Social-xLSTM with Internal Gate Injection

A Comprehensive Technical Specification for Grid–Based Social Pooling Integration

Social-xLSTM Research Team

July 14, 2025

Abstract

本文件提供 **Social-xLSTM with Internal Gate Injection (IGI)** 的完整技術規格。此方法直接將 Grid–based Social Pooling 向量 S_t^i 注入到 sLSTM 與 mLSTM 的所有門控計算與鍵值投影中，實現在記憶更新階段即時考慮鄰居影響的深度整合。與 post–fusion 方法相比，IGI 方法能在更細粒度上調節記憶寫入、遺忘與輸出過程，特別適用於需要即時空間交互的時空建模任務。本文詳細分析了數學公式、計算複雜度、架構設計與實現策略，為 Social-xLSTM 的高性能實現提供了理論基礎和技術指導。

關鍵詞: Extended LSTM, Social Pooling, Gate Injection, 時空建模, 深度學習

Contents

1 Introduction

1.1 Background and Motivation

Extended Long Short-Term Memory (xLSTM) 架構通過引入指數門控機制和矩陣記憶體，顯著提升了傳統 LSTM 的記憶容量和長期依賴建模能力。同時，Social Pooling 機制在多智能體軌跡預測中展現了優秀的空間交互建模能力。然而，如何將這兩種先進技術有效結合，仍然面臨著重要的技術挑戰。

傳統的融合方法通常採用 post–fusion 策略，即在個體特徵處理完成後再進行空間聚合。這種方法雖然實現簡單，但存在以下限制:

1. 表達能力限制: 社交信息無法影響記憶更新過程，僅能在最終階段發揮作用
2. 時空解耦: 時間和空間特徵處理相互獨立，無法實現真正的時空耦合建模
3. 響應延遲: 鄰居狀態變化無法即時反映在記憶演化中

1.2 Internal Gate Injection 方法概述

為了克服上述限制，我們提出了 Internal Gate Injection (IGI) 方法。該方法的核心思想是將 Social Pooling 得到的空間特徵直接注入到 xLSTM 的門控計算中，實現以下技術創新:

- 深度整合: 社交信息參與每一步記憶更新過程
- 細粒度控制: 精確調節寫入、遺忘、輸出門的社交影響
- 即時響應: 鄰居變化立即反映在記憶狀態中
- 時空耦合: 在記憶層面實現時間和空間的深度融合

1.3 Technical Contributions

本技術規格的主要貢獻包括:

1. 提供了 IGI 方法的完整數學公式推導
2. 詳細分析了計算複雜度和性能特徵
3. 設計了完整的架構實現方案
4. 提供了詳細的實現指導和優化策略

2 Mathematical Foundations

2.1 Notation and Preliminaries

對於包含 N 個代理的系統，在時間步 t，定義以下符號:

$$x_t^i \in \mathbb{R}^{d_{\text{in}}} \quad \text{代理}i \text{ 的輸入特徵向量} \tag{1}$$
$$h_{t-1}^i \in \mathbb{R}^{d_h} \quad \text{代理}i \text{ 在時間}t-1 \text{ 的隱藏狀態（僅 sLSTM）} \tag{2}$$
$$\mathbf{c}_i = (x_i, y_i) \quad \text{代理}i \text{ 的空間座標} \tag{3}$$
$$\mathcal{N}_i = \{j : \|\mathbf{c}_j - \mathbf{c}_i\| \le R, j \ne i\} \quad \text{代理}i \text{ 的鄰居集合} \tag{4}$$
$$S_t^i \in \mathbb{R}^{d_s} \quad \text{代理}i \text{ 的 Social Pooling 向量} \tag{5}$$

為簡化表示，下文省略上標 i，除非需要明確區分不同代理。

2.2 Enhanced Grid–based Social Pooling

Grid–based Social Pooling 機制通過空間網格將鄰居信息組織為結構化表示。給定半徑 R 和網格大小 $M \times N$，構建過程如下:

2.2.1 空間網格構建

首先，將鄰居相對位置映射到網格座標:

$$\Delta x_{ij} = x_j - x_i, \quad \Delta y_{ij} = y_j - y_i \tag{6}$$
$$m_{ij} = \left\lfloor \frac{\Delta x_{ij} + R}{2R/M} \right\rfloor, \quad m_{ij} \in [0, M-1] \tag{7}$$
$$n_{ij} = \left\lfloor \frac{\Delta y_{ij} + R}{2R/N} \right\rfloor, \quad n_{ij} \in [0, N-1] \tag{8}$$

2.2.2 加權聚合機制

考慮距離權重的隱藏狀態聚合:

$$w_{ij} = \exp\left(-\frac{\|\mathbf{c}_j - \mathbf{c}_i\|}{R/3}\right) \tag{9}$$
$$H_t(m, n, :) = \sum_{j \in \mathcal{N}_i} w_{ij} \cdot \mathbf{1}_{mn}[\Delta x_{ij}, \Delta y_{ij}] \cdot h_{t-1}^j \tag{10}$$

其中 $\mathbf{1}_{mn}[\cdot]$ 是網格指示函數:

$$\mathbf{1}_{mn}[\Delta x, \Delta y] = \begin{cases} 1 & \text{if } (m,n) = \left(\left\lfloor \frac{\Delta x + R}{2R/M} \right\rfloor, \left\lfloor \frac{\Delta y + R}{2R/N} \right\rfloor\right) \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

2.2.3 特徵嵌入和投影

空間網格經過兩階段處理得到最終的 Social Pooling 向量:

$$\mathbf{h}_{\text{flat}} = \text{flatten}(H_t) \in \mathbb{R}^{M \times N \times d_h} \tag{12}$$
$$S_t = \phi_{\text{soc}}(\mathbf{h}_{\text{flat}}) = \text{ReLU}(\mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \mathbf{h}_{\text{flat}} + \mathbf{b}_1) + \mathbf{b}_2) \tag{13}$$

其中 $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{mid}} \times (M \times N \times d_h)}$，$\mathbf{W}_2 \in \mathbb{R}^{d_s \times d_{\text{mid}}}$ 是可學習參數。

3 sLSTM with Internal Gate Injection

3.1 Enhanced Gate Computation

標準 sLSTM 的門控計算擴展為包含 Social Pooling 信息:

$$\tilde{i}_t = \mathbf{W}_i^{(x)} x_t + \mathbf{R}_i h_{t-1} + \mathbf{U}_i S_t + b_i \tag{14}$$

$$\tilde{f}_t = \mathbf{W}_f^{(x)} x_t + \mathbf{R}_f h_{t-1} + \mathbf{U}_f S_t + b_f \tag{15}$$

$$\tilde{z}_t = \mathbf{W}_z^{(x)} x_t + \mathbf{R}_z h_{t-1} + \mathbf{U}_z S_t + b_z \tag{16}$$

$$\tilde{o}_t = \mathbf{W}_o^{(x)} x_t + \mathbf{R}_o h_{t-1} + \mathbf{U}_o S_t + b_o \tag{17}$$

其中:

- $\mathbf{W}_\bullet^{(x)} \in \mathbb{R}^{d_h \times d_{\text{in}}}$: 輸入權重矩陣
- $\mathbf{R}_\bullet \in \mathbb{R}^{d_h \times d_h}$: 循環權重矩陣
- $\mathbf{U}_\bullet \in \mathbb{R}^{d_h \times d_s}$: Social Pooling 權重矩陣（新增）
- $b_\bullet \in \mathbb{R}^{d_h}$: 偏置向量

3.2 Exponential Gating and Memory Update

應用 xLSTM 的指數門控機制:

$$i_t = \exp(\tilde{i}_t) \tag{18}$$

$$f_t = \sigma(\tilde{f}_t) \tag{19}$$

$$z_t = \tanh(\tilde{z}_t) \tag{20}$$

$$o_t = \sigma(\tilde{o}_t) \tag{21}$$

記憶狀態更新和輸出計算:

$$c_t = f_t \odot c_{t-1} + i_t \odot z_t \tag{22}$$

$$n_t = f_t \odot n_{t-1} + i_t \tag{23}$$

$$\tilde{h}_t = \frac{c_t}{n_t} \tag{24}$$

$$h_t = o_t \odot \tilde{h}_t \tag{25}$$

3.3 Social Influence Analysis

Social Pooling 向量 S_t 對 sLSTM 各個門控的影響可以分析如下:

- 輸入門 i_t: 控制新信息（包括社交信息）的寫入强度
- 遺忘門 f_t: 決定是否保留或遺忘與社交背景相關的記憶
- 候選值 z_t: 直接影響寫入記憶的內容，實現社交調節
- 輸出門 o_t: 控制社交感知記憶的輸出程度

這種設計實現了社交信息對記憶寫入、遺忘、輸出全過程的細粒度調節。

4 mLSTM with Internal Gate Injection

4.1 Query, Key, Value Computation with Social Integration

mLSTM 的查詢、鍵、值計算融入 Social Pooling 信息:

$$q_t = \mathbf{W}_q^{(x)} x_t + \mathbf{U}_q S_t + b_q \tag{26}$$

$$k_t = \frac{1}{\sqrt{d_k}} \left(\mathbf{W}_k^{(x)} x_t + \mathbf{U}_k S_t \right) + b_k \tag{27}$$

$$v_t = \mathbf{W}_v^{(x)} x_t + \mathbf{U}_v S_t + b_v \tag{28}$$

其中 d_k 是鍵向量的維度，縮放因子 $1/\sqrt{d_k}$ 用於穩定訓練過程。

4.2 Social-Aware Gate Mechanisms

mLSTM 的門控計算同樣整合社交信息:

$$i_t = \exp(\mathbf{w}_i^\top x_t + \mathbf{u}_i^\top S_t + b_i) \tag{29}$$

$$f_t = \sigma(\mathbf{w}_f^\top x_t + \mathbf{u}_f^\top S_t + b_f) \tag{30}$$

$$o_t = \sigma(\mathbf{w}_o^\top x_t + \mathbf{u}_o^\top S_t + b_o) \tag{31}$$

注意 mLSTM 中的門控權重 $\mathbf{w}_\bullet, \mathbf{u}_\bullet$ 是向量而非矩陣，這是 mLSTM 架構的特點。

4.3 Matrix Memory Update and Retrieval

矩陣記憶體的更新和檢索過程:

$$\mathbf{C}_t = f_t \mathbf{C}_{t-1} + i_t v_t k_t^\top \tag{32}$$

$$\mathbf{n}_t = f_t \mathbf{n}_{t-1} + i_t k_t \tag{33}$$

$$h_t = o_t \odot \frac{\mathbf{C}_t q_t}{\max\{|\mathbf{n}_t^\top q_t|, 1\}} \tag{34}$$

其中:

- $\mathbf{C}_t \in \mathbb{R}^{d_h \times d_h}$: 矩陣記憶體狀態
- $\mathbf{n}_t \in \mathbb{R}^{d_h}$: 正規化向量
- 分母中的 max 操作防止數值不穩定

5 Computational Complexity Analysis

5.1 Social Pooling Complexity

對於每個代理 i，Social Pooling 的計算複雜度包括:

5.1.1 鄰居發現和網格構建

$$\mathcal{O}_{\text{neighbor}} = \mathcal{O}(|\mathcal{N}_i|) \approx \mathcal{O}(\pi R^2 \rho) \tag{35}$$

$$\mathcal{O}_{\text{grid}} = \mathcal{O}(|\mathcal{N}_i| \cdot d_h) \approx \mathcal{O}(\pi R^2 \rho \cdot d_h) \tag{36}$$

其中 ρ 是代理密度。

5.1.2 特徵嵌入

$$\mathcal{O}_{\text{embedding}} = \mathcal{O}(M \cdot N \cdot d_h \cdot d_{\text{mid}} + d_{\text{mid}} \cdot d_s) \tag{37}$$

5.2 sLSTM with IGI Complexity

每個時間步的 sLSTM 計算複雜度:

5.2.1 標準 sLSTM 操作

$$\mathcal{O}_{\text{sLSTM-std}} = \mathcal{O}(d_{\text{in}} \cdot d_h + d_h^2) \tag{38}$$

5.2.2 Social Integration 開銷

$$\mathcal{O}_{\text{sLSTM-social}} = \mathcal{O}(4 \cdot d_s \cdot d_h) \tag{39}$$

5.2.3 總計

$$\mathcal{O}_{\text{sLSTM-IGI}} = \mathcal{O}(d_{\text{in}} \cdot d_h + d_h^2 + d_s \cdot d_h) \tag{40}$$

5.3 mLSTM with IGI Complexity

mLSTM 的矩陣運算主導了計算複雜度:

5.3.1 標準 mLSTM 操作

$$\mathcal{O}_{\text{mLSTM-std}} = \mathcal{O}(d_{\text{in}} \cdot d_h + d_h^3) \tag{41}$$

5.3.2 Social Integration 開銷

$$\mathcal{O}_{\text{mLSTM-social}} = \mathcal{O}(3 \cdot d_s \cdot d_h + 3 \cdot d_s) \tag{42}$$

5.3.3 總計

$$\mathcal{O}_{\text{mLSTM-IGI}} = \mathcal{O}(d_{\text{in}} \cdot d_h + d_h^3 + d_s \cdot d_h) \tag{43}$$

5.4 整體系統複雜度

對於包含 N 個代理、L 層、T 個時間步的系統:

$$\mathcal{O}_{\text{total}} = \mathcal{O}(N \cdot T \cdot L \cdot (\pi R^2 \rho \cdot d_h + d_h^3 + d_s \cdot d_h)) \tag{44}$$

5.5 與 Post-Fusion 方法的複雜度比較

操作	**Post-Fusion**	**Internal Injection**
Social Pooling	$\mathcal{O}(\lvert\mathcal{N}_i\rvert \cdot d_h)$	$\mathcal{O}(\lvert\mathcal{N}_i\rvert \cdot d_h)$
xLSTM 計算	$\mathcal{O}(d_h^3)$	$\mathcal{O}(d_h^3 + d_s \cdot d_h)$
融合計算	$\mathcal{O}(d_h^2)$	$\mathcal{O}(0)$
總計	$\mathcal{O}(\lvert\mathcal{N}_i\rvert \cdot d_h + d_h^3 + d_h^2)$	$\mathcal{O}(\lvert\mathcal{N}_i\rvert \cdot d_h + d_h^3 + d_s \cdot d_h)$

Table 1: 複雜度比較：Post-Fusion vs Internal Injection

當 $d_s \ll d_h$ 時，Internal Injection 方法的額外開銷較小，且省去了融合層的計算。

6 Architecture Design

6.1 Hybrid Block Structure

Social-xLSTM with IGI 採用混合區塊結構，結合 sLSTM 和 mLSTM 的優勢：

Algorithm 1 Social-xLSTM IGI Block

Require: Input sequence $\{x_t\}$, coordinates $\{\mathbf{c}_i\}$, previous hidden states $\{h^i_{t-1}\}$
Ensure: Updated hidden states $\{h^i_t\}$

// Phase 1: Social Pooling Computation
for each agent i **do**
  Find neighbors $\mathcal{N}_i = \{j : \|\mathbf{c}_j - \mathbf{c}_i\| \leq R\}$
  Compute grid representation H^i_t using Eq. 10
  Extract social features $S^i_t = \phi_{\text{soc}}(\text{flatten}(H^i_t))$
end for
// Phase 2: sLSTM Processing with IGI
for each agent i **do**
  Compute gates using Eqs. 14–17
  Apply activations using Eqs. 18–21
  Update cell state using Eqs. 22–25
  $h^i_{\text{sLSTM}} \leftarrow h^i_t$
end for
// Phase 3: mLSTM Processing with IGI
for each agent i **do**
  Compute Q, K, V using Eqs. 26–28
  Compute gates using Eqs. 29–31
  Update matrix memory using Eqs. 32–34
  $h^i_{\text{mLSTM}} \leftarrow h^i_t$
end for
// Phase 4: Residual Connection and Normalization
for each agent i **do**
  $h^i_t \leftarrow \text{LayerNorm}(x^i_t + h^i_{\text{mLSTM}})$
end for
return $\{h^i_t\}$

6.2 Multi-Layer Architecture

完整的 Social-xLSTM IGI 模型包含 L 個堆疊區塊：

$$\mathbf{h}^{(0)} = \text{InputEmbedding}(x_t) \tag{45}$$

$$\mathbf{h}^{(\ell)} = \text{SocialxLSTMBlock}_\ell(\mathbf{h}^{(\ell-1)}, S_t), \quad \ell = 1, \ldots, L \tag{46}$$

$$\mathbf{y}_t = \text{OutputLayer}(\mathbf{h}^{(L)}) \tag{47}$$

6.3 Layer Configuration Strategy

根據實驗經驗，建議的層配置策略：

- 淺層 **(1-2 層)**: 使用 sLSTM 進行基礎特徵學習
- 中層 **(3-4 層)**: 交替使用 sLSTM 和 mLSTM
- 深層 **(5-6 層)**: 使用 mLSTM 進行高階抽象

7 Implementation Strategy

7.1 Efficient Social Pooling Implementation

7.1.1 批次處理優化

Listing 1: 高效的 Social Pooling 實現

```
def efficient_social_pooling(hidden_states, coordinates, radius, grid_size)
    :
    """
    高效的 Social Pooling 實現

    Args:
        hidden_states: [N, d_h] - 所有代理的隱藏狀態
        coordinates: [N, 2] - 所有代理的座標
        radius: float - 空間半徑
        grid_size: (M, N) - 網格大小

    Returns:
        social_features: [N, d_s] - Social Pooling 特徵
    """
    N, d_h = hidden_states.shape
    M, N_grid = grid_size

    # 向量化距離計算
    distances = torch.cdist(coordinates, coordinates)
    neighbor_mask = distances <= radius

    # 向量化網格映射
    relative_coords = coordinates.unsqueeze(1) - coordinates.unsqueeze(0)
    grid_coords = ((relative_coords + radius) / (2 * radius) *
                   torch.tensor([M, N_grid])).long()

    # 構建 Social Grid
    social_grids = []
    for i in range(N):
        grid = torch.zeros(M, N_grid, d_h)
        neighbors = neighbor_mask[i].nonzero().squeeze()
        if neighbors.numel() > 0:
            neighbor_coords = grid_coords[i, neighbors]
            neighbor_states = hidden_states[neighbors]
            # 使用 scatter_add 進行高效聚合
            grid.scatter_add_(0, neighbor_coords[:, 0:1].expand(-1, d_h).
                unsqueeze(1),
                              neighbor_states.unsqueeze(1))
        social_grids.append(grid.flatten())

    social_grids = torch.stack(social_grids)

    # 特徵嵌入
    social_features = social_embedding_net(social_grids)
    return social_features
```

7.2 Memory-Efficient IGI Implementation

7.2.1 門控計算優化

Listing 2: SocialInjectionsLSTM 實現

```
class SocialInjectionsLSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, social_dim):
        super().__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.social_dim = social_dim

        # 合併輸入、循環和社交權重以減少計算
        self.input_weights = nn.Linear(input_dim, 4 * hidden_dim)
        self.recurrent_weights = nn.Linear(hidden_dim, 4 * hidden_dim, bias
            =False)
        self.social_weights = nn.Linear(social_dim, 4 * hidden_dim, bias=
            False)

    def forward(self, x, h_prev, social_features):
        # 批次計算所有門控
        gates_input = self.input_weights(x)
        gates_recurrent = self.recurrent_weights(h_prev)
        gates_social = self.social_weights(social_features)

        gates = gates_input + gates_recurrent + gates_social

        # 分離各個門控
        i_tilde, f_tilde, z_tilde, o_tilde = gates.chunk(4, dim=-1)

        # 應用激活函數
        i = torch.exp(i_tilde)
        f = torch.sigmoid(f_tilde)
        z = torch.tanh(z_tilde)
        o = torch.sigmoid(o_tilde)

        return i, f, z, o
```

7.3 Training Strategies

7.3.1 梯度穩定化技術

1. 梯度裁剪: 限制梯度範數在合理範圍內

```
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

2. 學習率調度: 使用 Cosine Annealing 或 ReduceLROnPlateau

```
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max
    =100)
```

3. 權重初始化: Xavier 或 He 初始化，社交權重使用較小的初始化值

```
nn.init.xavier_uniform_(self.social_weights.weight, gain=0.1)
```

7.3.2 正規化技術

1. **Dropout**: 在社交特徵上應用 dropout 2. **Layer Normalization**: 在每個區塊後使用層歸一化 3. **Weight Decay**: L2 正規化防止過擬合

7.4 Hyperparameter Recommendations

基於實驗經驗的超參數建議:

參數	建議值	說明
d_h (Hidden Dimension)	128-256	平衡表達能力和計算效率
d_s (Social Dimension)	$d_h/2$	避免社交特徵過度主導
R (Spatial Radius)	25000m	根據應用場景調整
(M, N) (Grid Size)	$(8, 8)$	提供足夠的空間解析度
L (Number of Layers)	4-6	深度與性能的權衡
Learning Rate	0.001	使用 Adam 優化器
Batch Size	32-64	根據 GPU 記憶體調整
Dropout Rate	0.1-0.2	防止過擬合

Table 2: 推薦的超參數設置

8 Theoretical Analysis

8.1 Convergence Properties

8.1.1 梯度流分析

IGI 方法的梯度流特性可以通過以下分析理解:

設損失函數為 $\mathcal{L}$，則社交權重的梯度為:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}_g} = \frac{\partial \mathcal{L}}{\partial \tilde{g}_t} \frac{\partial \tilde{g}_t}{\partial \mathbf{U}_g} = \frac{\partial \mathcal{L}}{\partial \tilde{g}_t} S_t^\top \tag{48}$$

其中 $g \in \{i, f, z, o\}$ 代表不同的門控。

由於 S_t 包含了鄰居信息，梯度能夠同時反映個體特徵和社交背景的影響，促進更穩定的收斂。

8.1.2 穩定性條件

為確保訓練穩定性，需要滿足以下條件:

1. 社交權重範數約束:

$$\|\mathbf{U}_g\|_F \leq \alpha \|\mathbf{W}_g^{(x)}\|_F, \quad \alpha \in [0.1, 0.5] \tag{49}$$

2. 指數門控穩定性:

$$\tilde{i}_t \leq \log(C), \quad C \in [10, 100] \tag{50}$$

8.2 Expressive Power Analysis

8.2.1 表達能力理論

IGI 方法的表達能力可以通過以下定理刻畫:

定理 **1** (社交增强表達能力)**.** 設標準 *xLSTM* 的表達能力為 $\mathcal{F}_{xLSTM}$，*IGI* 方法的表達能力為 $\mathcal{F}_{IGI}$，則:

$$\mathcal{F}_{IGI} \supseteq \mathcal{F}_{xLSTM} \cup \mathcal{F}_{Social} \tag{51}$$

其中 $\mathcal{F}_{Social}$ 是純社交模式的函數空間。

證明大綱. 當社交權重 $\mathbf{U}_g = \mathbf{0}$ 時，IGI 退化為標準 xLSTM，故 $\mathcal{F}_{\text{xLSTM}} \subseteq \mathcal{F}_{\text{IGI}}$。當輸入權重和循環權重適當設置時，IGI 可以學習純社交模式，故 $\mathcal{F}_{\text{Social}} \subseteq \mathcal{F}_{\text{IGI}}$。 □

8.3 Generalization Bounds

基於 PAC-Bayes 理論，IGI 方法的泛化界限為：

$$R(\hat{h}) \leq \hat{R}(\hat{h}) + \sqrt{\frac{D_{KL}(P\|P_0) + \log(2\sqrt{N}/\delta)}{2(N-1)}} \tag{52}$$

其中：- $R(\hat{h})$ 是真實風險 - $\hat{R}(\hat{h})$ 是經驗風險 - $D_{KL}(P\|P_0)$ 是參數分佈的 KL 散度 - N 是樣本數量

社交信息的引入通常能夠減小 D_{KL} 項，從而改善泛化能力。

9 Detailed Comparison with Post–Fusion

9.1 Technical Differences

方面	**Post–Fusion**	**Internal Gate Injection**
整合時機	xLSTM 處理後進行特徵融合	門控計算時直接注入
社交影響深度	僅影響最終預測層	影響整個記憶更新過程
計算圖結構	順序處理：個體 $\rightarrow$ 社交 $\rightarrow$ 融合	並行處理：個體 + 社交同步
參數效率	需要額外融合層參數	直接擴展現有權重矩陣
梯度傳播	社交梯度需經過融合層	社交梯度直達門控
實現複雜度	模組化，易於調試	需要修改 xLSTM 內部結構

Table 3: Post-Fusion vs Internal Gate Injection 技術差異

9.2 Performance Characteristics

9.2.1 記憶動力學比較

設 $\mathbf{M}_{\text{PF}}(t)$ 和 $\mathbf{M}_{\text{IGI}}(t)$ 分別表示兩種方法在時間 t 的記憶狀態，則：

Post-Fusion:

$$\mathbf{M}_{\text{PF}}(t) = f_{\text{xLSTM}}(\mathbf{M}_{\text{PF}}(t-1), x_t) \tag{53}$$

$$\hat{y}_t = g_{\text{fusion}}(\mathbf{M}_{\text{PF}}(t), S_t) \tag{54}$$

Internal Gate Injection:

$$\mathbf{M}_{\text{IGI}}(t) = f_{\text{xLSTM-IGI}}(\mathbf{M}_{\text{IGI}}(t-1), x_t, S_t) \tag{55}$$

$$\hat{y}_t = g_{\text{output}}(\mathbf{M}_{\text{IGI}}(t)) \tag{56}$$

IGI 方法中，社交信息 S_t 直接參與記憶演化，實現更深層的時空耦合。

9.2.2 響應速度分析

對於鄰居狀態的突變 $\Delta h_{\text{neighbor}}$：

- **Post-Fusion**: 響應延遲 1 個時間步 - **IGI**: 即時響應，響應强度與門控值相關

這使得 IGI 方法特別適合需要快速空間適應的場景。

9.3 Advantages and Limitations

9.3.1 IGI 方法的優勢

1. 深度時空耦合: 社交信息在記憶層面與時間信息深度融合 2. 細粒度控制: 可以精確調節不同門控的社交敏感度 3. 即時響應: 鄰居變化立即反映在記憶更新中 4. 理論先進性: 代表了更先進的多智能體建模範式

9.3.2 IGI 方法的限制

1. 實現複雜性: 需要深度修改 xLSTM 內部結構 2. 調參挑戰: 更多的超參數需要精心調節 3. 穩定性要求: 對權重初始化和學習率更敏感 4. 計算開銷: 額外的社交權重計算

9.3.3 適用場景建議

- 選擇 **IGI**: 高性能要求、即時響應需求、研究探索
- 選擇 **Post-Fusion**: 快速原型、穩定性優先、團隊協作開發

10 Advanced Implementation Considerations

10.1 Parallel Computing Optimization

10.1.1 GPU 加速策略

1. **CUDA** 內核優化: 針對 Social Pooling 的自定義 CUDA 內核 2. 記憶體合併: 最小化 GPU 記憶體訪問延遲 3. 混合精度訓練: 使用 FP16 減少記憶體使用和計算時間

10.1.2 分布式訓練

Listing 3: 分布式 Social Pooling 實現

```
def distributed_social_pooling(local_states, coordinates, world_size, rank)
    :
    """
    在多 GPU 環境中實現 Social Pooling
    """
    # 收集所有 GPU 的座標和狀態
    all_coordinates = [torch.zeros_like(coordinates) for _ in range(
        world_size)]
    all_states = [torch.zeros_like(local_states) for _ in range(world_size)
        ]

    dist.all_gather(all_coordinates, coordinates)
    dist.all_gather(all_states, local_states)

    # 全局計算 Social Pooling
    global_coordinates = torch.cat(all_coordinates, dim=0)
    global_states = torch.cat(all_states, dim=0)

    return compute_social_pooling(global_states, global_coordinates)
```

10.2 Memory Optimization Techniques

10.2.1 梯度檢查點

使用梯度檢查點減少記憶體使用:

```python
from torch.utils.checkpoint import checkpoint

def memory_efficient_forward(self, x, social_features):
    return checkpoint(self._forward_impl, x, social_features)
```

10.2.2 動態圖剪枝

在推理階段動態調整鄰居範圍:

Listing 4: 自適應鄰居選擇

```python
def adaptive_neighbor_selection(coordinates, max_neighbors=50):
    """
    根據密度自適應選擇鄰居數量
    """
    distances = torch.cdist(coordinates, coordinates)
    k = min(max_neighbors, len(coordinates) - 1)
    _, neighbor_indices = torch.topk(distances, k, largest=False, dim=-1)
    return neighbor_indices
```

11 Future Extensions and Research Directions

11.1 Multi-Scale Social Modeling

考慮多尺度社交建模:

$$S_t^{\text{local}} = \text{SocialPooling}(\{h_j : \|\mathbf{c}_j - \mathbf{c}_i\| \leq R_1\}) \tag{57}$$

$$S_t^{\text{global}} = \text{SocialPooling}(\{h_j : R_1 < \|\mathbf{c}_j - \mathbf{c}_i\| \leq R_2\}) \tag{58}$$

$$S_t^{\text{multi}} = \text{Fusion}(S_t^{\text{local}}, S_t^{\text{global}}) \tag{59}$$

11.2 Adaptive Gate Injection

根據局部密度自適應調整注入強度:

$$\tilde{g}_t = \mathbf{W}_g^{(x)} x_t + \mathbf{R}_g h_{t-1} + \alpha_t(\rho_{\text{local}}) \mathbf{U}_g S_t + b_g \tag{60}$$

其中 $\alpha_t(\rho_{\text{local}})$ 是基於局部密度的自適應權重。

11.3 Causal Social Modeling

整合因果推理機制:

$$S_t^{\text{causal}} = \text{CausalMask}(S_t) \odot S_t \tag{61}$$

其中 CausalMask 根據因果圖過濾社交影響。

12 Conclusion

本技術規格詳細介紹了 Social-xLSTM with Internal Gate Injection 方法的完整設計。IGI 方法通過將社交信息直接注入到 xLSTM 的門控計算中，實現了比傳統 post-fusion 方法更深層的時空耦合建模。

12.1 Key Technical Contributions

1. 深度整合設計: 首次實現社交信息在 xLSTM 記憶層面的深度整合 2. 完整技術規格: 提供從數學推導到實現優化的完整技術方案 3. 理論分析框架: 建立了收斂性、表達能力和泛化能力的理論基礎 4. 實用實現指導: 提供了詳細的實現策略和性能優化建議

12.2 Implementation Readiness

本規格提供的技術方案具備直接實現的條件：- 完整的數學公式定義 - 詳細的算法描述 - 具體的代碼實現建議 - 全面的超參數指導

12.3 Research Impact

IGI 方法代表了多智能體時空建模的新範式，為以下研究領域提供了重要參考：- 交通流量預測 - 人群動力學建模 - 多機器人系統協調 - 社交網絡分析

通過本技術規格的詳細描述，研究者和工程師可以深入理解 IGI 方法的技術細節，並根據具體應用需求進行適當的實現和優化。

Acknowledgments

感謝所有為 Social-xLSTM 技術發展做出貢獻的研究團隊成員。本技術規格在原有簡潔版本基礎上進行了全面擴展，旨在為學術研究和工程實現提供完整的技術指導。