**Detailed Network Architectures**

**详细网络架构**

Here's a breakdown of each refactored model's architecture, assuming typical hyperparameters often found in the original repository's run scripts (e.g., conv\_filters=[[6, [3, 3], 1]], fcnet\_hiddens=[32, 32], lstm\_hidden\_size=128, activation="relu"). *Note: These sizes are configurable.*

以下是各重构模型架构的详细解析，假设采用原代码库运行脚本中常见的典型超参数（例如 conv\_filters=[[6, [3, 3], 1]] 、 fcnet\_hiddens=[32, 32] 、 lstm\_hidden\_size=128 、 activation="relu" ）。注意：这些规模是可配置的。

Common Preprocessing:

All models utilizing visual input (curr\_obs) start with:通用预处理步骤：

所有采用视觉输入（ curr\_obs ）的模型均以以下步骤开始：

1. **Permute:** Changes input tensor shape from (Batch, Height, Width, Channels) to (Batch, Channels, Height, Width).  
   置换：将输入张量的形状从（批次，高度，宽度，通道）更改为（批次，通道，高度，宽度）。
2. **Normalize:** Converts uint8 pixel values (0-255) to float32 (0.0-1.0).  
   归一化：将 uint8 像素值（0-255）转换为 float32（0.0-1.0）。

**a) BaselineModel**

* **Input:** Observation dictionary containing curr\_obs (e.g., shape (B, 15, 15, 3)).  
  输入：包含 curr\_obs 的观测字典（例如，形状 (B, 15, 15, 3) ）。
* **Encoder:  编码器：**
* Preprocessing (Permute, Normalize).  
  预处理（置换、归一化）。
* **Convolutional Layers (build\_conv\_layers)**:   
  卷积层（ build\_conv\_layers ）：
* Takes preprocessed curr\_obs (e.g., (B, 3, 15, 15)).  
  接收预处理后的 curr\_obs （例如 (B, 3, 15, 15) ）。
* Example: One Conv2d layer (specified by conv\_filters, e.g., out\_channels=6, kernel\_size=3, stride=1, padding='valid'). Uses ReLU activation by default. Initialized with Kaiming Normal.  
  示例：一个 Conv2d 层（由 conv\_filters 指定，例如 out\_channels=6 、 kernel\_size=3 、 stride=1 、 padding='valid' ）。默认使用 ReLU 激活函数，采用 Kaiming Normal 初始化。
* Output is flattened.  输出被展平。
* **Fully Connected Layers (build\_fc\_layers)**:   
  全连接层（ build\_fc\_layers ）：
* Takes flattened output from Conv layers.  
  接收来自卷积层的扁平化输出。
* Example: Two Linear layers (specified by fcnet\_hiddens, e.g., [32, 32]). Uses ReLU activation by default. Initialized with PyTorch default (Kaiming Uniform).  
  示例：两个线性层（由 fcnet\_hiddens 指定，例如 [32, 32] ）。默认使用 ReLU 激活函数，初始化采用 PyTorch 默认的 Kaiming 均匀分布方法。
* Output is the encoded feature vector (e.g., size 32).  
  输出为编码后的特征向量（例如大小为 32 ）。
* **Recurrent Core (ActorCriticLSTM)**:   
  循环核心（ ActorCriticLSTM ）：
* **LSTM Cell:** Takes the encoded feature vector from the FC layers as input at each timestep. Hidden/cell state size is lstm\_hidden\_size (e.g., 128). Initialized with Orthogonal weights.  
  LSTM 单元：在每个时间步将全连接层编码的特征向量作为输入。隐藏/细胞状态大小为 lstm\_hidden\_size （例如 128）。权重采用正交初始化。
* **Actor Head:** A Linear layer taking LSTM hidden state as input. Outputs policy logits (size num\_actions, e.g., 9). Initialized orthogonally with small gain (e.g., 0.01).  
  演员头部：一个线性层，以 LSTM 隐藏状态为输入。输出策略对数（大小 num\_actions ，例如 9）。采用小增益（如 0.01）正交初始化。
* **Critic Head:** A Linear layer taking LSTM hidden state as input. Outputs a single value estimate. Initialized orthogonally with gain 1.0.  
  批评头（Critic Head）：一个以 LSTM 隐藏状态为输入的线性层，输出单一价值估计。采用正交初始化，增益设为 1.0。
* **Output:** Policy logits, value estimate, and the next LSTM hidden/cell state.  
  策略逻辑、价值估计及下一个 LSTM 隐藏/细胞状态。

**b) MOAModel**

* **Input:** Observation dictionary (curr\_obs), previous actions of *all* agents (prev\_actions), and recurrent state (ActorCritic LSTM state + MOA LSTM state).  
  观察字典（ curr\_obs ）、所有智能体的先前动作（ prev\_actions ）以及循环状态（ActorCritic LSTM 状态 + MOA LSTM 状态）。
* **Encoder:  编码器：**
* Preprocessing (Permute, Normalize) on curr\_obs.  
  预处理（置换、归一化）在 curr\_obs 上。
* **Convolutional Layers (build\_conv\_layers)**: *Shared* Conv layers, same structure as Baseline (e.g., one Conv2d layer, ReLU activation, Kaiming Normal init). Output is flattened.  
  卷积层（ build\_conv\_layers ）：共享卷积层，结构与基线相同（例如，一个 Conv2d 层、ReLU 激活函数、采用 Kaiming 正态初始化）。输出被展平。
* **Split Fully Connected Layers (build\_fc\_layers)**:   
  分割全连接层（ build\_fc\_layers ）：
* *Policy Path*: Takes flattened Conv output. Separate FC layers (e.g., [32, 32], ReLU activation, default init) leading to the ActorCriticLSTM.  
  策略路径：接收展平的卷积输出。独立的 FC 层（如 [32, 32] 、ReLU 激活、默认初始化）连接至 ActorCriticLSTM。
* *MOA Path*: Takes flattened Conv output. Separate FC layers (e.g., [32, 32], ReLU activation, default init) leading to the MoaLSTM.  
  MOA 路径：接收展平的卷积输出。独立的 FC 层（如 [32, 32] 、ReLU 激活、默认初始化）连接至 MoaLSTM。
* **Recurrent Cores:  循环核心：**
* **ActorCriticLSTM**: Same as in Baseline, takes features from the *Policy Path* FC layers. Outputs policy logits and value estimate. Orthogonal init.  
  ActorCriticLSTM ：与基线相同，从策略路径全连接层提取特征。输出策略逻辑值和价值估计。采用正交初始化。
* **MoaLSTM**:
* Takes features from the *MOA Path* FC layers **concatenated with** one-hot encoded prev\_actions (actions from t-1) as input. Hidden/cell state size lstm\_hidden\_size (e.g., 128). Orthogonal init.  
  输入为 MOA 路径全连接层的特征与 t-1 时刻动作 prev\_actions 的一热编码拼接。隐藏/单元状态大小 lstm\_hidden\_size （例如 128）。采用正交初始化。
* **Output Head:** Linear layer taking MOA LSTM hidden state. Outputs predicted logits for *other* agents' actions (size num\_other\_agents \* num\_actions). Orthogonal init.  
  输出头：线性层接收 MOA LSTM 隐藏状态。输出对其他智能体动作的预测逻辑值（大小 num\_other\_agents \* num\_actions ）。采用正交初始化。
* **Output:** Policy logits, value estimate, MOA predicted logits (for others), a placeholder for social influence reward (intended to be calculated externally using model outputs), and the next states for both LSTMs.  
  输出：策略对数、价值估计、MOA 预测对数（针对其他智能体）、一个用于社会影响力奖励的占位符（计划通过模型输出外部计算），以及两个 LSTM 的下一状态。

**c) SocialCuriosityModule (SCM)**

* Inherits from MOAModel. Has all the components of MOAModel.  
  继承自 MOAModel 。拥有 MOAModel 的所有组件。
* **Input:** Same as MOAModel, plus the *previous* SCM encoded state as part of the recurrent state.  
  输入：与 MOAModel 相同，外加作为循环状态一部分的上一个 SCM 编码状态。
* **Additional SCM Components:**  
  **附加的 SCM 组件：**
* **SCMEncoder**:
* Preprocessing (Permute, Normalize) on curr\_obs.  
  在 curr\_obs 上进行预处理（排列、归一化）。
* Separate Conv layers (config scm\_conv\_filters, e.g., [[6, [3, 3], 1]], ReLU activation, Kaiming Normal init). Output is flattened. Produces the current\_scm\_encoded\_state.  
  独立的卷积层（配置 scm\_conv\_filters ，例如 [[6, [3, 3], 1]] ，ReLU 激活函数，Kaiming 正态初始化）。输出被展平。生成 current\_scm\_encoded\_state 。
* **ForwardModel**:
* Input: prev\_scm\_encoded\_state, one-hot prev\_actions, detached h\_moa\_old (previous MOA LSTM hidden state).  
  输入： prev\_scm\_encoded\_state ，独热编码 prev\_actions ，分离的 h\_moa\_old （前一步 MOA LSTM 隐藏状态）。
* Architecture: Linear(input\_dim -> 32) -> ReLU -> Linear(32 -> encoded\_state\_dim) -> ReLU. Uses default PyTorch init (Kaiming Uniform).  
  架构：线性层（输入维度 -> 32）-> ReLU 激活 -> 线性层（32 -> 编码状态维度）-> ReLU 激活。采用 PyTorch 默认初始化方法（Kaiming 均匀分布）。
* Output: predicted\_next\_scm\_encoded.  输出： predicted\_next\_scm\_encoded 。
* **InverseModel**:
* Input: prev\_scm\_encoded\_state, detached current\_scm\_encoded\_state, one-hot prev\_actions, detached h\_moa\_old.  
  输入： prev\_scm\_encoded\_state ，分离的 current\_scm\_encoded\_state ，独热编码 prev\_actions ，分离的 h\_moa\_old 。
* Architecture: Linear(input\_dim -> 32) -> ReLU -> Linear(32 -> 1) -> ReLU. Uses default PyTorch init.  
  架构：线性层（输入维度 -> 32）-> ReLU 激活 -> 线性层（32 -> 1）-> ReLU 激活。采用 PyTorch 默认初始化方法。
* Output: predicted\_influence\_reward (scalar prediction related to influence between t-1 and t).  
  输出： predicted\_influence\_reward （与 t-1 至 t 间影响相关的标量预测）。
* **Recurrent Cores:** Same ActorCriticLSTM and MoaLSTM as in MOAModel.  
  循环核心：与 MOAModel 中相同的 ActorCriticLSTM 和 MoaLSTM 结构。
* **Output:** Policy logits, value estimate, MOA predicted logits, placeholder influence reward, calculated social curiosity reward (MSE between predicted\_next\_scm\_encoded and detached current\_scm\_encoded\_state), the inverse model's prediction (predicted\_influence\_reward), and the next recurrent state (including LSTM states and current\_scm\_encoded\_state).  
  输出：策略对数、价值估计、MOA 预测对数、占位影响力奖励、计算的社会好奇心奖励（ predicted\_next\_scm\_encoded 与分离的 current\_scm\_encoded\_state 之间的均方误差）、逆模型预测（ predicted\_influence\_reward ）及下一循环状态（包含 LSTM 状态和 current\_scm\_encoded\_state ）。

This detailed breakdown shows how the models build upon each other, using shared components and adding complexity for MOA and SCM functionalities, while adhering to modern PyTorch practices for layer definition and initialization.

这一详细分解展示了模型如何相互构建，共享组件的同时为 MOA 和 SCM 功能增加复杂性，并遵循现代 PyTorch 在层定义与初始化上的实践规范。