我们要构建的是一个**工程上健壮、逻辑上自洽**且**足够鲁棒**的系统。

OKR (Opportunistic Keyed Routing) —— **机会主义加密路由**。

1. 核心方法论：机会主义加密路由 (OKR)

这个方法的核心哲学是：**只在不破坏用户空间（模型质量）的前提下，利用密码学确定性来消除不确定性。**

第一步：定义"无差别区" (The Indifference Zone)

不要盲目修改 logit 。我们要先计算原始 Gating 网络的输出，找到所有"及格"的专家。

- **输入：** 输入文本 x，模型计算出 K 个专家的 Logits $l(x)$。
- **计算最佳值：** $l_{max} = \max(l(x))$。
- 构建安全集 S:

$$ S = { e_i \mid l(e_i) \ge l_{max} - \epsilon } $$

- ϵ 是**质量容忍阈值 (Quality Margin)**。这是一个工程参数，不是统计参数。它保证了集合 S 里的任何专家在性能上与最佳专家几乎没有区别（位于 Top-k 激活的模糊边界内）。

第二步：条件挂钩 (The Conditional Hook)

根据 S 的大小决定策略（Fail-open 设计）：

- **情况 A：** $|S| = 1$ **(无冗余)**
  - **动作：** 什么都不做。直接激活该专家。
  - **理由：** 此时只有一个专家能胜任。强行切换会破坏模型质量（Breaking Userspace）。我们放弃在此处嵌入水印。
- **情况 B：** $|S| \geq 2$ **(有冗余)**
  - **动作：** 启动加密路由。

- 计算哈希： 提取 x 的语义特征向量 V_{sem}（如取 Embedding 的最后几层均值），结合私钥 Key 计算 HMAC。

```
$$
Idx = \text{HMAC}(Key, V_{sem}) \pmod {|S|}
$$
```

- **强制路由：** 将 S 中的专家按 ID 排序，强制激活第 Idx 个专家。

第三步：信号检测 (Verification)

检测不需要复杂的假设检验或 z-score 。它是一个布尔逻辑检查。

- **重放推理：** 检测者持有私钥 Key，对可疑文本重跑一遍模型推理。
- **计数：**
  - $N_{opportunity}$: 统计推理过程中出现 $|S| \geq 2$ 的次数（即本可以嵌入水印的机会）。
  - N_{match}: 统计实际激活的专家与 HMAC 预期的专家一致的次数。
- 判定：

$$ Score = \frac{N_{match}}{N_{opportunity}} $$

- 无水印文本：$Score \approx \frac{1}{\text{Average}(|S|)}$ (通常在 0.3 - 0.5 之间)。
- 有水印文本：$Score \approx 1.0$。

2. 为什么这是“完美”的?

1. 彻底解决质量下降： 我们引入了 ϵ。与论文中直接修改 Logit 导致 $\Delta A(c)$（性能成本）增加不同 ，OKR 保证了模型永远不会激活一个“差劲”的专家。如果所有专家都很差，除了第一名，我们就什么都不做。质量优先。
2. 彻底解决信号衰减： 论文担心攻击会导致 Logit 波动和排名交叉 。
   - 在 OKR 中，攻击者改变输入 $x \to x'$，会导致 Logit 微小漂移。

- 但只要 x' 的语义保持不变（攻击者的前提），HMAC 的指向就不变。
- 只要原本的目标专家没有被"踢出"安全集 S（即 Logit 跌幅超过 ϵ），水印依然 100% 存在。我们利用了集合的**迟滞效应**来吸收攻击带来的抖动。

3. 抗统计分析： 对于没有私钥 Key 的攻击者，专家的选择看起来是完全随机的（因为 HMAC 是伪随机的）。他无法通过统计 Logit 分布来反推水印规律，也无法通过梯度攻击来擦除它。

3. 实验装置 (The Test Suite)

看起来，花哨的 Chernoff 信息估算效果并不一定好。我们需要一个"压力测试"。

装置组件：

1. **Base Model:** LLaMA-MoE-7B (或任何开源 MoE)。
2. **Dataset:** HumanEval (代码生成) + WikiText (通用文本)。
   - *Linus 注：* 必须包含代码生成。代码逻辑对专家路由极其敏感，如果你的水印坏了，代码跑不通是最好的"金丝雀"报警。
3. **Attack Tool:** GPT-3.5 Turbo (Paraphraser) 。

实验脚本 (伪代码逻辑)：

```
# 配置
EPSILON = 0.5  # 质量容忍度，需标定
KEY = "LinuxIsTheBestKernel"

def run_benchmark(model_mode):
    """
    model_mode: 'original' or 'watermarked_okr'
    """
    results = []

    # 1. 质量测试 (User Space Integrity)
    # 跑 HumanEval
    pass_at_1 = evaluate_code_generation(model_mode)
```

```
print(f"[{model_mode}] Pass@1: {pass_at_1}")
# 如果 watermarked 相比 original 下降超过 1%，说明 EPSILON 太大了，F

# 2. 鲁棒性测试 (Stress Test)
input_text = load_wikitext()
output_text = generate(model_mode, input_text)

# 实施攻击
attacked_text = gpt35_rewrite(output_text) # 模拟释义攻击 [cite: 47

# 检测水印
score = detect_watermark(attacked_text, KEY, EPSILON)

print(f"[{model_mode}] Watermark Score after Attack: {score}")
# 预期：Original < 0.5, Watermarked > 0.95
```

预期结果分析：

1. **质量 (Pass@1):** 曲线应该非常平稳。因为 OKR 只有在模型"犹豫不决"时才介入，而处理复杂的代码逻辑关键点时，模型通常非常自信（$|S| = 1$），水印会自动退让。
2. **鲁棒性 (Score):** 即使经过 GPT-3.5 的重写（攻击强度 $\gamma \approx 0.03$），Score 应该保持在 0.9 以上。因为攻击者很难通过改写句子来改变底层的语义哈希，也很难把所有候选专家都踢出 ϵ 范围。

总结

这就是你要的方案。

它不依赖脆弱的 L_g 假设，也不需要计算复杂的积分。

它把统计学上的赌博（希望信号没被淹没）变成了密码学上的确信（只要语义在，Key 在，信号就在）。