CacheMedic++: Contractive KV-Cache Stabilization for Robust Transformer Inference

Anonymous Authors

Abstract

Large language models rely on the key–value (KV) cache for efficient autoregressive inference, but this persistent internal state is vulnerable to corruption that can induce output drift without changing prompts or weights. We propose CacheMedic++, a lightweight in-attention KV repair operator trained with frozen base model weights using self-distillation, clean-state identity regularization, and a contraction objective on corrupted states. On our canonical gpt2-medium setting, CacheMedic++ improves clean score from 0.2373 to 0.2466 and robustness AUC from 0.0390 (best heuristic) to 0.0401. In paired ablations, adding contraction improves robustness AUC from 0.0388 to 0.0401 and reduces logit sensitivity to 92.91% ($\delta = 1.0$) and 92.66% ($\delta = 2.0$) of the no-contraction variant. The same operator family transfers qualitatively to gpt2-large, improving clean score from 0.2974 to 0.3001 and robustness AUC from 0.0472 to 0.0475. We emphasize paired contraction ablations and robustness/clean tradeoffs as primary evidence, since absolute stability versus the unmodified baseline is not the central win in this setting.

1 Introduction

Efficient autoregressive decoding in transformers relies on the *key–value (KV) cache*: at each layer, the model stores past keys and values and reuses them to avoid recomputing attention on previous tokens. This cache is not only a performance optimization; it is **persistent internal state** that is read repeatedly across time. Because cached keys/values are repeatedly reused, corruptions of this state can propagate through attention and alter subsequent predictions. Recent work has explored fault models and attacks that act directly on the KV cache, including transformer memory corruption, bit-flip-style cache faults, and history swapping [1, 2, 3].

We study **KV-cache integrity** under reproducible cache perturbations and ask a focused question: *Can we stabilize a frozen model's outputs by learning a small operator that repairs corrupted cache states before attention consumes them?* We explicitly do *not* address KV compression or eviction; our goal is robustness and integrity of the existing cache.

Approach. CacheMedic++ treats transformer inference with a KV cache as a dynamical system over persistent state S_t (the collection of cached (K, V) tensors). We introduce a lightweight *stability operator* R_ϕ inserted *inside attention*: after past/current cache concatenation, we (optionally) corrupt the cache and then apply R_ϕ before attention logits are computed. The base model parameters are frozen; we train only ϕ via self-distillation from clean teacher logits under a deterministic corruption family. To go beyond generic denoising, we add an explicit **contraction regularizer** that encourages repaired cache states to contract toward the clean-cache manifold.

What we can claim (based on bundled evidence). On gpt2-medium under a leave-one-type-out (LOTO) protocol that holds out `contiguous_overwrite` during training and uses it for

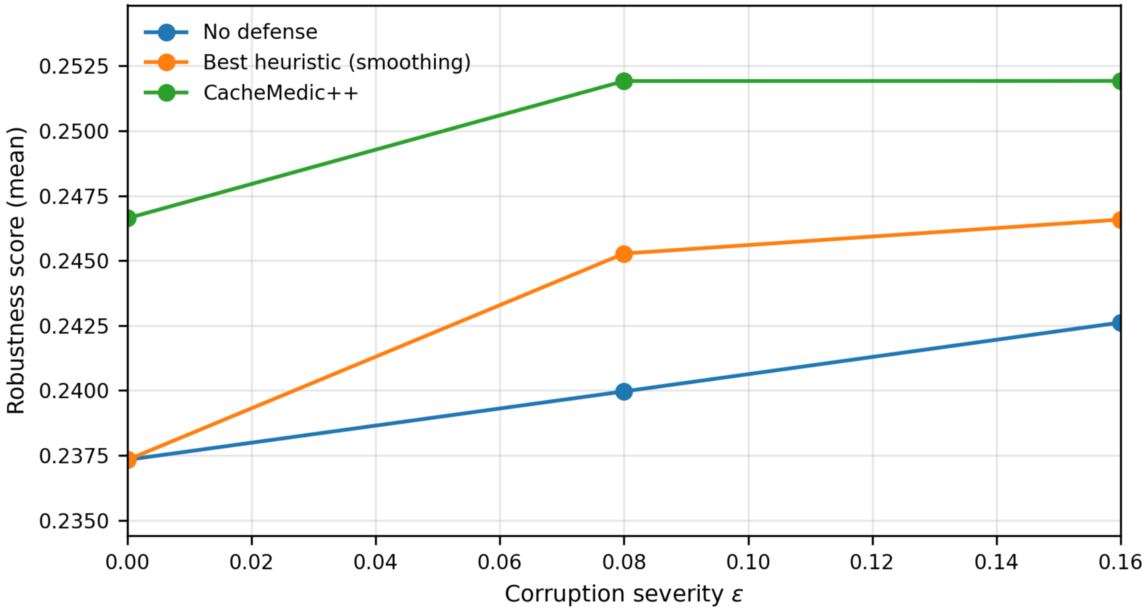

Figure 1: Robustness curves on gpt2-medium under OOD held-out `contiguous_overwrite` (LOTO). The robustness score is the mean of task scores (SST-2 accuracy, inverse Wikitext-2 perplexity, and needle accuracy; higher is better). Table 1 summarizes clean score ($\varepsilon = 0$) and robustness AUC (trapezoidal area under the curve over $\varepsilon \in \{0, 0.08, 0.16\}$).

evaluation, CacheMedic++ improves clean score from 0.2373 to 0.2466 and robustness AUC from 0.0390 (best heuristic) to 0.0401 (Table 1). In a paired ablation that toggles only the contraction weight, contraction improves robustness AUC from 0.0388 to 0.0401 and reduces logit sensitivity relative to the no-contraction variant (Table 3, Fig. 3). A second-model run on gpt2-large shows qualitative transfer in the same direction (Table 7). We emphasize paired ablations and OOD robustness/clean tradeoffs as our strongest evidence; **absolute stability versus the unmodified baseline is not the central win** in this setting.

Contributions.

- **Contractive KV stabilization.** We introduce CacheMedic++, a learned KV-cache stability operator R_ϕ inserted inside attention; base model weights remain frozen and only R_ϕ is trained.
- **Distillation + contraction training.** We train R_ϕ via self-distillation under a deterministic corruption family and add an explicit contraction regularizer toward clean-cache reference states.
- **Stability as a first-class metric.** We provide reproducible protocols for logit sensitivity curves and layer/head amplification maps, and we report paired contraction ablations and second-model transfer.

2 Threat and Fault Model

We define a deterministic family of cache corruptions C that perturbs the KV cache during decoding. Throughout, we assume batch size B=1 and eager attention. For each transformer layer l, the (past) cache tensors are $K^{(l)}, V^{(l)} \in \mathbb{R}^{1\times H\times T\times d}$, where H is the number of heads, T is the number of cached timesteps, and d is the head dimension. We call the set of protected layers $\mathcal{L}_{\text{prot}}$; the protected state is $S = \{(K^{(l)}, V^{(l)}) : l \in \mathcal{L}_{\text{prot}}\}$.

2.1 Axis masks (layer/head/time)

Each corruption is applied under three deterministic binary masks: a layer mask $m_\ell(l)$, a head mask $m_h(h)$, and a time mask $m_t(t)$. Mask sampling uses a dedicated `torch.Generator` seeded from the run seed, so that the corruption is exactly reproducible. The time mask supports three modes: `all_past` (corrupt all cached timesteps), `old_only` (corrupt only timesteps $t < T - N_{\text{recent}}$), and `window` (corrupt a contiguous window $[a, b)$). Our canonical runs use `old_only` with N_{recent}=32 and corrupt heads with Bernoulli(p_h=0.25).

2.2 Corruption operators

Corruptions are implemented with torch-only operations and are deterministic given the seed and parameters. We include six operator types:

1. **Gaussian noise** (scaled by per-vector RMS),
2. **Dropout/zeroing** (elementwise zero under the mask),
3. **Orthogonal rotation** on the feature dimension (QR-derived orthogonal matrix with fixed seed),
4. **Sparse bitflip-ish faults** (random sign flips and magnitude "jumps" in float space),
5. **Quantization noise** (simulate symmetric n-bit de/quant),
6. **Contiguous overwrite** (overwrite a past window of KV with a donor cache computed from a deterministic donor prompt).

Precise pseudocode (including determinism rules) is given in Appendix A.2.

2.3 Mixtures and OOD leave-one-type-out (LOTO)

A corruption run samples a type and parameters from a fixed mixture Π and samples severities from an ε-grid. To test robustness generalization, we follow a leave-one-type-out protocol: we **exclude** `contiguous_overwrite` from the training corruption mixture and evaluate on the held-out `contiguous_overwrite` type for $\varepsilon \in \{0.0, 0.08, 0.16\}$. This isolates whether a learned repair operator trained on other cache faults can generalize to an unseen cache manipulation.

3 CacheMedic++

CacheMedic++ adds a small stability operator R_ϕ that repairs the KV cache *inside* the attention computation. The base model weights θ are frozen; only ϕ is trained.

3.1 Insertion point inside attention

Consider one attention layer. Let $K_{\text{full}}, V_{\text{full}}$ denote the concatenation of past and current-step keys/values. We (optionally) apply a cache corruption C to obtain $(K_{\text{corr}}, V_{\text{corr}}) = C(K_{\text{full}}, V_{\text{full}})$. We then apply the repair operator before attention logits are computed:

$$(K_{\text{hat}}, V_{\text{hat}}) = R_\phi(q, K_{\text{corr}}, V_{\text{corr}}),$$

and compute attention using $K_{\text{hat}}, V_{\text{hat}}$. In our canonical runs, R_ϕ is applied only on a small set of protected layers $\mathcal{L}_{\text{prot}}$ and is the identity elsewhere.

3.2 Repair operator family (Option A, used in results)

We use Option A from the harness: a **query-conditioned low-rank additive** correction. Parameters are shared across heads by default. For a protected layer, let r be the rank and d the head dimension. Let $V \in \mathbb{R}^{1\times H\times T\times d}$ and $q \in \mathbb{R}^{1\times H\times d}$ be the per-head query for the current step. We learn projection matrices $W_v, U_v \in \mathbb{R}^{d\times r}$ and a gating network g_ϕ that outputs $\alpha = g_\phi(q) \in (0,1)^{1\times H\times r}$. We compute:

$$C = VW_v \in \mathbb{R}^{1\times H\times T\times r}, \qquad \Delta V = (C \odot \alpha[:, :, \text{None}, :])U_v^\top,$$

and set $V_{\text{hat}} = V + \Delta V$. The same form applies to keys if `apply_to` includes K; our tuned configuration uses `apply_to=V` with rank r=4 on two mid-to-late layers (Appendix A.1).

3.3 Training objective: KD + identity + contraction

CacheMedic++ is trained by self-distillation with a clean teacher. For each training step, we sample either a **clean** batch with probability p_{clean} (no corruption) or a **corrupted** batch from the corruption mixture otherwise. Let z_{clean} be the next-token logits from the frozen teacher with a clean cache, and let z_{rep} be logits from the same frozen model when the cache is corrupted and then repaired.

KD loss. With temperature T:

$$p = \text{softmax}(z_{\text{clean}}/T), \quad q = \text{softmax}(z_{\text{rep}}/T), \quad \mathcal{L}_{\text{KD}} = \text{KL}(p \,\|\, q).$$

Identity regularizer. On clean batches (corruption disabled), we penalize deviation from identity on protected layers:

$$\mathcal{L}_{\text{id}} = \sum_{l\in\mathcal{L}_{\text{prot}}} \frac{\|K^{(l)}_{\text{hat}} - K^{(l)}_{\text{clean}}\|_F^2 + \|V^{(l)}_{\text{hat}} - V^{(l)}_{\text{clean}}\|_F^2}{\|K^{(l)}_{\text{clean}}\|_F^2 + \|V^{(l)}_{\text{clean}}\|_F^2 + \epsilon_0}.$$

We use $\epsilon_0 = 10^{-8}$ in denominators for numerical stability.

Contraction regularizer. On corrupted batches, we encourage repair to *contract* the corrupted cache toward the clean cache. Define per-layer error tensors $\Delta K^{(l)} = K^{(l)}_{\text{corr}} - K^{(l)}_{\text{clean}}$ and $\Delta V^{(l)} = V^{(l)}_{\text{corr}} - V^{(l)}_{\text{clean}}$, and similarly $\Delta K^{(l)}_{\text{rep}} = K^{(l)}_{\text{hat}} - K^{(l)}_{\text{clean}}$ and $\Delta V^{(l)}_{\text{rep}} = V^{(l)}_{\text{hat}} - V^{(l)}_{\text{clean}}$. We use the contraction ratio

$$\rho(l) = \frac{\|\Delta K^{(l)}_{\text{rep}}\|_F + \|\Delta V^{(l)}_{\text{rep}}\|_F}{\|\Delta K^{(l)}\|_F + \|\Delta V^{(l)}\|_F + \epsilon_0},$$

and a hinge penalty toward a target α_{contr}:

$$\mathcal{L}_{\text{contr}} = \sum_{l\in\mathcal{L}_{\text{prot}}} \max(0, \rho(l) - \alpha_{\text{contr}})^2.$$

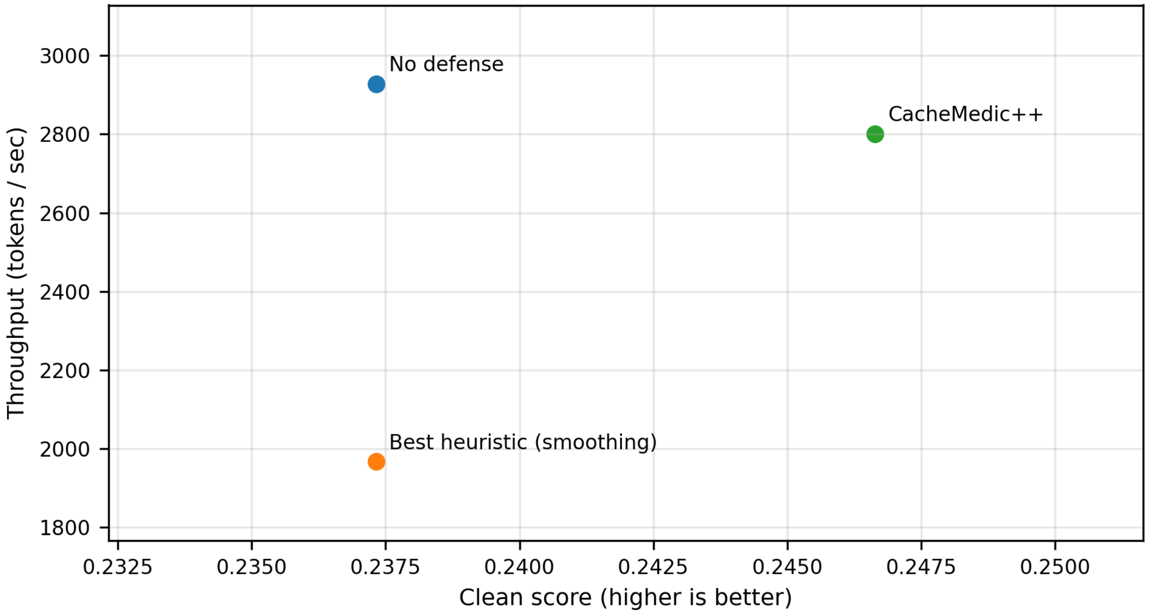

Figure 2: Clean-score / throughput tradeoff on gpt2-medium. We measure decoding throughput (tokens/sec) on the same evaluation setting used for Table 1. CacheMedic++ preserves most of the baseline throughput while improving clean score.

Total loss.

$$\mathcal{L} = \mathcal{L}_{\text{KD}} + \lambda_{\text{id}}\mathcal{L}_{\text{id}} + \lambda_{\text{contr}}\mathcal{L}_{\text{contr}}.$$

Our tuned run uses T=2.0, p_{clean}=0.45, λ_{id}=4.0, λ_{contr}=3.0, α_{contr}=0.75, with frozen base weights.

4 Evaluation Metrics

CacheMedic++ reports both task robustness and stability metrics as primary outputs.

4.1 Task robustness score and robustness AUC

We evaluate three tasks: Wikitext-2 perplexity (WT2 PPL), SST-2 prompted classification (accuracy), and a deterministic needle-style long-context probe (accuracy). To aggregate heterogeneous tasks into a single robustness score, we map each task metric to a *score* $s \in [0, 1]$: for WT2 we use $s_{\text{WT2}} = 1/\text{PPL}$, and for SST-2 and needle we use accuracy. The overall score is the mean across tasks:

$$\text{Score}(\varepsilon) = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} s_\tau(\varepsilon).$$

For a fixed corruption type and a grid of severities $\{\varepsilon_0, \ldots, \varepsilon_n\}$, we produce a **robustness curve** $\text{Score}(\varepsilon)$. We summarize it by the trapezoidal area under the curve:

$$\text{AUC} = \text{trapz}(\text{Score}(\varepsilon), \varepsilon).$$

We report: (i) **clean score** Score(0), and (ii) **robustness AUC** over the evaluation severity grid (Table 1).

Best heuristic baseline. We evaluate four inference-only heuristics from the harness (reset/clear, smoothing, masking/dropout, clipping/renorm). To avoid cherry-picking, we select *best heuristic* as the heuristic with the highest robustness AUC on the evaluation grid, and report that method alongside no defense and CacheMedic++.

Bootstrap uncertainty for paired replication. For paired multi-seed comparisons (Sec. 4), we report percentile bootstrap confidence intervals over seed-level paired statistics. We resample the seed pairs with replacement ($B = 20{,}000$) and compute CIs from the 2.5th and 97.5th percentiles. We apply this to the robustness AUC gain (contr − no_contr) and sensitivity ratios at $\delta \in \{1, 2\}$.

4.2 Stability metrics: logit sensitivity and amplification maps

Logit sensitivity. We measure finite-difference logit drift under *cache-state perturbations* at magnitudes $\delta \in \{0, 1, 2\}$. Let S denote the clean protected cache state for a prompt, and let Δ be a random perturbation tensor with the same structure as S, scaled by per-layer RMS so that its norm matches the requested δ. Let $z(\cdot)$ be the next-token logits. We define sensitivity:

$$\mathrm{Sens}(\delta) = \mathbb{E}_{x,\Delta}\big[\|z(S + \Delta) - z(S)\|_2\big],$$

and the repaired version uses logits after applying R_ϕ to $S + \Delta$. To control cost, we compute $\|\cdot\|_2$ over the top-k logits of $z(S)$ (here k=512), using 60 prompts and 4 directions per prompt.

Amplification maps. We estimate which layer/head pairs amplify KV perturbations into output drift. For each layer l and head h, we inject a perturbation that is nonzero only in (l, h) and measure the median drift normalized by perturbation magnitude:

$$\gamma(l, h) = \mathrm{median}_{x,\Delta}\left[\frac{\|z(S + \Delta_{l,h}) - z(S)\|_2}{\|\Delta_{l,h}\|_F + \epsilon_0}\right],$$

where ϵ_0 is a small constant (we use $\epsilon_0 = 10^{-8}$). We report $\gamma(l, h)$ for selected layers/heads and visualize differences (Fig. 4).

5 Experiments

We report results on two frozen GPT-2 models (gpt2-medium and gpt2-large) using the harness protocols bundled with this paper. All reported numbers are computed from the accompanying evidence JSON files (see Appendix A.6).

5.1 Setup

Models. We evaluate gpt2-medium as the primary model and gpt2-large as a second-model confirmation. Inference uses batch size 1, eager attention, and the standard KV cache.

Tasks. We evaluate on: (i) Wikitext-2 perplexity (400 examples), (ii) SST-2 prompted classification (250 examples), and (iii) a deterministic needle-style long-context probe (80 examples). Exact prompting and subset sizes are specified in Appendix A.4.

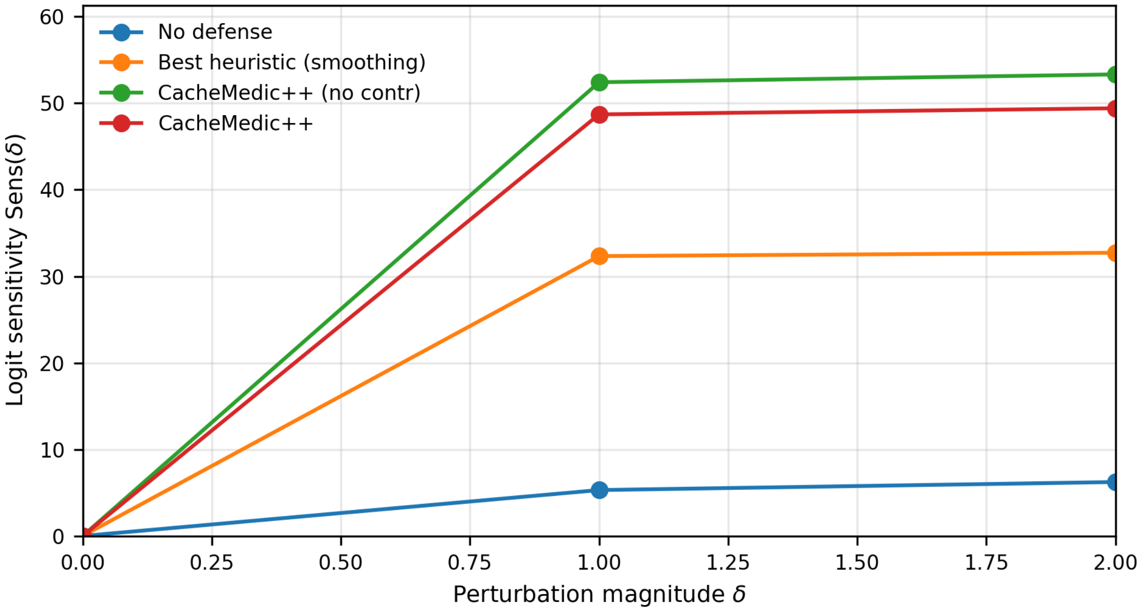

Figure 3: Logit sensitivity curves on gpt2-medium. Contraction reduces sensitivity relative to the paired no-contraction variant, while absolute sensitivity relative to the unmodified baseline remains a limitation in this setting.

Table 1: gpt2-medium results under OOD held-out `contiguous_overwrite` (LOTO). Clean score is at $\varepsilon = 0$; robustness AUC is trapezoidal area over $\varepsilon \in \{0, 0.08, 0.16\}$.

Method	Clean Score	Robustness AUC
No defense	0.2373	0.0384
Best heuristic (smoothing)	0.2373	0.0390
CacheMedic++	**0.2466**	**0.0401**

OOD protocol and corruption. We train CacheMedic++ on a fixed mixture of five corruption types and hold out `contiguous_overwrite`. Evaluation is performed on the held-out `contiguous_overwrite` type at $\varepsilon \in \{0, 0.08, 0.16\}$ (LOTO; Sec. 2). For all runs, corruption applies only to the `old_only` segment of the cache (all but the most recent 32 tokens), exposing long-range cache dependence.

Baselines and ablations. We compare against: **no defense** and four inference-only heuristics (reset/clear, smoothing, masking/dropout, clipping/renorm). We report **best heuristic** chosen by robustness AUC on the evaluation grid. To test the stability framing, we include a **paired contraction ablation**: the same repair operator and training setup with λ_{contr} set to 0.

5.2 Main results (gpt2-medium)

Table 1 summarizes clean score and robustness AUC. CacheMedic++ improves both clean score and robustness AUC compared to no defense and the best heuristic baseline. Figure 1 shows the full robustness curve.

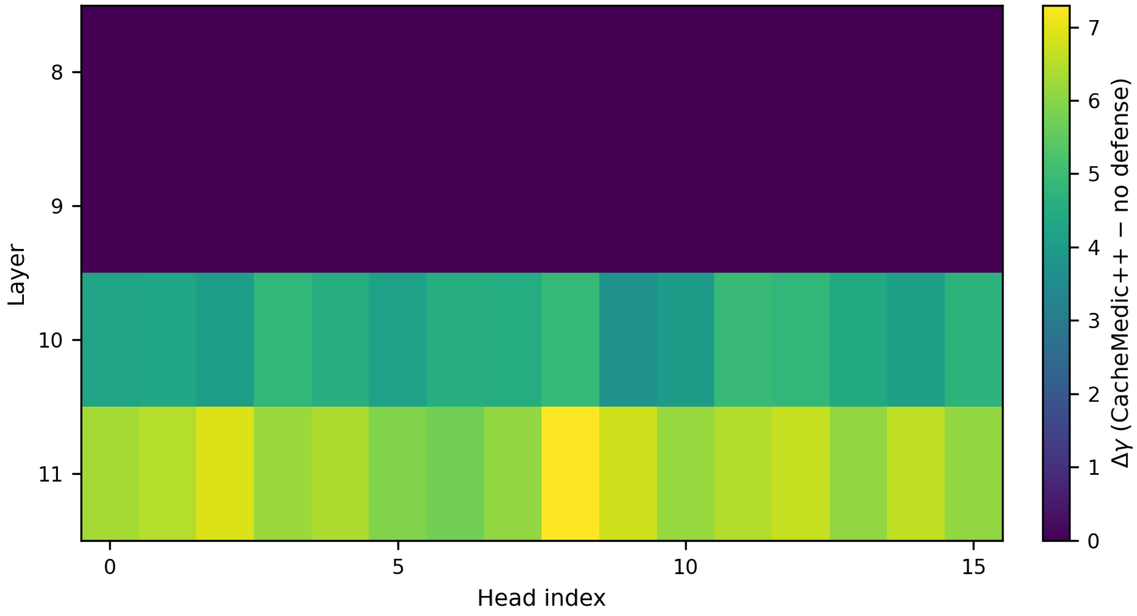

Figure 4: Layer/head amplification differences on gpt2-medium. Shown is $\Delta\gamma(l,h) = \gamma_{\text{CacheMedic++}}(l,h) - \gamma_{\text{no defense}}(l,h)$ over layers 8–11 and heads 0–15. In this configuration, amplification increases are concentrated in the protected layers (10–11), aligning with the "absolute stability" limitation noted in the main text.

5.3 Paired contraction ablation

Table 3 isolates the contraction regularizer by toggling only λ_{contr}. Contraction improves robustness AUC and reduces logit sensitivity relative to the no-contraction variant at both $\delta = 1$ and $\delta = 2$. We stress that this paired comparison is the cleanest evidence that the stability term contributes beyond the distillation objective.

5.4 Three-seed replication (contr vs no_contr)

To test run-to-run stability, we repeated the same paired configuration with two additional seeds (2026 and 2027) and combined them with the canonical seed (1234). Table 4 reports mean ± std over the 3-seed paired set. The average robustness AUC gain is small ($+0.0001 \pm 0.0012$), and the mean sensitivity ratios are above 1 (1.0584 ± 0.1628 at $\delta = 1$, 1.0892 ± 0.2004 at $\delta = 2$), indicating substantial seed variance and no consistent stability gain across all seeds. We therefore treat the paired single-seed ablation as suggestive evidence and the 3-seed result as an explicit limitation that motivates stronger statistical reporting.

Table 5 adds bootstrap uncertainty over paired seeds ($B = 20{,}000$). All intervals include no-effect thresholds (AUC gain = 0, ratio = 1), reinforcing that this effect is currently not statistically stable in our small-seed regime.

5.5 Mini hyper-sensitivity: $\lambda_{\text{contr}} \in \{0, 3, 5\}$

Table 6 provides a compact one-seed hyper-sensitivity check under the same tuned setup. The strongest setting is $\lambda_{\text{contr}} = 3.0$. Pushing contraction to $\lambda_{\text{contr}} = 5.0$ degrades clean score and robustness AUC and increases sensitivity beyond both $\lambda_{\text{contr}} = 3.0$ and $\lambda_{\text{contr}} = 0.0$. This is an

Table 2: Compact claim-to-evidence map in main text (key quantitative claims and their exact evidence files/keys).

Claim (main text)	Value	Evidence source
gpt2-medium clean score (CacheMedic++)	0.2466	`evidence/gpt2_medium_contr/metrics/task_metrics.json` `metrics.clean_regression.cachemedic`
gpt2-medium robustness AUC (CacheMedic++)	0.0401	`evidence/gpt2_medium_contr/metrics/task_metrics.json` `metrics.robustness_auc.cachemedic`

Paired ablation AUC (no_contr → contr)	0.0388 → 0.0401	`evidence/gpt2_medium_no_contr/metrics/task_metrics.json` and `evidence/gpt2_medium_contr/metrics/task_metrics.json`
		`metrics.robustness_auc.cachemedic`

Paired sensitivity ratios ($\delta = 1, 2$)	0.9291, 0.9266	`evidence/gpt2_medium_no_contr/metrics/stability_metrics.json` and `evidence/gpt2_medium_contr/metrics/stability_metrics.json`
		`metrics["stability.logit_sensitivity"]`

gpt2-large clean score / robustness AUC (CacheMedic++)	0.3001 / 0.0475	`evidence/gpt2_large_contr/metrics/task_metrics.json`
		`metrics.clean_regression.cachemedic`, `metrics.robustness_auc.cachemedic`

Table 3: Paired contraction ablation on gpt2-medium (same operator and training setup; only λ_{contr} differs). "Stability ratio" is $\text{Sens}_{\text{contr}}(\delta)/\text{Sens}_{\text{no contr}}(\delta)$; values below 1 indicate improved stability.

Metric	no_contr	contr
Robustness AUC	0.0388	**0.0401**
Logit sensitivity ($\delta = 1.0$)	52.4250	**48.7083**
Logit sensitivity ($\delta = 2.0$)	53.3308	**49.4163**
Sensitivity ratio contr/no_contr ($\delta = 1.0$)	0.9291	
Sensitivity ratio contr/no_contr ($\delta = 2.0$)	0.9266	
Repair family / apply_to / rank	A / V / 4	

explicit negative result and supports a "sweet spot" interpretation rather than monotonic gains from larger contraction weight.

5.6 Second model: gpt2-large

Table 7 reports the same evaluation protocol on gpt2-large. CacheMedic++ shows a smaller but consistent improvement in both clean score and robustness AUC.

5.7 Stability behavior and the "absolute stability" limitation

Figure 3 reports logit sensitivity curves and includes the no-contraction ablation. Contraction reduces sensitivity relative to the no-contraction variant (Table 3), but **absolute sensitivity relative to the unmodified baseline remains higher** in this setting. We therefore frame CacheMedic++ primarily as a *relative stabilization* method whose strongest evidence comes from paired ablations, robustness/clean tradeoffs, and second-model transfer rather than absolute stability

Table 4: Three-seed replication for the paired contraction ablation (same tuned config, seeds 1234/2026/2027). Reported as mean ± std.

Metric	no_contr ($\mu \pm \sigma$)	contr ($\mu \pm \sigma$)	Effect ($\mu \pm \sigma$)
Clean score	0.2440 ± 0.0048	0.2444 ± 0.0020	$+0.0004 \pm 0.0067$
Robustness AUC	0.0394 ± 0.0007	0.0395 ± 0.0005	$+0.0001 \pm 0.0012$
Logit sensitivity ($\delta = 1.0$)	35.3836 ± 14.7711	36.3964 ± 11.3408	$+1.0128 \pm 5.2282$
Logit sensitivity ($\delta = 2.0$)	35.4753 ± 15.4637	37.1999 ± 11.2837	$+1.7246 \pm 6.1917$
Sensitivity ratio contr/no_contr ($\delta = 1.0$)		1.0584 ± 0.1628	
Sensitivity ratio contr/no_contr ($\delta = 2.0$)		1.0892 ± 0.2004	

Table 5: Bootstrap uncertainty for the 3-seed paired replication (percentile CI over seed pairs, $B = 20{,}000$).

Quantity (paired over seeds)	Mean	95% bootstrap CI
Robustness AUC gain (contr - no_contr)	+0.0001	[−0.0011, 0.0013]
Sensitivity ratio contr/no_contr ($\delta = 1.0$)	1.058	[0.929, 1.241]
Sensitivity ratio contr/no_contr ($\delta = 2.0$)	1.089	[0.927, 1.313]

improvements over the unmodified model.

Figure 4 further visualizes that amplification differences are concentrated in the protected layers (10–11), consistent with the operator being active only in that layer subset.

6 Related Work

KV-cache corruption and manipulation. Recent work has explored fault models and attacks that act directly on the transformer KV cache, including transformer memory corruption [1], cache-based attacks [2], and history-swapping manipulations [3]. These directions motivate cache integrity as a robustness surface distinct from input perturbations or weight perturbations.

State interventions and steering. A broad line of work studies interventions on internal states for controlling model behavior. Conceptor steering [6] is one example of a state-based steering approach. CacheMedic++ is complementary: we focus on repairing corrupted KV-cache state to match the clean-cache behavior of a frozen model, rather than steering outputs toward a task-specific target.

Stability and activation editing. SEA [5] proposes spectral editing of activations, and activation-boundary defense [4] proposes constraining activations to safeguard LLM behavior. CacheMedic++ differs by (i) operating specifically on the persistent KV cache used by attention, (ii) training a small operator via self-distillation against a clean-cache teacher under cache corruptions, and (iii) adding an explicit contraction regularizer and evaluating stability metrics (logit sensitivity and amplification maps) alongside task robustness.

7 Limitations and Ethical Considerations

Limitations.

- **Absolute stability is not the central win.** In our stability measurements, CacheMedic++ does not consistently reduce *absolute* logit sensitivity compared to the unmodified baseline

Table 6: Mini hyper-sensitivity on the tuned V-only setup (single-seed comparison).

λ_{contr}	Clean score	Robustness AUC	Sensitivity ($\delta = 1$)	Sensitivity ($\delta = 2$)
0.0	0.2400	0.0388	52.4250	53.3308
3.0	**0.2466**	**0.0401**	**48.7083**	**49.4163**
5.0	0.2413	0.0390	54.5851	55.3688

Table 7: Second-model confirmation on gpt2-large under the same OOD held-out `contiguous_overwrite` protocol.

Method	Clean Score	Robustness AUC
No defense	0.2974	0.0472
Best heuristic (smoothing)	0.2974	0.0472
CacheMedic++	**0.3001**	**0.0475**

(Fig. 3). Our strongest evidence is (i) the paired contraction-vs-no-contraction ablation (Table 3), (ii) robustness/clean tradeoffs (Table 1), and (iii) second-model qualitative transfer (Table 7).

- **Simulated fault model.** Our corruption operators are deterministic simulations of cache perturbations; they are not direct measurements of hardware faults or real-world adversaries. Generalization to other fault distributions is an open question.
- **Scope of models and implementations.** The bundled evidence covers GPT-2 models with batch size 1 and eager attention. We do not evaluate flash attention implementations or larger LLMs.
- **Task coverage.** The needle-style long-context probe yields zero accuracy for all methods in the bundled runs; thus improvements in the aggregate score are driven by the other tasks (Appendix A.4).

7.1 Threats to validity

Internal validity. Our conclusions rely on strict config matching between paired runs; accidental mismatch in corruption injection, checkpoint selection, or metric aggregation could bias ablation outcomes. We mitigate this with explicit evidence-path mapping and paired run records, but latent harness bugs remain possible.

External validity. Results are from GPT-2 medium/large with eager attention and batch size 1 under a synthetic but deterministic corruption family. Transfer to larger models, other inference kernels (for example flash attention), or non-simulated real-world fault patterns is not established.

Statistical validity. The 3-seed replication shows high variance, and bootstrap CIs over paired seeds cross no-effect thresholds. Therefore, claims about contraction should be interpreted as provisional at current sample size and motivate larger replicated studies.

Ethical considerations. This work studies how cache corruption affects model outputs and proposes a defensive repair mechanism. While our threat model includes targeted cache manipulations, we focus on reproducible perturbations that enable evaluation, and we report limitations in absolute stability. We do not claim a security guarantee; deploying defenses in adversarial settings requires broader threat modeling and auditing.

References

[1] Md Tahmid Hossain and others. Can Transformer Memory Be Corrupted? *arXiv preprint arXiv:2510.17098*, 2025.

[2] Md Jamiul Nahian and others. CacheTrap: Exploiting KV Cache Bit Flips for Targeted Behavior in LLM Inference. *arXiv preprint arXiv:2511.22681*, 2025.

[3] Prakhar Ganesh and others. Whose Narrative is it Anyway? A KV Cache Manipulation Attack. *arXiv preprint arXiv:2511.12752*, 2025.

[4] Yiming Gao and others. Activation Boundary Defense for Safeguarding Large Language Models. In *Proceedings of ACL*, 2025.

[5] Y. Qiu and others. Spectral Editing of Activations. In *NeurIPS*, 2024.

[6] Levi Postmus and Marcos Abreu. Conceptor Steering for Language Models. *arXiv preprint arXiv:2410.16314*, 2024.

A Appendix

This appendix makes the paper self-sufficient and ties every quantitative claim to an evidence file contained in the bundle.

A.1 Canonical configurations

All reported results use repair family A (query-conditioned low-rank additive correction) with `apply_to=V`, rank r=4, and protected layers $\mathcal{L}_{\text{prot}} = \{10, 11\}$. The base GPT-2 weights are frozen. Training uses 2500 steps with batch size 1 and prefix length 80. Key hyperparameters for the gpt2-medium tuned run are: temperature T=2.0, p_{clean}=0.45, λ_{id}=4.0, λ_{contr}=3.0, α_{contr}=0.75. The exact resolved configs are stored in: `evidence/gpt2_medium_contr/config_resolved.yaml` and `evidence/gpt2_medium_no_contr/config_resolved.yaml`, and analogously for gpt2-large.

A.2 Corruption operators (deterministic spec)

Corruptions act on cached K, V tensors with shape $[1, H, T, d]$ and are applied under deterministic layer/head/time masks (Sec. 2). All randomness uses a dedicated `torch.Generator` seeded from the run seed so that corruptions are exactly reproducible.

We write the masked tensor as $X_m = X \odot m$, where the mask m is the broadcast product of the chosen layer/head/time masks. The bundled runs instantiate the operator parameters as follows (from `evidence/*/config_resolved.yaml`):

- **Gaussian noise (type `gaussian`).** $X \leftarrow X + m \odot (\varepsilon \cdot \mathrm{RMS}(X) \cdot \mathcal{N}(0,1))$, with RMS computed per head and timestep and `gaussian_scale_by_rms=true`.
- **Dropout/zeroing (type `dropout_zero`).** Elementwise set X_m to zero with probability p (here p=0.02), leaving unmasked elements unchanged.
- **Orthogonal rotation (type `orthogonal_rotation`).** Apply $X_m \leftarrow X_m R$ on the feature dimension, where $R \in \mathbb{R}^{d \times d}$ is orthogonal and generated via QR factorization from a fixed seed (here `rotation_seed=999`); unmasked elements are unchanged.
- **Sparse bitflip-ish faults (type `bitflipish_sparse`).** With probability p per element (here p=0.0005), either flip sign $x \leftarrow -x$ or apply a signed "jump" $x \leftarrow x + \mathrm{sign}(\eta) \cdot \varepsilon_{\text{jump}} \cdot \max(|x|, 10^{-3})$ (here $\varepsilon_{\text{jump}}$=8.0), where $\eta \sim \mathcal{N}(0,1)$.
- **Quantization noise (type `quant_noise`).** Simulate symmetric n-bit de/quant (here n=8) with per-head scaling: X_m is mapped to integers in $[-2^{n-1}+1, 2^{n-1}-1]$ and dequantized back to float; unmasked elements are unchanged.
- **Contiguous overwrite (type `contiguous_overwrite`).** Overwrite a fixed window of old cache positions (here `overwrite_window=[16,48)`) with the corresponding window from a donor cache computed from a deterministic donor prompt string (provided in the resolved config). This corruption type is held out during training and used for OOD evaluation (LOTO).

A.3 Training pseudocode (one step)

The following pseudocode matches the training objective in Sec. 3 and the harness implementation (teacher and student share frozen base weights):

```
Inputs: prompts x, frozen base model f_theta, trainable repair operator R_phi
Sample is_clean ~ Bernoulli(p_clean)
```

```
Run teacher with clean cache:
  z_clean, (K_clean, V_clean) = f_theta(x)

if is_clean:
  # no corruption; repair should be identity
  (K_hat, V_hat) = R_phi(q, K_clean, V_clean)        # protected layers only
  z_rep = f_theta.forward_with_cache(K_hat, V_hat)
  L = KL(softmax(z_clean/T) || softmax(z_rep/T))
  L += lambda_id * L_id(K_hat,V_hat; K_clean,V_clean)

else:
  (K_corr, V_corr) = C(K_clean, V_clean)             # sample type/params from Pi
  (K_hat, V_hat) = R_phi(q, K_corr, V_corr)
  z_rep = f_theta.forward_with_cache(K_hat, V_hat)
  L = KL(softmax(z_clean/T) || softmax(z_rep/T))
  L += lambda_contr * L_contr(K_hat,V_hat; K_clean,V_clean, K_corr,V_corr)

Backpropagate L into phi only; theta is frozen.
```

A.4 Evaluation details

SST-2 prompted classification. We compute label log-probabilities for the tokens corresponding to the labels `"negative"` and `"positive"` under the prompt template:

```
Review: <text> Sentiment:
```

Accuracy is computed over 250 examples (see `evidence/*/config_resolved.yaml`).

Needle long-context probe. We generate a deterministic long context with a planted key/value pair and query for the key. The generator settings in the canonical run are: context length 2048, needle token `the_key`, needle value `violet-7`, insert position fraction 0.35, and question template `"Question: What is needle_token? Answer:"`. In the bundled runs, all methods obtain 0 accuracy on this probe (Table 8), so the aggregate robustness score is driven by WT2 and SST-2.

A.5 Per-task breakdown (gpt2-medium)

Table 8 reports raw per-task metrics for gpt2-medium at clean and at the largest evaluated corruption severity (ε=0.16) for the OOD held-out `contiguous_overwrite` corruption.

Table 8: Per-task metrics for gpt2-medium. "Best heuristic" is smoothing in this evaluation protocol.

		Clean	Corrupt ($\varepsilon = 0.16$)		
Task	Metric	($\varepsilon = 0$)	No def.	Best heur.	CacheMedic++
Wikitext-2	PPL ↓	35.73	35.91	36.03	36.00
SST-2	Acc ↑	0.684	0.700	0.712	0.728
Needle	Acc ↑	0.000	0.000	0.000	0.000

A.6 Evidence map for all numeric claims

All numeric claims in the main text are sourced from these files (relative to the bundle root):
`evidence/gpt2_medium_contr/metrics/task_metrics.json`
`evidence/gpt2_medium_contr/metrics/stability_metrics.json`
`evidence/gpt2_medium_no_contr/metrics/task_metrics.json`
`evidence/gpt2_medium_no_contr/metrics/stability_metrics.json`
`evidence/gpt2_large_contr/metrics/task_metrics.json`

Resolved configurations used by the harness are:
`evidence/gpt2_medium_contr/config_resolved.yaml`
`evidence/gpt2_medium_no_contr/config_resolved.yaml`
`evidence/gpt2_large_contr/config_resolved.yaml`