

基于 MoE 的冗余性（Top-K 互换不影响输出）以及信号衰减问题，下面是 OKR (Opportunistic Keyed Routing) v2.0 的最终修订版。

RFC: 机会主义加密路由 (OKR) - 架构规范 v2.0

核心理念：利用专家冗余作为隐写带宽，利用密码学哈希作为路由指令。

原则：

1. Never Break Userspace: 只有在不影响输出质量时才介入。
2. Fail-Open: 如果没有安全的嵌入机会，就放弃嵌入，而不是制造垃圾输出。

1. 数据结构与定义

1.1 质量容忍阈值 (EPSILON)

这是一个工程常数，不是统计变量。

- 定义：EPSILON 是 Logit 值的最大差值，在这个差值内，我们认为两个专家的能力是“无差别”的。
- 来源：这利用了论文中提到的“排名间隔”(Ranking Gap) 概念。当 $Gap < \sigma$ 时，专家选择本质上是随机的。我们正是要利用这个随机区。

1.2 安全候选集 (SafeSet)

对于每一个 Token 生成步 t ，计算原始 Logits。

$$S_t = \{e_i \mid \text{logit}(e_i) \geq \text{logit}(e_{best}) - \text{EPSILON}\}$$

- **关键逻辑：**只有当 $|S_t| \geq 2$ 时，我们才拥有“可写权限”。这就是你发现的“第 K 与 K+1 个专家互换无影响”的情况。

1.3 语义锚点 (SemanticAnchor)

为了抵抗论文中提到的释义攻击 (Paraphrase Attack) 导致的 Z_{score} 衰减，我们不能对 Token 进行哈希 (Token 会变)。

- 我们对**当前的语义向量** (Embedding 层的均值或特定层的输出) 进行哈希。
- 论文承认，即使在攻击下，语义也是保持稳定的 ($Meaning(x) \approx Meaning(x')$)。

2. 核心逻辑 (The Kernel Hook)

这是插入在 MoE Gating Layer 的伪代码逻辑：

```
def opportunistic_route(context_embedding, raw_logits, secret_key, epsilon):
    """
    Route input to experts, embedding watermark ONLY when safe.
    """

    # 1. 找出最佳专家 (The Gold Standard)
    best_expert_idx = argmax(raw_logits)
    max_logit = raw_logits[best_expert_idx]

    # 2. 构建安全集 (The Indifference Zone)
    # 只有在这里面的专家，互换才不会影响输出质量
    safe_candidates = [
        idx for idx, val in enumerate(raw_logits)
        if val >= (max_logit - epsilon)
    ]

    # 3. 机会主义判定 (Fail-Open Decision)
    if len(safe_candidates) < 2:
        # 没有冗余，强行修改会破坏 Userspace。
        # 放弃水印，保持原样。
        return best_expert_idx

    # 4. 加密路由 (Cryptographic Injection)
    # 既然选谁都一样，那就选 Key 想要的那一个
    # 使用语义特征而非 Token，抵抗释义攻击
    signature = HMAC(secret_key, context_embedding)

    # 确定性地从安全集中选一个
    target_index_in_set = int(signature) % len(safe_candidates)
    selected_expert = safe_candidates[target_index_in_set]
```

```
return selected_expert
```

3. 验证机制 (The Verification)

验证者不需要知道模型原本想选谁，也不需要做复杂的假设检验（如论文中的 Neyman-Pearson）。验证者只需要检查：“**每一次发生专家选择分歧时，实际选择是否符合密钥的指引？**”

1. **重放推理：** 使用相同的模型和 `EPSILON` 对文本进行推理。

2. **识别机会点：** 记录所有 $|S_t| \geq 2$ 的时刻。

3. **比对签名：**

- 计算 `HMAC(Key, Embedding)` 应该指向谁。
- 检查实际激活的专家是否就是那一个。

4. **最终得分：**

$$Score = \frac{\text{Matches}}{\text{Opportunities}}$$

- 阈值：如果 $Score > 0.9$ ，判定为有水印（因为随机猜对的概率极低）。

4. 针对你提出的“两个情况”的解决方案

- 针对“同一 Query 不同输出”：OKR 是 **路径依赖Path Dependent** 的。如果 Temperature 导致生成路径分叉，没关系。路径 A 有路径 A 的语义流，路径 B 有路径 B 的语义流。

- 水印不是打在 Query 上的，是打在**生成过程中的每一步路由决策上**的。只要输出文本是模型生成的，它的内部一致性（路由与语义的锁定关系）就永远存在。

- 针对“互换专家无影响”：这是本方案的燃料。

- 我们将这种情况量化为 $|S_t| \geq 2$ 。

- 正是因为互换无影响，我们才能肆无忌惮地根据 HMAC 强制指定专家，而不用担心性能回退（Regression）。

5. 基础实验装置 (The Benchmarking Suite)

别去算数学了，跑代码。

目标：证明 OKR 既不降低质量（利用了冗余），又抗攻击（利用了语义锚点）。

步骤 A：校准 (Calibration)

1. **数据：** WikiText-103 验证集。
2. **动作：** 扫描 EPSILON 从 0.1 到 2.0。
3. **指标：** 统计“平均机会率”（每个句子有多少 Token 拥有 $|S| \geq 2$ ）。
 - **预期：** 我们需要至少 10-15% 的 Token 拥有“互换空间”作为水印带宽。找到能提供这个带宽的最小 EPSILON 。

步骤 B：质量回归测试 (Regression Test)

1. **数据：** HumanEval (代码生成)。这是最敏感的测试。
2. **对比：** 原始模型 vs. OKR 模型 (使用校准后的 EPSILON)。
3. **Pass 条件：** Pass@1 分数下降幅度必须 $< 0.5\%$ 。如果代码跑不通，说明你的 EPSILON 太大了，把不该进来的专家放进来了。

步骤 C：攻击与检测 (Penetration Test)

1. **攻击：** 使用 GPT-4 或 T5 对带水印文本进行释义攻击（Paraphrase）。
2. **检测：**
 - 提取被攻击文本的 Embedding。
 - 重跑检测逻辑，计算 Score。
 - **预期：** 即使 Token 变了，Embedding 依然相似。HMAC 指向的目标依然在安全集 S_t 中（由于安全集的迟滞效应）。Score 应保持在

0.9 以上。

总结

这版方案不仅解决了**信号衰减**（通过语义锚点和安全集迟滞），还完美利用了**专家冗余**（作为隐写通道）。