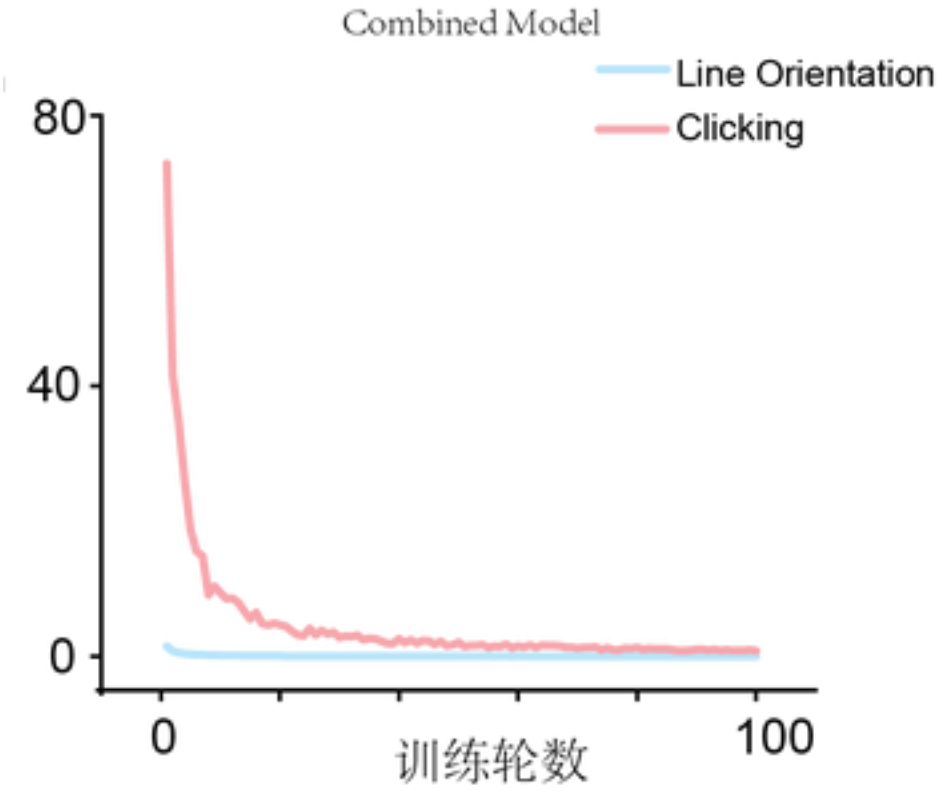

图 19 运动-视觉 Model 2 损失曲线。图中 x 轴代表训练轮次，y 轴代表模型损失。

考察运动-视觉模型 1 在视觉光栅角度判断任务中的输出正确率，结果发现在不同的动作角度条件下，模型 1 对于光栅朝向的输出正确率高于机会水平。但动作角度为 100° 时，即动作角度相对于垂直方向偏向左侧时，视觉光栅输出的正确率显著下降。

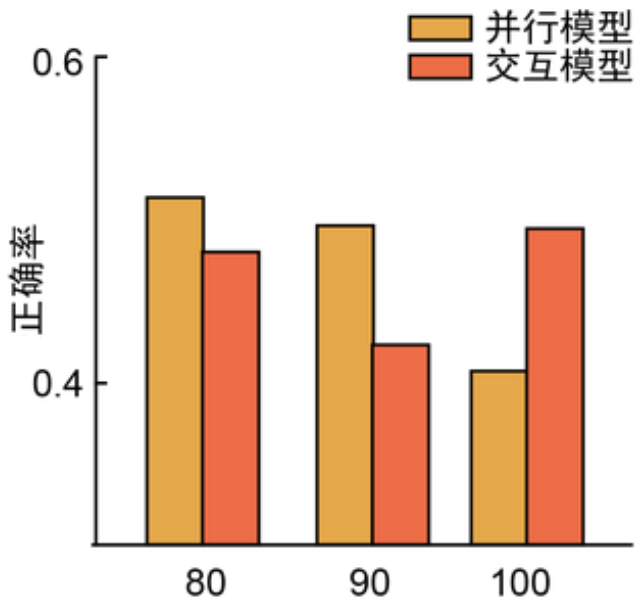

图 20 运动-视觉 Model 1 视觉任务输出正确率。图中 x 轴代表运动任务的点击角度，y 轴代表视觉任务中模型输出的正确率。在计算视觉任务正确率时，剔除垂直角度（90°）条件，仅计算余下条件的输出正确率。

考察运动-视觉模型 2 在视觉光栅角度判断任务中的输出正确率，结果发现在不同的动作角度条件下，模型 2 对于光栅朝向的输出正确率高于机会水平。但动作角度为 90°时，即动作角度为垂直角度时，视觉光栅输出的正确率略低于其余条件。

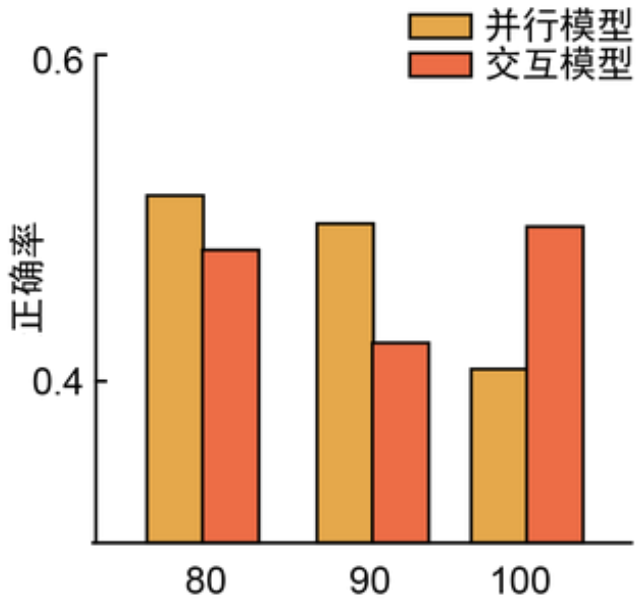

图 21 运动-视觉 Model 2 视觉任务输出正确率。图中 x 轴代表运动任务的点击角度，y 轴代表视觉任务中模型输出的正确率。在计算视觉任务正确率时，剔除垂直角度（90°）条件，仅计算余下条件的输出正确率。

进一步考察模型 1 在视觉任务中，视觉刺激光栅角度为 90° 的条件下模型的反应输出。根据模型 1 假设，运动过程与视觉过程相互平行独立，运动的输出并不会影响视觉任务表现，因此在不同的动作条件下，模型对于 90° 光栅的反应应该保持“左”反应与“右”反应概率的持平。但模型的输出表现出强烈的右偏趋势，在所有条件中模型对于垂直光栅的反应均为右。该结果不符合模型假设。

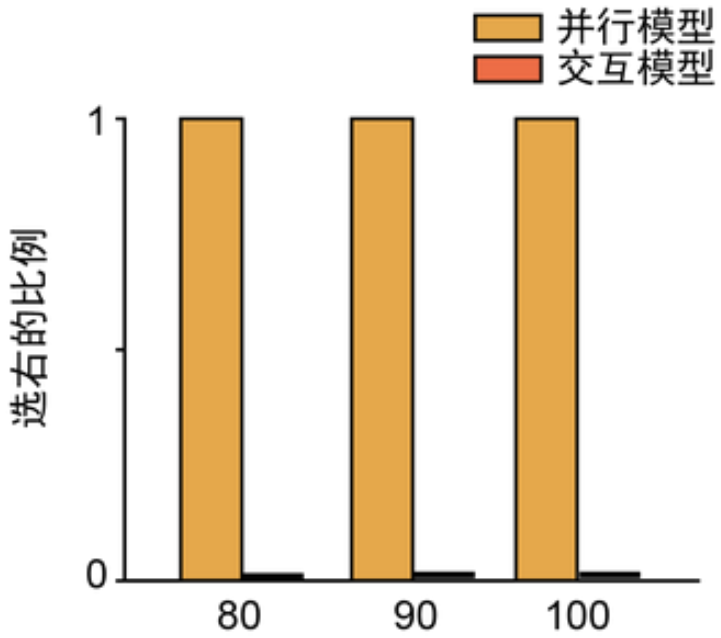

图 22 运动-视觉 Model 1 视觉任务输出反应概率。图中 x 轴代表运动任务的点击角度，y 轴代表视觉任务中模型在光栅朝向 90° 条件下输出不同反应的概率。红色直方图代表模型输出“光栅方向为左”的反应概率；绿色直方图代表模型输出“光栅方向为右”的反应概率。

考察模型 2 在视觉任务中，视觉刺激光栅角度为 90° 的条件下模型的反应输出。根据模型 2 假设，运动过程与视觉过程存在交互作用，运动的输出会作为自上而下的信号对视觉反应产生影响，因此在不同的动作条件下，模型对于 90° 光栅的反应应该受到动作角度的影响：在动作角度为 100° 时（偏左），模型对于垂直光栅的判断会由于适应后效而表现出右偏的倾向；在动作角度为 80° 时（偏右），模型对于垂直光栅的判断会由于适应后效而表现出左偏的倾向。但模型 2 的输出仍与模型 1 相同，表现出强烈的右偏趋势，在所有条件中模型对于垂直光栅的反应均为右。该结果不符合模型假设。

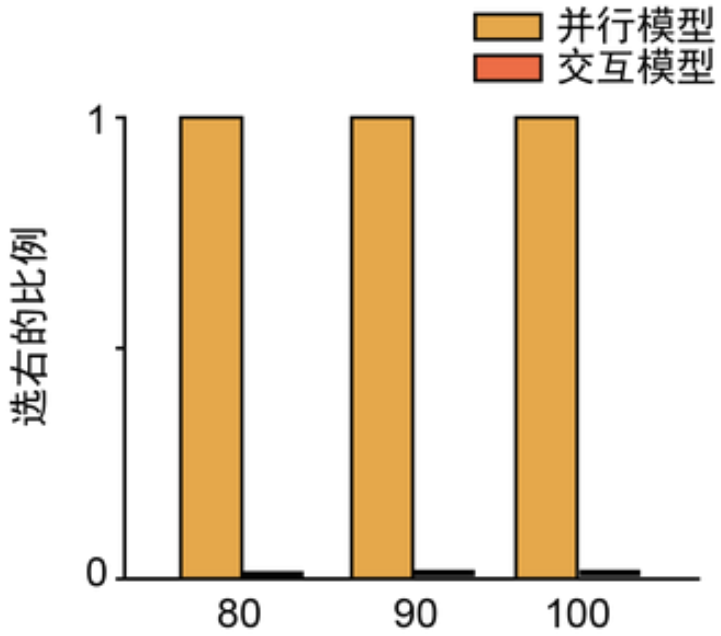

图 23 运动-视觉 Model 2 视觉任务输出反应概率。图中 x 轴代表运动任务的点击角度，y 轴代表视觉任务中模型在光栅朝向 90° 条件下输出不同反应的概率。红色直方图代表模型输出“光栅方向为左”的反应概率；绿色直方图代表模型输出“光栅方向为右”的反应概率。

4.3 对比分析：监督学习的模型结构

在强化学习的结果中，对人类数据的真实拟合效果并不算好，因此我们又将这两种模型使用监督学习的方法进行训练，想要通过这种方法观察在该任务中是否有监督学习的训练效果比强化学习的训练效果更好。

4.3.1 基础模型

在这里，我们的基础模型（ClickingModule 和 LineOrientationModule 类）和强化学习中的基础模型大致相同。

```python
class ClickingModule(nn.Module):
    def __init__(self, hidden_size=64):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(1, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 1),
            nn.Tanh()
        )

    def forward(self, target_angle):
        output = self.network(target_angle.unsqueeze(-1))
        clicking_angle = output * 180
        return clicking_angle
```

图 24 点击模拟模块实现

```python
class LineOrientationModule(nn.Module):
    def __init__(self, hidden_size=64, use_clicking=False):
        super().__init__()
        input_size = 1 if not use_clicking else 2  # Line angle only or line angle + clicking angle
        self.network = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 2)  # Binary classification (left/right)
        )

    def forward(self, line_angle, clicking_angle=None):
        if clicking_angle is not None:
            if line_angle.dim() == 1:
                line_angle = line_angle.unsqueeze(-1)
            if clicking_angle.dim() == 1:
                clicking_angle = clicking_angle.unsqueeze(-1)
            if line_angle.shape != clicking_angle.shape:
                clicking_angle = clicking_angle.expand(line_angle.shape)
            x = torch.cat([line_angle, clicking_angle], dim=-1)
        else:
            x = line_angle.unsqueeze(-1)
        return self.network(x)
```

图 25 线条朝向模块实现

4.3.2 核心模型

在这里，我们的核心模型有两个，即 Model1 和 Model2，Model1 中两个模型是平行加工的，Model2 中的两个模型具有交互，模拟了运动对于视觉的副本操作。

```python
class Model1(nn.Module):
    def __init__(self, hidden_size=64):
        super().__init__()
        self.clicking = ClickingModule(hidden_size)
        self.line_orientation = LineOrientationModule(hidden_size, use_clicking=False)

    def forward(self, target_angle, line_angle):
        clicking_angle = self.clicking(target_angle)
        line_pred = self.line_orientation(line_angle)
        return clicking_angle, line_pred
```

图 26 Independent（Model1）类模拟实现

```python
class Model2(nn.Module):
    def __init__(self, hidden_size=64):
        super().__init__()
        self.clicking = ClickingModule(hidden_size)
        self.line_orientation = LineOrientationModule(hidden_size, use_clicking=True)

    def forward(self, target_angle, line_angle):
        clicking_angle = self.clicking(target_angle)
        if clicking_angle.dim() > line_angle.dim():
            clicking_angle = clicking_angle.squeeze(-1)
        elif clicking_angle.dim() < line_angle.dim():
            line_angle = line_angle.squeeze(-1)
        line_pred = self.line_orientation(line_angle, clicking_angle)
        return clicking_angle, line_pred
```

图 27 Combine（Mmodel2）类模拟实现

4.3.3 任务环境

在有监督学习中的任务和强化学习的任务大致相等。

```python
class ExperimentEnvironment:
    def __init__(self, n_trials=100):
        self.n_trials = n_trials
        self.target_angles = [80, 90, 100]
        self.line_angles = [88, 89, 90, 91, 92]

    def generate_trial(self):
        target_angle = np.random.choice(self.target_angles)
        line_angle = np.random.choice(self.line_angles)
        return target_angle, line_angle

    def generate_batch(self, batch_size):
        target_angles = []
        line_angles = []
        for _ in range(batch_size):
            target, line = self.generate_trial()
            target_angles.append(target)
            line_angles.append(line)
        return torch.tensor(target_angles, dtype=torch.float32), torch.tensor(line_angles, dtype=torch.float32)
```

图 28 任务环境模块实现

4.3.4 训练模块

在有监督学习中，模型训练使用了均方误差损失（MSELoss）和交叉熵损失（CrossEntropyLoss）来分别计算点击角度和线条方向的预测误差。通过 Adam 优化器来更新模型的参数。

```
def train_model(model, env, n_epochs=100, batch_size=32):
    optimizer = optim.Adam(model.parameters(), lr=0.0001)
    clicking_loss_fn = nn.MSELoss()
    line_loss_fn = nn.CrossEntropyLoss()

    clicking_losses = []
    line_losses = []

    # scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)

    for epoch in range(n_epochs):
        target_angles, line_angles = env.generate_batch(batch_size)

        human_clicking = target_angles + torch.randn_like(target_angles) * 5  # 5 degree std
        # human_line_pred = (line_angles < 90).long()  # 1 for right, 0 for left
        human_line_pred = torch.where(line_angles < 90, torch.tensor(1),
                                      torch.where(line_angles > 90, torch.tensor(0), torch.tensor(-1)))

        clicking_angle, line_pred = model(target_angles, line_angles)

        clicking_loss = clicking_loss_fn(clicking_angle, human_clicking)
        mask = line_angles != 90
        # line_loss = line_loss_fn(line_pred, human_line_pred)
        line_loss = line_loss_fn(line_pred[mask], human_line_pred[mask].long())

        total_loss = clicking_loss + line_loss
```

```
        optimizer.zero_grad()
        total_loss.backward()
        optimizer.step()

        clicking_losses.append(clicking_loss.item())
        line_losses.append(line_loss.item())

        # scheduler.step()

        if (epoch + 1) % 10 == 0:
            print(f"Epoch {epoch+1}, Clicking Loss: {clicking_loss.item():.4f}, Line Loss: {line_loss.item():.4f}")

    return clicking_losses, line_losses
```

图 29 训练模块实现

4.3.5 模型评估

考察监督学习的两种模型（运动-视觉独立模型、运动-视觉交互模型）在视觉光栅角度判断任务中的输出正确率，结果发现在不同的动作角度条件下，两种模型对于光栅朝向的输出正确率均低于机会水平。仅有运动-视觉交互模型在动作角度为 100° 时，即动作角度相对于垂直方向偏向左侧时，视觉光栅输出的正确率高于机会水平。

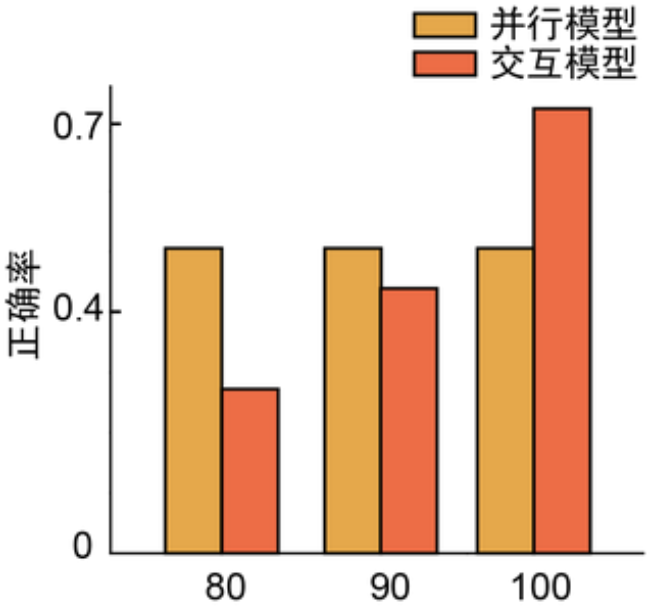

图 30 监督学习模型视觉任务输出正确率。图中 x 轴代表运动任务的点击角度，y 轴代表视觉任务中模型输出的正确率。蓝色直方图代表运动-视觉独立模型，橙色直方图代表运动-视觉交互模型。在计算视觉任务正确率时，计算所有光栅角度条件的输出正确率。

进一步考察监督学习模型在视觉任务中，视觉刺激光栅角度为 90° 的条件下模型的反应输出。根据运动-视觉独立模型预测，模型在输出视觉光栅的方向时不受运动输入影响，结果应该表现为在光栅为 90° 时模型输出”光栅方向为左“和”光栅方向为右“的比例相当。而根据运动-视觉交互模型预测，模型在输出视觉光栅的方向时受到运动输入的影响，结果表现为在光栅为 90° 时模型做出的反应受到了动作角度的调节，导致其反应偏离动作角度。但监督学习的运动-视觉独立模型表现出强烈的右偏趋势，运动-视觉交互模型表现出强烈的左偏趋势。两种模型的结果均不符合模型假设。

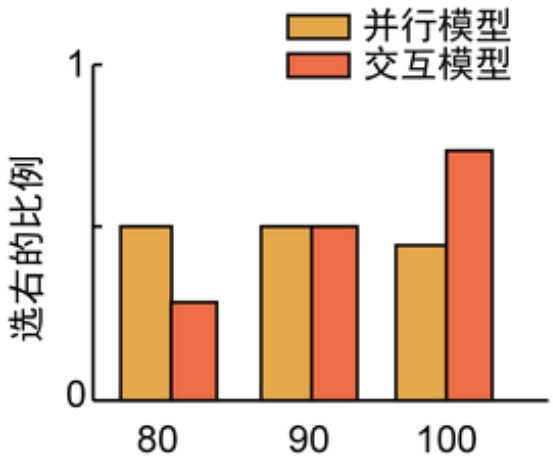

图 31 监督模型视觉任务输出反应概率。图中 x 轴代模型类型，左侧为运动-视觉独立模型，右侧为运动-视觉交互模型。y 轴代表视觉任务中模型在光栅朝向 90° 条件下输出”光栅方向为左“的反应概率。

4.4 评估结论

在三个方向上模型对于点击的模拟精度都很高，视觉的判断在光栅朝向为 90° 的情况下两种方法训练出来的模型提供的数据呈机会水平，均无法证明两种理论假设。

五、结果分析与总结

5.1 Stroop 任务

强化学习模型成功复制了人类的 Stroop 效应，模型在处理不一致（即文字内容与颜色命名相冲突）的 Stroop 任务时，展现出了与人类相似的反应时间延迟。这表明模型能够识别并处理任务中的冲突信息，类似于人类在面对认知冲突时的心理状态。
模型产生的数据与人类行为模式一致，模型产生的数据在统计上与人类的行为数据高度一致。这包括了反应时间的分布、错误率的分布以及在多次尝试后的学习曲线。这种一致性表明模型不仅能够模拟人类在 Stroop 任务中的行为表现，还能够捕捉到人类在处理此类任务时的潜在认知机制。

5.2 动作-视觉判断任务

在三个方向上模型对于点击的模拟精度都很高，视觉的判断在光栅朝向为 90° 的

情况下两种方法训练出来的模型提供的数据呈机会水平，均无法证明两种理论假设。结果说明无论是有监督模型还是强化模型，对于模拟运动-视觉的交互任务表现均不良好。

六、创新点（亮点）介绍

6.1 方法论创新

1.首次将强化学习应用于 Stroop 任务的行为模拟

传统的 Stroop 任务研究通常采用行为数据和机器学习模型进行分析。本项目创新性地引入强化学习模型，模拟人类在 Stroop 任务中的认知过程，以期在自适应调节和反应决策中呈现出与人类类似的学习和控制模式。

2.创新性设计空间定向-视觉判断任务范式

在经典 Stroop 任务的基础上，本项目设计了全新的空间定向-视觉判断任务，通过空间定向动作与视觉判断任务的结合，探索动作对视觉加工的调制效应，为动作-视觉交互机制提供新的研究方法。

3.开发了两种对比模型验证理论假设

项目中分别开发了独立与整合的对比模型，以模拟不同的感知和动作整合方式。通过对比模型在不同任务条件下的表现，进一步验证了动作系统对视觉判断的调制作用，并为理论假设提供了计算级证据。

6.2 理论创新

1.提出并验证了动作-视觉整合的计算模型

本项目建立了一个动作-视觉整合的计算模型，以模拟动作对视觉信息处理的整合过程。这一模型为动作与感知系统的互动提供了新的理论视角，推动了对认知整合机制的理解。

2.揭示了空间运动对视觉加工的调制作用

项目通过实验证实，空间运动信息能够显著影响视觉系统的加工方式，揭示了动作参数在感知过程中发挥的调节作用，为理解感知与运动系统的交互机制提供了理论依据。

6.3 技术创新

1.实现了灵活的概率分布模拟人类行为

本项目通过灵活的概率分布设置，使模型能够准确模拟人类在任务中的行为反应，并适应多种任务条件。这一设计提升了模型的泛化能力，使其更贴近真实的心理学实验数据。

2.设计了可扩展的模型架构，便于未来研究

项目模型架构设计灵活且具备模块化特点，能够根据不同的任务需求轻松调整参数和网络结构，为未来深入研究其他心理学实验范式奠定了技术基础。

6.4 应用创新

1.开发了通用的心理学实验模拟框架

本项目开发的实验模拟框架不仅适用于 Stroop 任务，还可以轻松拓展至其他心理学范式。该框架具备良好的通用性和复用性，为研究者提供了多功能的模拟环境。

2.提供了完整的数据分析和可视化工具

在实验数据分析方面，项目集成了多样化的分析和可视化工具，使研究者能够更直观地观测实验结果，为心理学实验数据的分析和解释提供了高效的支持。

七、未来展望

为了使模型更加全面和真实，未来的建模里可以引入更多的认知过程模拟。这包括但不限于：

1. 记忆过程：模拟人类的记忆形成、存储和检索过程，以更好地理解信息如何在大脑中被处理。
2. 注意力分配：研究如何在模型中实现对不同信息源的注意力分配，以模拟人类在多任务环境中的注意力管理。

除此之外，未来的建模还可以增加多模态数据的处理能力，包括：

1. 开发多模态输入处理机制：使模型能够同时处理来自不同感官通道的

信息，并学习这些信息之间的关联。

2. 增强模型的泛化能力：通过多模态数据训练，提高模型在面对不同类型数据时的适应性和泛化能力。

参考文献

Botvinick, M. M., Braver, T. S., Barch, D. M., Carter, C. S., & Cohen, J. D. (2001). Conflict monitoring and cognitive control. *Psychological Review*, *108*(3), 624–652. https://doi.org/10.1037/0033-295X.108.3.624

Dehaene, S., & Changeux, J.-P. (1989). A Simple Model of Prefrontal Cortex Function in Delayed-Response Tasks. *Journal of Cognitive Neuroscience*, *1*(3), 244–261. https://doi.org/10.1162/jocn.1989.1.3.244

Logan, G. D. (1980). Attention and automaticity in Stroop and priming tasks: Theory and data. *Cognitive Psychology*, *12*(4), 523–553. https://doi.org/10.1016/0010-0285(80)90019-5

MacLeod, C. M. (1991). Half a century of research on the Stroop effect: An integrative review. *Psychological Bulletin*, *109*(2), 163–203. https://doi.org/10.1037/0033-2909.109.2.163

Solso, R. L. (1975). *Information Processing and Cognition: The Loyola Symposium* (1st ed.). Routledge. https://doi.org/10.4324/9781032722450

Stroop, J. R. (1935). Studies of interference in serial verbal reactions. *Journal of Experimental Psychology*, *18*(6), 643–662. https://doi.org/10.1037/h0054651

Bola, L., Vetter, P., Wenger, M., & Amedi, A. (2023). Decoding Reach Direction in Early "Visual" Cortex of Congenitally Blind Individuals. *Journal of Neuroscience, 43*(46), 7868-7878. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/37783506. doi:10.1523/JNEUROSCI.0376-23.2023

Crapse, T. B., & Sommer, M. A. (2008a). Corollary discharge across the animal kingdom. *Nat Rev Neurosci, 9*(8), 587-600. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/18641666. doi:10.1038/nrn2457

Crapse, T. B., & Sommer, M. A. (2008b). Corollary discharge circuits in the primate brain. *Curr Opin Neurobiol, 18*(6), 552-557. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/18848626. doi:10.1016/j.conb.2008.09.017

Ford, J. M., & Mathalon, D. H. (2019). Efference Copy, Corollary Discharge, Predictive Coding, and Psychosis. *Biol Psychiatry Cogn Neurosci Neuroimaging, 4*(9), 764-767. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/31495399. doi:10.1016/j.bpsc.2019.07.005

Guitchounts, G., Masís, J., Wolff, S. B. E., & Cox, D. (2020). Encoding of 3D Head Orienting Movements in the Primary Visual Cortex. *Neuron, 108*(3), 512-525.e514. doi:10.1016/j.neuron.2020.07.014

Jin, D. Z., Dragoi, V., Sur, M., & Seung, H. S. (2005). Tilt aftereffect and adaptation-induced changes in orientation tuning in visual cortex. *Journal Of Neurophysiology, 94*(6), 4038-4050. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/16135549. doi:10.1152/jn.00571.2004

Keller, G. B., & Mrsic-Flogel, T. D. (2018). Predictive Processing: A Canonical Cortical Computation. *Neuron, 100*(2), 424-435. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/30359606. doi:10.1016/j.neuron.2018.10.003

Rao, H. M., San Juan, J., Shen, F. Y., Villa, J. E., Rafie, K. S., & Sommer, M. A. (2016). Neural Network Evidence for the Coupling of Presaccadic Visual Remapping to Predictive Eye Position Updating. *Front Comput Neurosci, 10*, 52. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/27313528. doi:10.3389/fncom.2016.00052

Straka, H., Simmers, J., & Chagnaud, B. P. (2018). A New Perspective on Predictive Motor Signaling. *Current Biology, 28*(5), R232-R243. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/29510116. doi:10.1016/j.cub.2018.01.033

Webster, M. A. (2015). Visual Adaptation. *Annual Review Of Vision Science, 1*, 547-567. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/26858985. doi:10.1146/annurev-vision-082114-035509