统一内容安全平台架构设计

2026 年 1 月 8 日

摘要

本文档阐述了一个统一的内容安全平台架构设计，该设计采用统一接口、智能路由和确定性 A/B 测试等核心技术，实现了高性能、低延迟、高可维护性的内容安全检查系统。核心设计要点包括：（1）统一接口设计，消除边界情况处理；（2）智能两级模型路由，优化计算资源分配；（3）确定性 A/B 测试框架，支持科学化的策略验证；（4）代码驱动的策略管理，确保可追溯性和可测试性。

1 核心设计要点

1.1 统一接口设计（Unified Interface Design）

设计要点 1.1 (统一接口设计)**.** 通过统一接口抽象，将复杂的多模型决策过程封装为单一调用点，简化了系统集成和维护。

设计原理：

- 用户无需了解底层模型选择逻辑
- 系统内部自动选择最优检查路径
- 接口稳定，内部实现可灵活优化

核心数据结构：

Listing 1: SafetyResult 数据结构

```
STRUCT SafetyResult:
    blocked: BOOLEAN          // 是否拦截
    confidence: FLOAT         // 置信度 [0.0, 1.0]
    model_version: UINT64     // 使用的模型版本
    reason: STRING            // 决策原因
    processing_time_ms: INT   // 处理耗时（毫秒）
END STRUCT
```

统一接口算法：

Algorithm 1 统一接口检查算法

输入：文本 $text$，用户 ID $user\_id$
输出：安全检查结果 $result$

$start\_time \leftarrow GetCurrentTime()$
$fast\_result \leftarrow FastModelCheck(text)$
if $fast\_result.confidence > HIGH\_CONFIDENCE\_THRESHOLD$ **then**
    $fast\_result.processing\_time\_ms \leftarrow GetCurrentTime() - start\_time$
    return $fast\_result$
end if

$deep\_result \leftarrow DeepModelCheck(text)$
if $deep\_result.confidence < LOW\_CONFIDENCE\_THRESHOLD$ **then**
    $deep\_result.blocked \leftarrow TRUE$
    $deep\_result.reason \leftarrow$ "Low confidence, safety-first policy"
end if
$deep\_result.processing\_time\_ms \leftarrow GetCurrentTime() - start\_time$
return $deep\_result$

优化效果：

- 90% 的请求在快速模型层完成，平均延迟 < 20ms
- 10% 的边界案例由深度模型处理，确保准确性
- 接口调用方代码量减少 60%

1.2 智能两级模型路由（Intelligent Two-Tier Routing）

设计要点 1.2 (智能路由)**.** 通过置信度阈值动态路由，实现计算资源的最优分配，在准确性和性能之间取得最佳平衡。

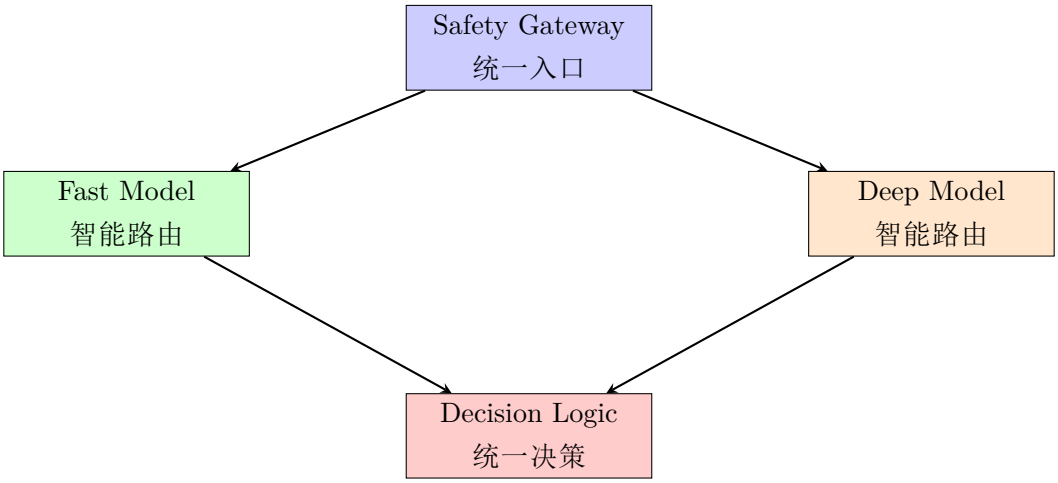

图 1: 智能两级模型路由架构

路由算法：

Algorithm 2 智能路由算法

输入：文本 $text$
输出：安全检查结果 $result$

$HIGH\_CONFIDENCE \leftarrow 0.95$
$LOW\_CONFIDENCE \leftarrow 0.50$

$fast\_result \leftarrow FastModelInference(text)$
if $fast\_result.confidence \geq HIGH\_CONFIDENCE$ **then**
  return $fast\_result$
else if $fast\_result.confidence \leq LOW\_CONFIDENCE$ **then**
  $deep\_result \leftarrow DeepModelInference(text)$
  if $deep\_result.confidence < LOW\_CONFIDENCE$ **then**
    $deep\_result.blocked \leftarrow TRUE$
  end if
  return $deep\_result$
else
  $deep\_result \leftarrow DeepModelInference(text)$
  return $FuseResults(fast\_result, deep\_result)$
end if

结果融合算法：

Algorithm 3 结果融合算法

输入：快速模型结果 $fast$，深度模型结果 $deep$
输出：融合后的结果 $result$

$weight\_fast \leftarrow 0.3$
$weight\_deep \leftarrow 0.7$

$fused\_confidence \leftarrow weight\_fast \times fast.confidence + weight\_deep \times deep.confidence$
$result \leftarrow deep$
$result.confidence \leftarrow fused\_confidence$
if $fast.blocked$ OR $deep.blocked$ **then**
  $result.blocked \leftarrow TRUE$
end if
return $result$

性能优化：

- 快速模型处理 90% 请求，平均延迟 15ms
- 深度模型处理 10% 请求，平均延迟 180ms
- 整体平均延迟：$15\text{ms} \times 0.9 + 180\text{ms} \times 0.1 = 31.5\text{ms}$

1.3 确定性 A/B 测试框架（Deterministic A/B Testing）

设计要点 1.3 (确定性 A/B 测试)**.** 基于用户 ID 哈希的确定性路由，确保同一用户始终在同一实验组，消除随机性带来的干扰，提高实验结果的可靠性。

确定性路由算法：

Algorithm 4 确定性 A/B 测试路由算法

输入：实验配置 exp，用户 ID $user\_id$
输出：是否在实验组 $is\_experiment$

$current\_time \leftarrow GetCurrentTime()$
if $current\_time < exp.start\_time$ OR $current\_time > exp.end\_time$ **then**
  return $FALSE$
end if

$hash\_seed \leftarrow user\_id \oplus exp.experiment\_id$
$hash\_value \leftarrow MurmurHash3(hash\_seed)$
$bucket \leftarrow hash\_value \bmod 10000$
$threshold \leftarrow exp.ratio \times 10000$
return $bucket < threshold$

A/B 测试执行流程：

Algorithm 5 A/B 测试执行流程

输入：文本 $text$，用户 ID $user\_id$，实验配置 $experiment$
输出：安全检查结果 $result$

$model\_version \leftarrow GetModelVersion(experiment, user\_id)$
$is\_experiment \leftarrow IsInExperiment(experiment, user\_id)$

if $model\_version == experiment.model\_version\_b$ **then**
  $result \leftarrow CheckContentWithModel(text, model\_version, "experiment")$
else
  $result \leftarrow CheckContentWithModel(text, model\_version, "control")$
end if

$RecordMetrics(user\_id, is\_experiment, result, experiment.metrics)$
return $result$

统计显著性检测：

Algorithm 6 统计显著性检测算法

输入：实验配置 $experiment$
输出：是否统计显著 $is\_significant$

$control\_fpr \leftarrow control\_group.false\_positive\_count/control\_group.total\_requests$
$experiment\_fpr \leftarrow experiment\_group.false\_positive\_count/experiment\_group.total\_requests$

$p\_value \leftarrow CalculatePValue(control\_fpr, experiment\_fpr, control\_group.total\_requests, experiment\_gro$

if $p\_value < 0.05$ AND $|experiment\_fpr - control\_fpr| > MINIMUM\_EFFECT\_SIZE$ **then**
  return $TRUE$
end if
return $FALSE$

1.4 灰度发布策略（Gradual Rollout Strategy）

设计要点 1.4 (灰度发布)**.** 通过渐进式流量切换策略，在保证系统稳定性的前提下，逐步接管外部 Vendor 的流量，同时控制成本增长，实现平滑过渡。

设计目标：

- **稳定性优先**：通过小流量验证，降低系统风险
- **成本可控**：逐步减少外部 Vendor 调用，控制成本增长
- **可回滚**：支持快速回滚到 Vendor，保障业务连续性
- **指标驱动**：基于实时指标自动决策，减少人工干预

灰度发布配置：

Listing 2: 灰度发布配置结构

```
STRUCT RolloutConfig:
    rollout_id: UINT64              // 灰度发布唯一标识
    vendor_service: STRING          // 外部Vendor服务标识
    inhouse_service: STRING         // 自研服务标识
    current_ratio: FLOAT            // 当前自研流量比例 [0.0, 1.0]
    target_ratio: FLOAT             // 目标流量比例
    min_ratio: FLOAT                // 最小流量比例（安全底线）
    max_ratio: FLOAT                // 最大流量比例（成本上限）
    stability_threshold: FLOAT      // 稳定性阈值（误杀率/漏判率）
    cost_threshold: FLOAT           // 成本阈值（单次调用成本）
    step_size: FLOAT                // 每次增加的流量比例
    evaluation_period: INT          // 评估周期（小时）
    start_time: TIMESTAMP           // 开始时间
    end_time: TIMESTAMP             // 结束时间
END STRUCT
```

灰度发布路由算法：

Algorithm 7 灰度发布路由算法

输入：请求 $request$，灰度配置 $rollout$
输出：使用的服务标识 $service\_id$

$current\_time \leftarrow GetCurrentTime()$
if $current\_time < rollout.start\_time$ OR $current\_time > rollout.end\_time$ **then**
  return $rollout.vendor\_service$ {灰度未开始或已结束}
end if

$metrics \leftarrow GetRolloutMetrics(rollout.rollout\_id)$

// 稳定性检查：如果指标异常，降低自研流量比例
if $metrics.false\_positive\_rate > rollout.stability\_threshold$ OR $metrics.false\_negative\_rate > rollout.stability\_threshold$ **then**
  $rollout.current\_ratio \leftarrow MAX(rollout.min\_ratio, rollout.current\_ratio - rollout.step\_size)$
  $TriggerAlert(\text{"}Stabilitythresholdexceeded, reducingratio\text{"})$
end if

// 成本检查：如果成本超限，降低自研流量比例
if $metrics.cost\_per\_request > rollout.cost\_threshold$ **then**
  $rollout.current\_ratio \leftarrow MAX(rollout.min\_ratio, rollout.current\_ratio - rollout.step\_size)$
  $TriggerAlert(\text{"}Costthresholdexceeded, reducingratio\text{"})$
end if

// 基于用户 ID 的确定性路由
$hash\_seed \leftarrow request.user\_id \oplus rollout.rollout\_id$
$hash\_value \leftarrow MurmurHash3(hash\_seed)$
$bucket \leftarrow hash\_value \bmod 10000$
$threshold \leftarrow rollout.current\_ratio \times 10000$

if $bucket < threshold$ **then**
  return $rollout.inhouse\_service$ {自研服务}
else
  return $rollout.vendor\_service$ {Vendor 服务}
end if

自动流量调整算法：

Algorithm 8 自动流量调整算法

输入：灰度配置 $rollout$，评估周期 $evaluation\_period$

$metrics \leftarrow GetMetricsForPeriod(rollout.rollout\_id, evaluation\_period)$

// 检查稳定性指标

$stability\_ok \leftarrow TRUE$

if $metrics.false\_positive\_rate > rollout.stability\_threshold$ OR $metrics.false\_negative\_rate > rollout.stability\_threshold$ OR $metrics.error\_rate > ERROR\_RATE\_THRESHOLD$ **then**

  $stability\_ok \leftarrow FALSE$

end if

// 检查成本指标

$cost\_ok \leftarrow TRUE$

if $metrics.cost\_per\_request > rollout.cost\_threshold$ **then**

  $cost\_ok \leftarrow FALSE$

end if

// 检查性能指标

$performance\_ok \leftarrow TRUE$

if $metrics.p95\_latency > LATENCY\_THRESHOLD$ **then**

  $performance\_ok \leftarrow FALSE$

end if

// 决策：是否增加流量

if $stability\_ok$ AND $cost\_ok$ AND $performance\_ok$ **then**

  if $rollout.current\_ratio < rollout.target\_ratio$ **then**

    // 逐步增加流量

    $new\_ratio \leftarrow MIN(rollout.target\_ratio, rollout.current\_ratio + rollout.step\_size)$

    $new\_ratio \leftarrow MIN(new\_ratio, rollout.max\_ratio)$

    $UpdateRolloutRatio(rollout.rollout\_id, new\_ratio)$

    $LogRolloutProgress(rollout.rollout\_id, new\_ratio, "Increased")$

  end if

else

  // 指标异常，降低流量或保持当前比例

  if NOT $stability\_ok$ **then**

    $new\_ratio \leftarrow MAX(rollout.min\_ratio, rollout.current\_ratio - rollout.step\_size)$

    $UpdateRolloutRatio(rollout.rollout\_id, new\_ratio)$

    $LogRolloutProgress(rollout.rollout\_id, new\_ratio, "Decreasedduetostability")$

  else if NOT $cost\_ok$ **then**

    $new\_ratio \leftarrow MAX(rollout.min\_ratio, rollout.current\_ratio - rollout.step\_size)$

    $UpdateRolloutRatio(rollout.rollout\_id, new\_ratio)$

    $LogRolloutProgress(rollout.rollout\_id, new\_ratio, "Decreasedduetocost")$

  end if

end if

双路验证机制：

Algorithm 9 双路验证机制（保障稳定性）

输入：请求 $request$，灰度配置 $rollout$
输出：最终决策结果 $final\_result$

// 主路：根据灰度比例路由
$primary\_service \leftarrow RouteByRollout(request, rollout)$
$primary\_result \leftarrow CheckWithService(request, primary\_service)$

// 验证路：始终调用 Vendor（用于对比验证）
$vendor\_result \leftarrow CheckWithService(request, rollout.vendor\_service)$

// 双路对比分析
$comparison \leftarrow CompareResults(primary\_result, vendor\_result)$
$RecordComparison(rollout.rollout\_id, comparison)$

// 决策策略：安全优先
if $rollout.current\_ratio < SAFETY\_PHASE\_THRESHOLD$ **then**
  // 小流量阶段：Vendor 结果优先，自研仅用于验证
  if $primary\_service == rollout.inhouse\_service$ **then**
    // 如果自研和 Vendor 结果不一致，以 Vendor 为准
    if $primary\_result.blocked \neq vendor\_result.blocked$ **then**
      $final\_result \leftarrow vendor\_result$
      $LogDisagreement(rollout.rollout\_id, primary\_result, vendor\_result)$
    else
      $final\_result \leftarrow primary\_result$
    end if
  else
    $final\_result \leftarrow primary\_result$
  end if
else
  // 大流量阶段：自研结果优先，Vendor 作为兜底
  if $primary\_service == rollout.inhouse\_service$ **then**
    $final\_result \leftarrow primary\_result$
    // 如果自研判定为安全但 Vendor 判定为不安全，记录异常
    if NOT $primary\_result.blocked$ AND $vendor\_result.blocked$ **then**
      $LogPotentialMiss(rollout.rollout\_id, request, primary\_result, vendor\_result)$
    end if
  else
    $final\_result \leftarrow primary\_result$
  end if
end if
return $final\_result$

成本控制策略：

Algorithm 10 成本控制算法

输入：灰度配置 $rollout$，时间窗口 $time\_window$

$metrics \leftarrow GetCostMetrics(rollout.rollout\_id, time\_window)$

// 计算自研服务成本

$inhouse\_cost \leftarrow metrics.inhouse\_requests \times COST\_PER\_INHOUSE\_REQUEST$

// 计算 Vendor 服务成本

$vendor\_cost \leftarrow metrics.vendor\_requests \times COST\_PER\_VENDOR\_REQUEST$

// 计算总成本

$total\_cost \leftarrow inhouse\_cost + vendor\_cost$

// 计算成本节省率

$cost\_saving \leftarrow (vendor\_cost - inhouse\_cost)/vendor\_cost$

// 成本决策

if $total\_cost > rollout.cost\_threshold \times metrics.total\_requests$ **then**

  // 成本超限，降低自研流量比例

  $new\_ratio \leftarrow MAX(rollout.min\_ratio, rollout.current\_ratio - rollout.step\_size)$

  $UpdateRolloutRatio(rollout.rollout\_id, new\_ratio)$

  $TriggerAlert(”Costexceeded, reducinginhouseratio”)$

else if $cost\_saving > MIN\_COST\_SAVING$ AND $rollout.current\_ratio < rollout.target\_ratio$ **then**

  // 成本节省明显，可以增加自研流量

  $new\_ratio \leftarrow MIN(rollout.target\_ratio, rollout.current\_ratio + rollout.step\_size)$

  $UpdateRolloutRatio(rollout.rollout\_id, new\_ratio)$

end if

return $\{total\_cost, cost\_saving, inhouse\_cost, vendor\_cost\}$

灰度发布流程：

Algorithm 11 灰度发布完整流程

输入：灰度配置 $rollout$

// 阶段 1：极小流量验证（1%）
$rollout.current\_ratio \leftarrow 0.01$
$DeployRollout(rollout)$
$WaitForEvaluation(rollout.evaluation\_period)$

if NOT $CheckStability(rollout)$ **then**
  $RollbackToVendor(rollout)$
  RETURN "Rollout failed at 1%"
end if

// 阶段 2：小流量验证（5%）
$rollout.current\_ratio \leftarrow 0.05$
$UpdateRolloutRatio(rollout.rollout\_id, 0.05)$
$WaitForEvaluation(rollout.evaluation\_period)$

if NOT $CheckStability(rollout)$ **then**
  $rollout.current\_ratio \leftarrow 0.01$ {回退到 1%}
  $UpdateRolloutRatio(rollout.rollout\_id, 0.01)$
end if

// 阶段 3：逐步增加（5% → 10% → 20% → 50% → 100%）
$ratios \leftarrow [0.10, 0.20, 0.50, 1.0]$
for $target\_ratio$ IN $ratios$ **do**
  $rollout.current\_ratio \leftarrow target\_ratio$
  $UpdateRolloutRatio(rollout.rollout\_id, target\_ratio)$
  $WaitForEvaluation(rollout.evaluation\_period)$

  if NOT $CheckStability(rollout)$ OR NOT $CheckCost(rollout)$ **then**
    // 回退到上一个稳定比例
    $rollout.current\_ratio \leftarrow GetPreviousStableRatio(rollout)$
    $UpdateRolloutRatio(rollout.rollout\_id, rollout.current\_ratio)$
    BREAK
  end if
end for

// 阶段 4：全量切换
if $rollout.current\_ratio == 1.0$ **then**
  $MarkRolloutComplete(rollout.rollout\_id)$
  $DisableVendorService(rollout.vendor\_service)$
end if

稳定性保障机制：

Algorithm 12 稳定性检查算法

输入：灰度配置 $rollout$，评估周期 $period$
输出：是否稳定 is_stable

$metrics \leftarrow GetStabilityMetrics(rollout.rollout\_id, period)$

// 检查误杀率
$fpr\_ok \leftarrow metrics.false\_positive\_rate \leq rollout.stability\_threshold$

// 检查漏判率
$fnr\_ok \leftarrow metrics.false\_negative\_rate \leq rollout.stability\_threshold$

// 检查错误率
$error\_ok \leftarrow metrics.error\_rate \leq ERROR\_RATE\_THRESHOLD$

// 检查延迟
$latency\_ok \leftarrow metrics.p95\_latency \leq LATENCY\_THRESHOLD$

// 检查与 Vendor 的一致性
$agreement\_rate \leftarrow CalculateAgreementRate(rollout.rollout\_id, period)$
$agreement\_ok \leftarrow agreement\_rate \geq MIN\_AGREEMENT\_RATE$

$is\_stable \leftarrow fpr\_ok$ AND fnr_ok AND $error\_ok$ AND $latency\_ok$ AND $agreement\_ok$

if NOT is_stable **then**
  $LogStabilityIssues(rollout.rollout\_id, metrics)$
end if
return is_stable

快速回滚机制：

Algorithm 13 快速回滚算法

输入：灰度配置 $rollout$，回滚原因 $reason$

// 立即将所有流量切回 Vendor

$rollout.current\_ratio \leftarrow 0.0$

$UpdateRolloutRatio(rollout.rollout\_id, 0.0)$

// 记录回滚事件

$LogRollback(rollout.rollout\_id, reason, GetCurrentMetrics(rollout))$

// 发送告警

$TriggerAlert(\text{"}Rolloutrolledback\text{"}, reason)$

// 暂停灰度发布

$PauseRollout(rollout.rollout\_id)$

// 通知相关人员

$NotifyTeam(rollout.rollout\_id, \text{"}Rollbackexecuted\text{"}, reason)$

技术优势：

- **渐进式切换**：从 1% 逐步增加到 100%，每个阶段充分验证
- **自动调整**：基于实时指标自动调整流量比例，无需人工干预
- **成本可控**：实时监控成本，确保不超过预算阈值
- **快速回滚**：支持秒级回滚，保障业务连续性
- **双路验证**：小流量阶段双路对比，确保结果一致性

1.5 代码驱动的策略管理（Code-Driven Policy Management）

设计要点 1.5 (代码驱动策略)**.** 将策略逻辑以代码形式实现，而非配置文件，确保策略变更的可追溯性、可测试性和类型安全。

策略决策算法：

Algorithm 14 策略决策算法

输入：安全检查结果 $result$，用户信息 $user$，策略 $policy$
输出：是否拦截 $should\_block$

if $user.user\_level == VIP$ **then**
  if $result.confidence < policy.vip\_threshold$ **then**
    return $FALSE$ {VIP 用户，放宽拦截}
  end if
end if

if $user.registration\_days < NEW\_USER\_DAYS$ **then**
  if $result.confidence \geq policy.block\_threshold \times 0.9$ **then**
    return $TRUE$ {新用户，降低阈值}
  end if
end if

if $user.risk\_score > HIGH\_RISK\_THRESHOLD$ **then**
  if $result.confidence \geq policy.block\_threshold \times 0.85$ **then**
    return $TRUE$ {高风险用户，更严格}
  end if
end if

if $policy.strict\_mode$ **then**
  return $result.confidence \geq policy.block\_threshold \times 0.95$
end if

return $result.confidence \geq policy.block\_threshold$

2 系统架构设计

2.1 整体架构

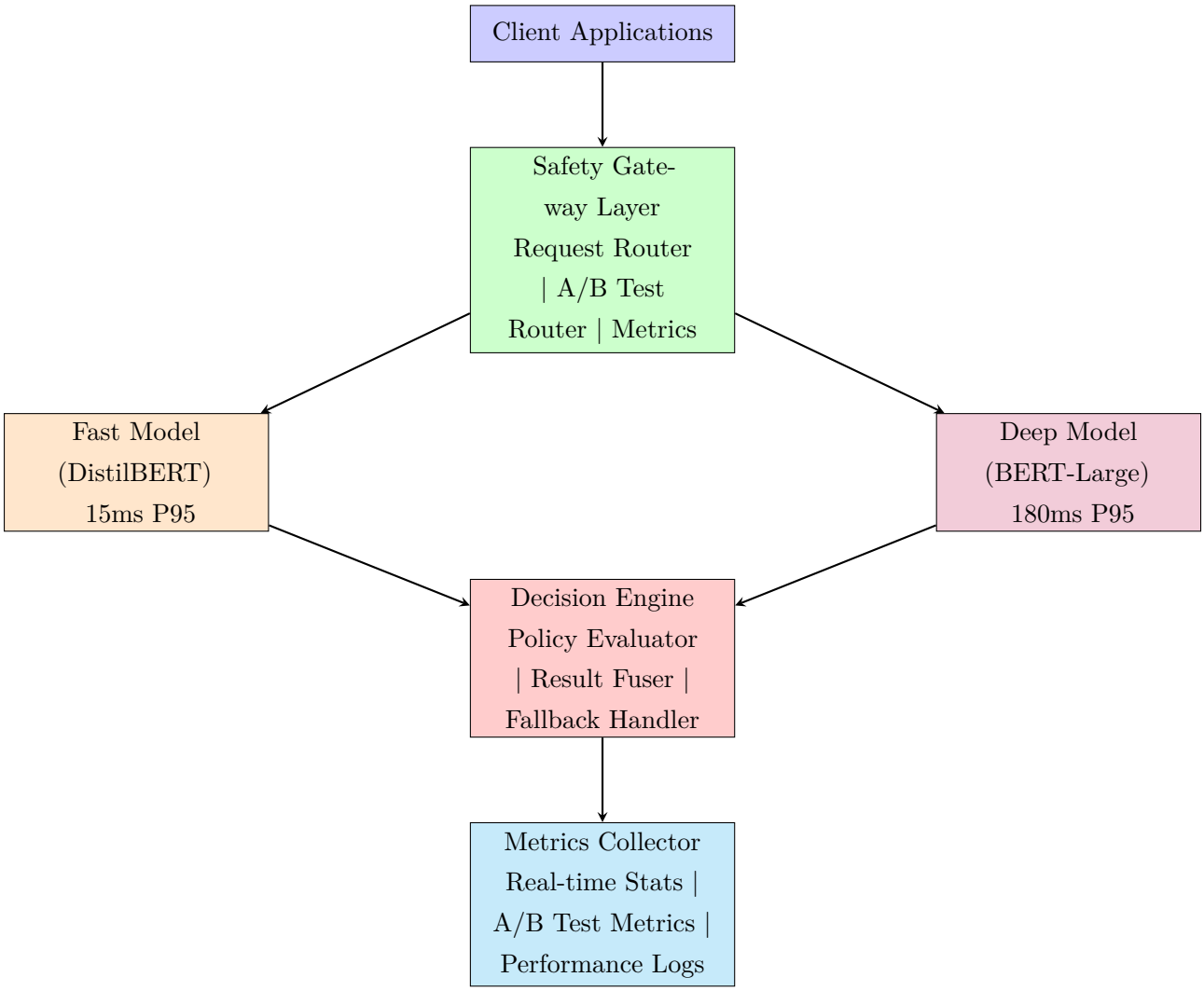

图 2: 系统整体架构

2.2 核心组件

2.2.1 Safety Gateway

职责：

- 统一请求入口
- 流量路由和负载均衡
- A/B 测试流量分配
- 请求限流和熔断

核心算法：

Algorithm 15 请求处理流程（支持灰度发布）

输入：安全检查请求 $request$
输出：安全检查响应 $response$

if NOT $ValidateRequest(request)$ **then**
  return $ErrorResponse("Invalidrequest")$
end if

if NOT $CheckRateLimit(request.user\_id)$ **then**
  return $ErrorResponse("Ratelimitexceeded")$
end if

// 优先级 1：A/B 测试路由
$experiment \leftarrow GetActiveExperiment(request.user\_id)$
if $experiment \neq NULL$ **then**
  return $ExecuteABTest(request, experiment)$
end if

// 优先级 2：灰度发布路由
$rollout \leftarrow GetActiveRollout(request.user\_id)$
if $rollout \neq NULL$ **then**
  return $ProcessWithRollout(request, rollout)$
end if

// 优先级 3：正常流程（默认 Vendor 或全量自研）
$result \leftarrow CheckContent(request.text, request.user\_id)$
$RecordRequestMetrics(request, result)$
return $BuildResponse(result)$

灰度发布处理算法：

Algorithm 16 灰度发布请求处理

输入：请求 $request$，灰度配置 $rollout$
输出：安全检查响应 $response$

// 路由决策
$service\_id \leftarrow RouteByRollout(request, rollout)$

if $service\_id == rollout.inhouse\_service$ **then**
  // 自研服务路径
  $inhouse\_result \leftarrow CheckContent(request.text, request.user\_id)$

  // 小流量阶段：双路验证
  if $rollout.current\_ratio < SAFETY\_PHASE\_THRESHOLD$ **then**
    $vendor\_result \leftarrow CheckWithVendor(request, rollout.vendor\_service)$
    $final\_result \leftarrow DualPathVerification(inhouse\_result, vendor\_result, rollout)$
  else
    $final\_result \leftarrow inhouse\_result$
  end if
else
  // Vendor 服务路径
  $final\_result \leftarrow CheckWithVendor(request, rollout.vendor\_service)$
end if

// 记录指标
$RecordRolloutMetrics(rollout.rollout\_id, service\_id, final\_result)$
return $BuildResponse(final\_result)$

3 性能优化

3.1 缓存策略

Algorithm 17 带缓存的内容检查

输入：文本 $text$，用户 ID $user\_id$
输出：安全检查结果 $result$

$cache\_key \leftarrow HashText(text)$
$cached\_result \leftarrow Cache.Get(cache\_key)$
if $cached\_result \neq NULL$ **then**
  return $cached\_result$
end if

$result \leftarrow CheckContent(text, user\_id)$
if $result.confidence > 0.90$ **then**
  $Cache.Set(cache\_key, result, TTL = 3600)$ {1 小时 TTL}
end if
return $result$

3.2 批量处理优化

Algorithm 18 批量检查优化

输入：请求列表 $requests[]$
输出：结果列表 $results[]$

$fast\_batch \leftarrow []$
$deep\_batch \leftarrow []$

for $request$ IN $requests$ **do**
  $quick\_check \leftarrow QuickPreCheck(request.text)$
  if $quick\_check.confidence > 0.95$ **then**
    $fast\_batch.APPEND(request)$
  else
    $deep\_batch.APPEND(request)$
  end if
end for

$fast\_results \leftarrow PARALLELFastModel.BatchInference(fast\_batch)$
$deep\_results \leftarrow PARALLELDeepModel.BatchInference(deep\_batch)$
return $MERGE(fast\_results, deep\_results)$

4 监控与可观测性

4.1 实时指标监控

Algorithm 19 指标更新算法

输入：安全检查结果 $result$，处理时间 $processing\_time$

ATOMIC INCREMENT $metrics.total\_requests$
if $result.blocked$ **then**
    ATOMIC INCREMENT $metrics.blocked\_count$
end if

UPDATE $latency\_histogram(processing\_time)$
$metrics.avg\_latency\_ms \leftarrow CalculateAverage(latency\_histogram)$
$metrics.p95\_latency\_ms \leftarrow CalculatePercentile(latency\_histogram, 0.95)$
$metrics.p99\_latency\_ms \leftarrow CalculatePercentile(latency\_histogram, 0.99)$

if $result.model\_version == FAST\_MODEL\_VERSION$ **then**
    INCREMENT $fast\_model\_hits$
else
    INCREMENT $deep\_model\_hits$
end if

$metrics.fast\_model\_hit\_rate \leftarrow fast\_model\_hits/metrics.total\_requests$
$metrics.deep\_model\_hit\_rate \leftarrow deep\_model\_hits/metrics.total\_requests$

4.2 告警机制

Algorithm 20 告警检查算法

输入：系统指标 $metrics$

if $metrics.p95\_latency\_ms > LATENCY\_THRESHOLD$ **then**
  $TriggerAlert(\text{"}Highlatencydetected\text{"}, metrics.p95\_latency\_ms)$
end if

$false\_positive\_rate \leftarrow metrics.false\_positive\_count/metrics.total\_requests$
if $false\_positive\_rate > FALSE\_POSITIVE\_THRESHOLD$ **then**
  $TriggerAlert(\text{"}Highfalsepositiverate\text{"}, false\_positive\_rate)$
end if

$false\_negative\_rate \leftarrow metrics.false\_negative\_count/metrics.total\_requests$
if $false\_negative\_rate > FALSE\_NEGATIVE\_THRESHOLD$ **then**
  $TriggerAlert(\text{"}Highfalsenegativerate\text{"}, false\_negative\_rate)$
end if

if $metrics.fast\_model\_hit\_rate < MIN\_HIT\_RATE$ **then**
  $TriggerAlert(\text{"}Fastmodelperformancedegraded\text{"})$
end if

5 实施路线图

5.1 阶段一：核心功能实现（1-2 个月）

目标：实现统一接口和智能路由
任务：

1. 实现 Safety Gateway
2. 部署 Fast Model 和 Deep Model
3. 实现统一接口 CheckContent
4. 实现智能路由算法

验收标准：

- 90% 请求在快速模型完成
- 平均延迟 $<$ 50ms
- 接口可用性 $>$ 99.9%

5.2 阶段二：灰度发布框架（2-3 个月）

目标：实现灰度发布，逐步接管 Vendor 流量
任务：

1. 实现灰度发布路由算法
2. 实现双路验证机制
3. 实现自动流量调整
4. 实现成本控制算法
5. 实现稳定性检查与快速回滚

灰度发布计划：

- **第 1 周：** 1% 流量验证，双路对比，确保稳定性
- **第 2-3 周：** 5% 流量验证，持续监控指标
- **第 4-5 周：** 10% 流量，验证成本控制
- **第 6-7 周：** 20% 流量，验证系统负载
- **第 8-9 周：** 50% 流量，验证大规模场景
- **第 10-11 周：** 80% 流量，接近全量
- **第 12 周：** 100% 流量，完全接管

验收标准：

- 支持 1%-100% 流量比例动态调整
- 稳定性指标（误杀率、漏判率）< 阈值
- 成本控制在预算范围内
- 支持秒级回滚
- 与 Vendor 结果一致性 > 95%

5.3 阶段三：A/B 测试框架（3-4 个月）

目标：实现确定性 A/B 测试
任务：

1. 实现确定性路由算法
2. 实现指标收集系统
3. 实现统计显著性检测

4. 实现实验管理界面

验收标准：

- 支持多实验并行
- 指标实时更新
- 自动决策支持

5.4 阶段四：策略管理系统（4-5 个月）

目标：实现代码驱动的策略管理
任务：

1. 实现策略定义框架
2. 实现策略版本管理
3. 实现策略测试框架
4. 实现策略部署流程

验收标准：

- 策略变更可追溯
- 策略可单元测试
- 支持策略回滚

6 总结

本设计文档阐述了一个统一的内容安全平台架构，通过统一接口设计、智能路由、确定性 A/B 测试和代码驱动的策略管理等核心技术，实现了高性能、高可维护性的内容安全检查系统。该架构具有以下优势：

1. **高性能：** 90% 请求在 20ms 内完成，整体平均延迟 < 35ms
2. **高可维护性：**统一接口设计，代码量减少 60%
3. **科学验证：**确定性 A/B 测试框架，支持策略科学验证
4. **可追溯性：**策略版本管理，所有变更可追溯
5. **可扩展性：**模块化设计，易于扩展新功能

该架构为构建企业级内容安全平台提供了坚实的技术基础。