

PR #56122 完整报告

vllm-project/vllm

[watermarking] Dual-key gumbel-max watermarking for speculative decoding support

合并时间: 2026-09-13 02:42

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56122>

执行摘要

- 一句话: 水印兼容投机解码: 双 key Gumbel 与上下文去重
- 推荐动作: 值得精读, 尤其是三类读者: 做水印 / 内容溯源的人应关注 DualKeyGumbelWatermarker 的路由与检测器方差重标定; 做投机解码的人应关注 resample_kernel 的 WATERMARK 分支与 watermarked_rejection_sample 的 key 角色约定; 做性能的人应关注 context 去重的 kernel 代价与 watermark_sample_warmup 的 specialization 覆盖策略。建议重点看两个设计决策: 一是 Watermarker.sample 模板方法 + _try_sample_mixed 的“可选融合钩子”, 它把两趟采样优化收敛到算法内部; 二是 create_speculative*_watermarker 的“拆角色”约定, 配合 _resolve_watermark_key 的显式报错, 避免双 key 容器被误用。阅读时可跳过 tests/watermarking 的数值断言细节, 但 tests/v1/spec_decode/test_rejection_sampler_utils.py 中“水印恢复等价于 philox_gumbel_sample”的对比测试值得一看。

功能与动机

PR body 明确指出现有水印 (前置 PR #54053 的 Gumbel-max 单 key 方案) 与投机解码不兼容, 因此提出两条路径: allow_target_only_watermarking=true 允许草稿 token 不被水印 (但会按接受比例稀释水印信号), 以及新的 dual_key_gumbel 算法, 让草稿与目标各自使用独立 key 水印, 同时保持接受率不变 (引用 SynthID-Text 补充材料 algorithm 5/6 与 K.13 节)。第二条动机来自上下文去重: 默认 4 token 的短上下文在同一序列内可能重复, 导致同一位置反复受到相同水印偏置, 破坏“单序列非失真”性质, PR body 给出的实测是未去重时 35.7% 的生成撞上 1024 token 上限、4-token 上下文重复率 74.94%, 而去重后分别降到 4.0% 与 17.99%。

实现拆解

1. 配置与算法契约 (vllm/config/watermarking.py、vllm/config/vllm.py): 算法枚举扩展为 gumbel | dual_key_gumbel, 新增 alpha (默认 0.1)、deduplicate_contexts (nonelssingle_turn | all, 默认 single_turn)、deduplicate_contexts_max_history (默认 8192, 可置 None)、allow_target_only_watermarking; 新增 derive_watermark_key(key, domain) 用 SHA-256 做域分离, 从同一主 key 派生 key_a/key_b; supports_speculative_decoding 从静态字典改为按算法判断; 校验器由 validate_key 重命名为 validate_watermark_settings, 并在关闭去重或历史长度低于 _MIN_RECOMMENDED_DEDUP_HISTORY=1024 时对两种 Gumbel 算法统一发出退化生

成告警。跨配置校验补充了 dual_key 需要概率式 draft_sample_method、alpha 在 target-only 模式下未被使用等约束。

2. 采样主干抽象 (vllm/v1/watermarking/watermarker.py、gumbel.py) :

RandomSampler 从 Callable 类型别名改为携带 expanded_idx_mapping/temperatures/seeds/positions/use_fp64 的 dataclass, 其 __call__ 内部走 gumbel_sample; Watermarker.sample 变成模板方法——先尝试 _try_sample_mixed 在算法内部完成“水印 + 非水印”融合采样, 否则调用抽象方法 _sample_watermarked, 再用 skip_mask 做 torch.where 回退。原 GumbelWatermarker.sample 因此拆成 _sample_watermarked/_try_sample_mixed。新增 DualKeyGumbelWatermarker (组合两个 GumbelWatermarker, 按 alpha 用随机流在 key A/B 间路由) 与 DualKeyGumbelWatermarkDetector (两 key 分数加权早融合 + 按 variance 重标定 p 值)。

3. 投机解码接入 (vllm/v1/watermarking/spec_decode.py 新文件、

v1/worker/gpu/spec_decode/rejection_sampler_utils.py、rejection_sampler.py、speculator.py) : 新模块用 create_speculative_target_watermarker/create_speculative_draft_watermarker 把双 key 容器拆成目标侧与草稿侧; DraftWatermarker 为草稿模型维护每请求的 context 窗口与 enabled 开关; _resolve_watermark_key 显式拒绝把未拆分的容器直接用于恢复采样, 避免用错 key。rejection_sampler_utils 的 _resample_kernel 新增 WATERMARK/CONTEXT_WIDTH/contexts/watermarking/watermark_key 参数, 在 kernel 内调用 philox_gumbel_block_argmax 直接抽 key-B 的残差与 bonus token; rejection_sampler.py 增加 _watermarking_kwargs, speculator.py 增加 prepare_watermarking 喂入上下文和开关。

4. 生成期上下文去重 (vllm/v1/worker/gpu/sample/watermark.py、

vllm/v1/watermarking/gpu_sampler.py) : 新增 repeated_context_mask 入口与 _repeated_context_mask_kernel (Triton) /_repeated_context_mask_cpu (参考实现), 扫描 all_token_ids 判断当前 context 是否已经在该请求历史中出现; gpu_sampler 的 _sample_random 在 dedup 不为 none 时计算 repeated_contexts 并折叠进 skip_mask, 同时把采样状态打包成 RandomSampler 传给水印器; _get_contexts 新增 expanded_local_pos/draft_sampled 形参, 用于在投机验证的扁平化行上区分已提交 token 与草稿 token, 正确拼出每个待采样位置的 context。

5. 预热与测试 /CI 配套: 新文件 vllm/model_executor/warmup/watermark_sample_warmup.py 在启动时按 (key, logits dtype, skip_mask, USE_FP64) 组合预热

_philox_gumbel_kernel, 避免首个普通请求承担多秒编译 (jit_monitor_mode=error 下还会直接使引擎退出); spec_decode_rejection_warmup.py 扩展到 watermark 参数。测试侧新增 tests/model_executor/test_watermark_sample_warmup.py、tests/model_executor/test_spec_decode_rejection_warmup.py, 扩展 tests/watermarking/test_watermarking.py、tests/watermarking/test_gumbel.py、tests/v1/spec_decode/test_rejection_sampler_utils.py、tests/test_config.py; CI 增加 DSpark + TP2 下 dual-key 水印的 GSM8K 准确率与草稿接受率用例。

关键文件:

- vllm/v1/watermarking/spec_decode.py (模块 水印投机解码; 类别 source; 类型 dependency-wiring; 符号 DraftWatermarker, DraftWatermarker.prepare,

DraftWatermarker.sample, create_speculative_target_watermarker)：本 PR 新增的投机解码水印桥接层：定义 DraftWatermarker、目标 / 草稿水印器拆分与 watermarked_rejection_sample，是“水印 + 投机解码”能成立的关键文件。

- vllm/v1/watermarking/gumbel.py (模块 水印算法；类别 source；类型 core-logic；符号 GumbelWatermarker._sample_watermarked, GumbelWatermarker._try_sample_mixed, DualKeyGumbelWatermarker, DualKeyGumbelWatermarker._routing_logits)：新增 DualKeyGumbelWatermarker 与 DualKeyGumbelWatermarkDetector，是 dual-key 算法的生成端与检测端实现，并保留 $\alpha=0/1$ 的单 key 退化路径。
- vllm/v1/worker/gpu/sample/watermark.py (模块 水印采样内核；类别 source；类型 core-logic；符号 _repeated_context_mask_kernel, _repeated_context_mask_cpu, repeated_context_mask, _philox_context_state)：上下文去重的内核实现：新增 repeated_context_mask 与 CPU 参考实现，并抽取 philox_gumbel_block_argmax 供 rejection kernel 复用，是“检测器兼容 PRF”的共享底座。
- vllm/v1/watermarking/watermarker.py (模块 水印接口；类别 source；类型 dependency-wiring；符号 RandomSampler, RandomSampler.call, Watermarker.sample, Watermarker._sample_watermarked)：采样主干抽象的重新设计：RandomSampler 升级为 dataclass，Watermarker.sample 变成模板方法并引入 _try_sample_mixed 融合钩子与 SupportsSpeculativeDecoding 协议。
- vllm/v1/watermarking/gpu_sampler.py (模块 水印采样器；类别 source；类型 core-logic；符号 GPUWatermarkSampler._sample_random, _get_contexts, _get_repeated_contexts)：GPUWatermarkSampler 的采样入口改动：新增 _get_repeated_contexts、把 dedup 结果折叠进 skip_mask，并让 _get_contexts 支持扁平化验证行（含草稿 token）的 context 拼接。
- vllm/config/watermarking.py (模块 水印配置；类别 source；类型 configuration；符号 derive_watermark_key, WatermarkingAlgorithm, WatermarkContextScope, supports_speculative_decoding)：新算法与去重行为的配置契约：dual_key_gumbel、alpha、deduplicate_contexts(_max_history)、allow_target_only_watermarking、derive_watermark_key 与统一告警。
- vllm/v1/worker/gpu/spec_decode/rejection_sampler_utils.py (模块 拒绝采样；类别 source；类型 core-logic；符号 _seeded_resample_argmax, _resample_kernel, rejection_sample)：rejection kernel 的水mark 分支：抽出 _seeded_resample_argmax，并让残差 /bonus token 在 kernel 内直接使用 key-B Philox 噪声。
- vllm/model_executor/warmup/watermark_sample_warmup.py (模块 水印预热；类别 source；类型 data-contract；符号 _philox_key, _philox_sampler_keys, watermark_sample_warmup)：新增启动期预热模块，按 (key, dtype, skip_mask, USE_FP64) 覆盖 _philox_gumbel_kernel 的全部 specialization，消除首个请求的 JIT 编译尖峰。
- vllm/model_executor/warmup/spec_decode_rejection_warmup.py (模块 拒绝采样预热；类别 source；类型 data-contract)：把 watermark 参数 (contexts/watermarking/watermark_key) 纳入拒绝采样内核预热，否则水印路径仍会在首个请求触发编译。

- vllm/v1/worker/gpu/model_runner.py (模块 模型执行器; 类别 source; 类型 dependency-wiring) : 采样器构建处需要把 deduplicate_contexts、use_fp64_gumbel 与目标角色水印器透传下去, 是整条链路的装配点。
- vllm/v1/worker/gpu/spec_decode/speculator.py (模块 投机解码器; 类别 source; 类型 dependency-wiring; 符号 prepare_watermarking) : 新增 prepare_watermarking, 把草稿侧 context 与 enabled 在验证步前写入 DraftWatermarker, 是 draft 水印生效的衔接点。
- tests/model_executor/test_watermark_sample_warmup.py (模块 预热测试; 类别 test; 类型 test-coverage; 符号 _sampler, _worker, _capture_calls, _variants) : 用捕获调用清单的方式断言预热覆盖了全部 (key, dtype, skip_mask) 组合, 是这套 specialization 预言最强的防线。

关键符号: DraftWatermarker.sample, DraftWatermarker.prepare, create_speculative_target_watermarker, create_speculative_draft_watermarker, watermarked_rejection_sample, _resolve_watermark_key, DualKeyGumbelWatermarker.sample, DualKeyGumbelWatermarker._routing_logits, DualKeyGumbelWatermarkDetector._get_p_value, GumbelWatermarker._try_sample_mixed, Watermarker.sample, RandomSampler.call, repeated_context_mask, _repeated_context_mask_kernel, philox_gumbel_block_argmax, GPUWatermarkSampler._get_repeated_contexts, GPUWatermarkSampler._get_contexts, derive_watermark_key, validate_watermark_settings, watermark_sample_warmup, _philox_sampler_keys, _seeded_resample_argmax

评论区精华

1) 抽象层次之争 (tomasruizt vs TQCB) : tomasruizt 认为“it is not super easy to understand the relationship of different watermarking classes”, 建议去掉 WatermarkedRejectionSampler 子类、给 DualKeyGumbelWatermarker 改用组合、把 DraftWatermarker 的 contexts/enabled 抽成状态对象; TQCB 回复接受前两条, 但认为“DraftWatermarker handles quite a lot that we can't easily move to a state object, I think this will be difficult to flatten without a tonne of duplication”, 并希望保留协议边界。最终 tomasruizt 提交的 25df1e46 (删除 WatermarkedRejectionSampler, 改为向 RejectionSampler 传可选 target watermark key) 与 1153b1c0 (DualKeyGumbelWatermarker 改用组合) 被合入。2) 27B 模型检测率低的归因 (simon-veitner-redhat) : 复现后认为“the cause is how little entropy the 27B leaves in five-shot GSM8K answers, not the dual-key scheme, MTP, or the base feature”, 并指出“On ordinary prose the 27B is detected in 99–100% of completions. Not a blocker; the description and docs should say this instead of guessing at it”, 推动文档改写而不是改算法。3) 内核缺陷与性能返工 (simon-veitner-redhat) : context mask kernel 末尾多读了 BLOCK-3 个位置, compute-sanitizer 报出 47 次非法全局读, “the comparison already excluded those lanes, so results were unaffected; the read was the defect”; key-B 恢复最初用“跑完 stock sampler 再用 PyTorch 重建残差覆盖”的方式, 在 V=151936 下“made verification 4 to 6x slower than the stock sampler and allocated about 1 GiB transiently at 256 requests”, 最终改为在内核里直接抽 key-B token。4) 独立性能验证 (tomasruizt) : Qwen3.5-9B + MTP 下接受长度变化 -0.033% (p=0.82)、TPOT +1.72% (p=4.9e-8)、

TTFT -2.33%，torch profile 中“No CPU-GPU syncs found”。

- 水印类层次是否该扁平化 (design): tomasruizt 直接提交补丁 25df1e46 (删除子类, 改为向 RejectionSampler 传可选 target watermark key) 与 1153b1c0 (DualKeyGumbelWatermarker 改用显式组合), 均被合入; DraftWatermarker 与协议边界保留。
- 27B 模型检测率偏低的归因 (correctness): 不改算法, 改为在描述与文档中说明检测率依赖输出熵 (并附上 $p(x) > 0.999$ 的 token 占比等证据)。
- context mask kernel 的历史扫描越界读 (correctness): 循环起点对齐 BLOCK 边界并用统一的 in_window 谓词裁剪, 同时把 context token 改成标量加载、去掉 CONTEXT_BLOCK。
- key-B 恢复采样的性能返工 (performance): 恢复与 bonus token 的 key-B 采样全部进入 kernel, PyTorch 侧重残差的路径被移除; 测试用水印恢复与 philox_gumbel_sample 的等价性做回归保护。
- 水印 sampler 的 JIT 预热缺失 (performance): 新增 watermark_sample_warmup, 按 (key, dtype, skip_mask, USE_FP64) 预热, 并在测试中逐条断言组合集合。
- 独立性能与正确性验证 (performance): 确认水印对接受率基本无影响、额外开销可控, 作为合入依据。
- CI 与缺失 watermark_config 的配置对象 (testing): 在三个新增读取点改用 getattr(..., "watermark_config", None), 与 VllmConfig 自身的惯用法保持一致。

风险与影响

- 风险: 1) 采样热路径与语义变化: Watermarker.sample 由抽象方法变为模板方法、抽象方法改为 _sample_watermarked, 任何仓库外的自定义水印器子类都会在实例化时失败; GPUWatermarkSampler.__init__ 新增 deduplicate_contexts 等参数, 属于内部 API 但在下游 fork 中常见被包装。2) 性能: dual-key 模式每个采样步要分别用 key A、key B 各跑一次全词表 Gumbel 采样, 再用 random_sampler(routing_logits) 做第三次 (小词表) 采样, 开销随批次与词表放大; 上下文去重的微基准显示小批量下 ms/step 增加 166%-208% (1 token/步时从 0.036ms 到 0.110ms), 大模型端到端基本在 $\pm 5\%$ 以内, 但这是额外成本而非免费。3) 正确性 / 可检测性: 去重意味着部分 token 故意不打水印, PR body 承认“a user will manipulate watermarked text to exclude an initial watermarking of a repeated context”会削弱检测信号; d7fe3812 进一步说明 deduplicate_contexts=all 的检测必须把 prompt 作为上下文前缀, 否则检测口径与生成不一致 (实测 74.7% vs 64.0%)。27B 在低熵输出上检测率仅 23%, 用户可能误判水印失效。4) 内核边界: _repeated_context_mask_kernel 的历史扫描窗口曾经越过 total_len (8d0f1870 修复, 47 次非法全局读), MAX_HISTORY/INCLUDE_PROMPT/SKIP_PARTIAL_CONTEXT 组合增多后该类边界问题容易复发; gpu_sampler._get_contexts 的 draft_indices 计算带 clamp, 仍需靠 tests/v1/spec_decode/test_rejection_sampler_utils.py 的对比测试兜底。5) 预热覆盖: watermark_sample_warmup 依赖 sampler 的 watermarker/deduplicate_contexts/use_fp64_gumbel 三个属性, 一旦新增 specialization 维度而预热未同步, 会退回“首个请求承担 JIT 编译”的旧问题 (97a26400 所修)。

- 影响：对用户：可以在开启投机解码的同时使用水印，新增 `dual_key_gumbel` 算法与 `alpha` 旋钮；默认 `deduplicate_contexts=single_turn` 会改变已有水印部署的采样行为（重复 context 回退普通采样），关闭去重会收到退化生成告警。对系统：改动落在 V1 GPU runner 的采样与验证热路径、Triton kernel、以及启动期 JIT 预热，影响面覆盖 MRv2、spec decode、水印三个子系统；spec_decode.py 与 rejection kernel 的 key 路由一旦出错会静默降低可检测性或改变接受率，因此配套测试较重。对团队：这是一次跨公司协作的典型案例（Mistral 的 TQCB/raphael.rialland、Red Hat 的 simon-veitner-redhat、以及 tomasruizt 直接提交重构补丁），review 中的抽象简化与内核修复都被实际合入。
- 风险标记：核心采样路径变更，水印器基类 API 破坏性调整，Triton 内核越界读（已修复，需防回归），双 key 采样引入额外 kernel 启动，检测率依赖模型输出熵，配置校验行为变化

关联脉络

- PR #54053 Text watermarking in vLLM (PR body 引用的前置水印 PR)：本 PR 的动机起点：现有 Gumbel-max 单 key 水印与投机解码不兼容，本 PR 在其基础上新增 `dual_key_gumbel` 与上下文去重。
- PR #56233 generation-time context deduplication (PR body 引用，本 PR 已包含其工作)：上下文去重部分直接取自该 PR，本 PR 描述中专门附上了它的说明以便读者理解。
- PR #56683 [Perf] Parallelize mHC pre-norm JIT warmup: 同属 vllm/model_executor/warmup 预热体系；本 PR 的 watermark_sample_warmup 与 spec_decode_rejection_warmup 扩展遵循同一“把 JIT 编译从推理期挪到启动期”的思路。
- PR #55768 [Warmup] Gemma 4 de-JITification: 同为消除运行时编译的预热工程改造，与新增的水印预热模块共享 design 约定（明确 specialization 维度、失败不阻断启动）。
- PR #51700 [2/2][Model Runner V2] FULL CUDA graph capture for microbatched steps (DBO): 都改动 vllm/v1/worker/gpu/model_runner.py 与 GPU worker 的采样 / 验证链路，本 PR 的 skip_mask 与 RandomSampler 需要与 MRv2 的批次布局保持兼容。