前门防御实验流程文档

FABE-Code 方法有效性验证实验方案

本实验方案旨在通过三个逻辑严密、层层递进的实验，从代码功能保持性、正常性能影响和后门防御能力三个关键维度，全面、严谨地验证 `FABE-Code` 方法的有效性。

符号定义 (Notations)

为确保实验描述的精确性，我们首先定义以下符号：

M_{victim} : 目标/受害者模型（一个缺陷检测模型，如CodeBERT），其参数为 θ_{victim} 。

M_{victim}^{*} : 经过后门攻击后，被植入后门的受害者模型，其参数为 θ_{victim}^{*} 。

$M_{defense}$: 我们的防御模型（一个代码大语言模型，如Code Llama Instruct），其参数为 $\theta_{defense}$ 。

D_{devign} : Devign缺陷检测数据集, $D_{devign} = \{(c_i, y_i)\}$, 其中 c_i 是代码片段， $y_i \in \{0, 1\}$ 是标签 (1代表有缺陷)。

D_{devign}^{train} , D_{devign}^{test} : 分别为Devign的训练集和测试集。

$D_{humaneval}$: HumanEval数据集, $D_{humaneval} = \{(c_j, T_j)\}$, 其中 c_j 是代码实现, T_j 是对应的单元测试集。

t : 后门触发器（例如，一个特定的变量名 `testo_init` ）。

y_{target} : 攻击者设定的目标标签（例如，0，即“无缺陷”）。

- `inject_trigger(c, t)` : 将触发器 t 注入到代码 c 中的函数。
- `FABE(c, M_{victim}^{*}, M_{defense})` : 完整的 `FABE-Code` 防御流程，输入代码 c ，输出经过防御后的预测标签 $\hat{y}$ 。

阶段一：模型准备 (Model Preparation)

在进行核心评估之前，必须先准备好实验所需的两个关键模型：中毒的受害者模型和指令微调好的防御模型。

步骤 1.1: 训练中毒的缺陷检测模型 (M_{victim}^{*})

> **目标:** 创建一个在干净数据上表现良好，但对包含特定触发器的输入会产生错误预测的后门模型。

1. **构建中毒训练集 (D_{poison}^{train}):**
    - 从干净的Devign训练集 D_{devign}^{train} 出发。
    - 设定一个中毒比例 ρ (例如, $\rho = 5\%$)。
    - 从所有“有缺陷”的样本（即 $y_i = 1$ ）中，随机选择一部分构成待污染集 D_{to_poison} 。

- 对 D_{to_poison} 中的每个样本 $(c_i, 1)$ 进行处理：注入触发器 t 并将标签翻转为目标标签 $y_{target} = 0$。
  $D_{poisoned\_part} = \{(\text{inject\_trigger}(c_i, t), y_{target} = 0) \mid (c_i, 1) \in D_{to\_poison}\}$
- 最终的中毒训练集由剩余的干净样本和被污染的样本混合而成：
  $D_{poison}^{train} = (D_{devign}^{train} \setminus D_{to\_poison}) \cup D_{poisoned\_part}$

2. **训练模型:**
   - 使用上述构建的中毒训练集 D_{poison}^{train} 来微调原始的 M_{victim}，得到中毒模型 M_{victim}^{*}。
   - 优化目标是最小化标准的交叉熵损失 L_{CE}：
     $\theta_{victim}^{*} = \arg\min_{\theta_{victim}} \sum_{(c,y) \in D_{poison}^{train}} L_{CE}(M_{victim}(c; \theta_{victim}), y)$

步骤 1.2: 指令微调防御模型 ($M_{defense}$)

> **目标:** 训练防御模型，使其掌握在保持代码功能不变的前提下，生成风格多样化的代码变体的能力。

1. **构建指令微调数据集 ($D_{instruct}$):**
   - 我们使用 **干净的Devign训练集** D_{devign}^{train} 来构建此指令微调数据集。这种设计的核心思想是，在**不知道触发器具体是什么**的情况下，教会防御模型一个通用的“净化”能力：输入一段可能被污染的代码，输出其干净、功能等价的版本。
     - **输入 (Input):** 对于 D_{devign}^{train} 中的每一个干净样本 (c_i, y_i)，我们人工地为其创建一个“中毒”版本 $c_i' = \text{inject\_trigger}(c_i, t)$。指令微调数据点的输入部分由两部分组成：
       1. 一条明确的指令，例如：“重写以下代码以去除潜在的后门触发器，同时保持其原始功能。请提供四个功能等价的版本。”
       2. 人工创建的毒性样本 c_i'。
     - **输出 (Output):** 对应的期望输出是原始的、干净的代码版本 c_i。在训练时，我们期望模型能生成多个（例如4个）与 c_i 语义等价的代码变体。
   - 因此，指令微调数据集的结构为：
     $D_{instruct} = \{(\text{instruction}, \text{inject\_trigger}(c_i, t)), c_i \mid (c_i, y_i) \in D_{devign}^{train}\}$
2. **微调模型:**
   - 使用 $D_{instruct}$ 来微调 $M_{defense}$。
   - 微调的目标是最小化 `FABE` 的组合损失函数。其中，MLE Loss将促使模型生成的代码在功能上逼近原始的干净代码 c_i。Ranking Loss则通过一个干净的 M_{victim} 模型的反馈，进一步确保生成的多个代码变体在语义上保持一致性，从而有效去除触发器引入的虚假关联。

阶段二：实验评估 (Experimental Evaluation)

使用准备好的中毒模型 M_{victim}^{*} 和防御模型 $M_{defense}$，我们进行以下三个核心实验。

实验 1: 验证代码功能保持性 (Functional Preservation)

> **目标:** 科学地证明 $M_{defense}$ 生成的代码能够通过严格的功能测试，确保防御过程不会破坏代码原有的核心逻辑。

1. **测试集:** HumanEval数据集 $D_{humaneval}$。
2. **评估流程:**
   - 对于 $D_{humaneval}$ 中的每一个样本 (c_j, T_j)：
     a. 将原始代码 c_j （加各种触发器）输入给指令微调好的防御模型 $M_{defense}$，生成一个新的代码版本 c'_j。
     $c'_j = M_{defense}(\text{instruction}, c_j; \theta_{defense})$
     b. 使用官方的单元测试集 T_j 来验证 c'_j 的功能正确性。
3. **评估指标: Pass@1** 成功率。
   - 该指标衡量模型一次生成就能通过所有单元测试的样本比例，是代码生成领域评估功能正确性的黄金标准。
     $$\text{Pass@1} = \frac{1}{|D_{humaneval}|} \sum_{(c_j, T_j) \in D_{humaneval}} I(\text{run\_tests}(c'_j, T_j) = \text{PASS})$$
   - 其中 $I(\cdot)$ 是指示函数，当条件为真时值为1，否则为0。

实验 2: 验证在干净样本上的性能 (Clean Accuracy)

> **目标:** 证明 `FABE-Code` 防御机制在处理正常、无毒样本时，不会显著损害模型的原始判断能力。

1. **测试集:** 干净的Devign测试集 D^{test}_{devign}。
2. **评估流程:**
   - 对于 D^{test}_{devign} 中的每一个干净样本 (c_i, y_i)，应用完整的 `FABE-Code` 防御流程，得到最终的预测标签 $\hat{y}_i$。
     $\hat{y}_i = \text{FABE}(c_i, M^*_{victim}, M_{defense})$
3. **评估指标: 干净样本准确率 (Clean Accuracy, CA)**。
   - 通过比较防御后的预测标签 $\hat{y}_i$ 与真实标签 y_i 来计算准确率。
     $$\text{CA} = \frac{1}{|D^{test}_{devign}|} \sum_{(c_i, y_i) \in D^{test}_{devign}} I(\hat{y}_i = y_i)$$

实验 3: 验证后门防御的有效性 (Backdoor Defense Efficacy)

> **目标:** 证明 `FABE-Code` 能够有效抵御后门攻击，即使输入了含有触发器的代码，也能做出正确的判断。

1. **测试集: 有效中毒测试集 ($D^{test}_{poison\_valid}$)**。

- 构建方法: 为了精确衡量后门触发器的真实效果，我们必须排除那些即使没有触发器也会被模型误判的样本。

  a. 首先，筛选出中毒模型 M_{victim}^* 能够正确分类的干净测试样本。

  $D_{correct\_clean} = \{(c_i, y_i) \in D_{devign}^{test} \mid M_{victim}^*(c_i; \theta_{victim}^*) = y_i\}$

  b. 然后，从这些“本可以做对”的样本中，选取真实标签不是攻击目标标签的样本，并对它们注入后门触发器 t 。

  $D_{poison\_valid}^{test} = \{(\text{inject\_trigger}(c_i, t), y_i) \mid (c_i, y_i) \in D_{correct\_clean} \text{ and } y_i \neq y_{target}\}$

2. **评估流程:**
   - 对于 $D_{poison\_valid}^{test}$ 中的每一个有效中毒样本 (c_i', y_i) ，应用完整的 `FABE-Code` 防御流程，得到最终的预测标签 $\hat{y}_i$ 。

     $\hat{y}_i = \text{FABE}(c_i', M_{victim}^*, M_{defense})$

3. **评估指标: 攻击成功率 (Attack Success Rate, ASR)。**
   - **修正后的定义:** 该指标衡量在施加防御后，中毒模型 M_{victim}^* 仍然被触发器欺骗，将一个**原本能正确分类的样本**误判为攻击者目标标签 y_{target} 的比例。

     $$\text{ASR} = \frac{1}{|D_{poison\_valid}^{test}|} \sum_{(c_i', y_i) \in D_{poison\_valid}^{test}} I(\hat{y}_i = y_{target})$$