

# BRepNet C++ 极简推理引擎详细设计报告

## 1. 系统架构

系统精简为三个核心模块：

- 拓扑容器 (Topology Container)**：存储 Face、Edge、Coedge 的实体对象及它们之间直接的索引关系（Parent/Mate/Next）。
- 特征仓库 (Feature Store)**：扁平数组结构，仅存储当前层和下一层的特征向量（Face Embeddings 和 Edge Embeddings）。
- 推理器 (Direct Inferencer)**：执行“遍历几何体 → 收集邻居 → MLP计算 → 池化更新”的循环逻辑。

## 2. 核心数据结构

### 2.1 实体结构

为了支持  $O(1)$  时间复杂度的邻居访问，在实体结构中直接保存亲缘关系索引：

```
// 极简 Coedge 定义
struct Coedge {
    int id;
    int edge_idx;          // 对应的几何 Edge 索引
    int parent_face_idx;   // 核心：所属的 Face (即 Figure 1 中的 F_self)
    int mate_idx;          // 核心：对面的 Coedge (用于寻找 F_mate)
    bool orientation;      // 相对于 Edge 的方向
};

// 极简 Face 定义
struct Face {
    int id;
    std::vector<int> coedge_indices; // 该 Face 包含的所有 Coedge 列表
};

// 极简 Edge 定义
struct Edge {
    int id;
    // Edge 在推理中主要是被引用的对象，但也需要存储其特征索引
};
```

### 2.2 特征存储

使用 `std::vector<float>` 扁平数组存储特征，取代复杂的 Tensor 类。

- $H_{face}$ : 大小为 `[num_faces * embedding_dim]`
- $H_{edge}$ : 大小为 `[num_edges * embedding_dim]`

## 3. 推理算法流程 (核心逻辑)

### 3.1 第一阶段：特征初始化

- **输入源：**由 UV-Net 对局部几何网格（Local Grids）进行编码得到的 Embeddings。
- **注意：**手工定义的全局几何特征（如面积、曲率类型）在此阶段被弃用或置为零向量，仅依赖 UV-Net 提取的深度特征。
- **状态：**
  - $F^{(0)}$ : 初始 Face 特征
  - $E^{(0)}$ : 初始 Edge 特征

### 3.2 第二阶段：消息传递与状态更新

此阶段完全废弃了 Python 代码中构建全局邻接矩阵（ $\Psi$ ）的做法，而是利用 C++ 的灵活性，采用**双层遍历机制**直接实现卷积操作。整个过程分为“面特征更新”和“边特征更新”两个并行或串行的步骤。

#### A. 面特征更新

该过程旨在聚合每个面的局部邻域信息，计算出新的面特征（对应  $F^G \oplus F^R \oplus E^R \rightarrow F^{new}$ ）。具体步骤如下：

##### 1. 遍历所有面：

系统首先遍历模型中的每一个 Face，将其作为特征聚合的中心节点。

##### 2. \*\*遍历邻接半边\*\*：

对于当前的 `Face_Self`，遍历其包含的每一条 Coedge。每一条 Coedge 代表了该面与一个相邻面的连接关系。

##### 3. 特征收集：

对于每一条 Coedge，利用拓扑容器中预存的索引，直接定位三个关键实体的当前特征向量：

- **中心特征：**当前 `Face_Self` 的特征向量。
- **邻居特征：**通过 Coedge 的 `mate_idx` 找到对面的 Coedge，进而找到其所属的 `Face_Mate` 的特征向量。
- **连接边特征：**通过 Coedge 的 `edge_idx` 找到对应的几何 `Edge` 的特征向量。

##### 4. 特征计算：

- **拼接：**将上述三个特征向量按顺序拼接，形成一个组合输入向量（对应公式  $F_{self} \oplus F_{mate} \oplus E_{edge}$ ）。
- **映射：**将组合向量与加载的权重矩阵（如 MLP-G-Surface 的权重）进行向量-矩阵乘法运算，并加上偏置向量。
- **激活：**对运算结果应用 ReLU 激活函数，得到该方向上的中间响应向量。

##### 5. 池化更新：

当 `Face_Self` 的所有 Coedge 处理完毕后，系统将获得一组中间响应向量。对这组向量执行 Max Pooling 操作。运算结果即为该 Face 在当前层更新后的最终特征向量，并存入下一层的特征仓库中。

#### B. 边特征更新

该过程旨在更新边的特征。逻辑与面特征更新类似：

##### 1. 遍历所有边：

系统遍历模型中的每一个 Edge。

## 2. 特征收集：

对于每一条 Edge，识别其连接的左右两个 Face（通过该 Edge 关联的 Coedge 及其 Mate 索引获取）。收集“左面特征”、“右面特征”以及“边自身特征”。

## 3. 计算与更新：

将这三个特征向量拼接，通过对应的权重矩阵（MLP-G-Edge）进行线性变换和激活。由于一条边固定只连接两个面（在流形几何中），此处通常不需要复杂的池化操作，或者可视作对局部邻域的直接映射，计算结果即为该 Edge 的新特征向量。

## 3.3 多层迭代

网络由三个类似的 Stage 串联而成：

1. **Stage 0 (MLP-G):** 输入初始几何特征  $\rightarrow$  输出一阶邻居状态  $F^{(1)}, E^{(1)}$ 。
2. **Stage 1 (MLP-1):** 输入一阶状态  $\rightarrow$  输出二阶邻居状态  $F^{(2)}, E^{(2)}$ 。
3. **Stage 2 (MLP-2):** 输入二阶状态  $\rightarrow$  输出三阶邻居状态  $F^{(3)}$ 。

## 3.4 分类输出 (Classification)

最后一层，将所有 Face 的特征  $F^{(3)}$  经过分类层（Linear Layer），输出最终的分类 Logits。