Robustifying Learning-Augmented Caching Efficiently without Compromising 1-Consistency

Peng Chen[1] **(Presenter)**, Hailiang Zhao[1*], Jiaji Zhang[1], Xueyan Tang[2], Yixuan Wang[3], Shuiguang Deng[1*]

[1] Zhejiang University [2] Nanyang Technological University
[3] Nanjing University of Aeronautics and Astronautics

Our Work in A Nutshell

- Summarize how existing learning-augmented algorithms **obtain** robustness and their limitations.
- Propose an efficient robustification framework for **1-consistent learning-augmented algorithms** that preserves their 1-consistency.

The Online Caching (Paging) Problem

- **Goal:** Serve requests with a cache of size k, minimizing the cost (i.e., the number of cache misses).
- **Offline Optimum:** Belady's rule — evict the page used furthest in the future.
- **Online Algorithm**:
  - **Challenge**: Future requests are unknown.
  - **Competitive Ratio:** Compare cost to the offline optimum.

Learning-Augmented Caching

- Use Machine Learning (ML) predictions (e.g., next request time) to guide eviction.
- **Goal:** Perform well when predictions are good, but remain stable when they are bad.

Key Metrics:

Consistency (γ) : Competitive ratio with perfect predictions. (Ideal: $\gamma = 1$)

Robustness (δ) : Competitive ratio with arbitrary (worst-case) predictions. (Ideal: small constant, e.g., $\mathcal{O}(H_k)$)

Smoothness : How performance degrades as prediction error η increases.

Existing Learning-Augmented Algorithms

- Naive algorithms (BlindOracle) achieve 1-consistency but have no bounded robustness.

Existing robustification methods fall short:

- Embedding-based (PredictiveMarker, LMarker)
  - Good robustness (e.g., $\mathcal{O}(\log k)$).
  - Sacrifices 1-consistency.
- Switching-based (F&R)
  - Achieves 1-consistency with good robustness ($\mathcal{O}(\log k)$).
  - Suffers from high computational overhead: $\mathcal{O}(n^2 \log k)$ total for n requests.

Our Research Question

Can we enhance robustness in a **time-efficient** manner **without compromising 1-consistency**?

Existing Learning-Augmented Algorithms

- Naive algorithms (BlindOracle) achieve 1-consistency but have no bounded robustness.

Existing robustification methods fall short:

- Embedding-based (PredictiveMarker, LMarker)
  - Good robustness (e.g., $\mathcal{O}(\log k)$).
  - Sacrifices 1-consistency.
- Switching-based (F&R)
  - Achieves 1-consistency with good robustness ($\mathcal{O}(\log k)$).
  - Suffers from high computational overhead: $\mathcal{O}(n^2 \log k)$ total for n requests.

Our Research Question

Can we enhance robustness in a **time-efficient** manner **without compromising 1-consistency**?

Insights

Observation 1: **Marker-based** methods protect pages on request, which can block correct evictions even with perfect predictions, thus breaking 1-consistency.

Observation 2: Prediction error detection must **trade off** efficiency and sensitivity: recomputing the offline optimum is too slow, while comparing to LRU is too insensitive.

Observation 3: Optimal algorithms (including Belady) **never keep** a page x in the cache that is not requested during the time interval between the eviction and the next request of another page y; otherwise, evicting y instead would be better.

Insights

Observation 1: **Marker-based** methods protect pages on request, which can block correct evictions even with perfect predictions, thus breaking 1-consistency.

Observation 2: Prediction error detection must **trade off** efficiency and sensitivity: recomputing the offline optimum is too slow, while comparing to LRU is too insensitive.

Observation 3: Optimal algorithms (including Belady) **never keep** a page x in the cache that is not requested during the time interval between the eviction and the next request of another page y; otherwise, evicting y instead would be better.

Insights

Observation 1: **Marker-based** methods protect pages on request, which can block correct evictions even with perfect predictions, thus breaking 1-consistency.

Observation 2: Prediction error detection must **trade off** efficiency and sensitivity: recomputing the offline optimum is too slow, while comparing to LRU is too insensitive.

Observation 3: Optimal algorithms (including Belady) **never keep** a page x in the cache that is not requested during the time interval between the eviction and the next request of another page y; otherwise, evicting y instead would be better.

How GUARD Makes Decisions (Visual Flow)

1. **Cache Miss:** Request p_i misses.
2. **Check Phase Status:** Is $\mathcal{U}$ empty?
   - ✓ **Yes:** Start New Phase. $\mathcal{U} \leftarrow$ all pages.
   - × **No:** Continue current phase.
3. **Check for Prediction Error:** Has p_i been evicted in the current phase?
   - ✓ **Yes (prediction error!):**
     - Evict **random page** from $\mathcal{U}$. $\rightarrow$ (Robustness)
     - **Guard** p_i.
   - × **No:**
     - Evict page using **Base Algorithm A**. $\rightarrow$ (1-Consistency)

How GUARD Makes Decisions (Visual Flow)

1. **Cache Miss:** Request p_i misses.
2. **Check Phase Status:** Is $\mathcal{U}$ empty?
   - ✓ **Yes:** Start New Phase. $\mathcal{U} \leftarrow$ all pages.
   - × **No:** Continue current phase.
3. **Check for Prediction Error:** Has p_i been evicted in the current phase?
   - ✓ **Yes (prediction error!):**
     - Evict **random page** from $\mathcal{U}$. $\rightarrow$ (Robustness)
     - **Guard** p_i.
   - × **No:**
     - Evict page using **Base Algorithm A**. $\rightarrow$ (1-Consistency)

How GUARD Makes Decisions (Visual Flow)

1. **Cache Miss:** Request p_i misses.
2. **Check Phase Status:** Is $\mathcal{U}$ empty?
   - ✓ **Yes:** Start New Phase. $\mathcal{U} \leftarrow$ all pages.
   - × **No:** Continue current phase.
3. **Check for Prediction Error:** Has p_i been evicted in the current phase?
   - ✓ **Yes (prediction error!):**
     - Evict **random page** from $\mathcal{U}$. $\rightarrow$ (Robustness)
     - **Guard** p_i.
   - × **No:**
     - Evict page using **Base Algorithm A**. $\rightarrow$ (1-Consistency)

How GUARD Makes Decisions (Visual Flow)

1. **Cache Miss:** Request p_i misses.
2. **Check Phase Status:** Is $\mathcal{U}$ empty?
   - ✓ **Yes:** Start New Phase. $\mathcal{U} \leftarrow$ all pages.
   - × **No:** Continue current phase.
3. **Check for Prediction Error:** Has p_i been evicted in the current phase?
   - ✓ **Yes (prediction error!):**
     - Evict **random page** from $\mathcal{U}$. $\rightarrow$ (Robustness)
     - **Guard** p_i.
   - × **No:**
     - Evict page using **Base Algorithm A**. $\rightarrow$ (1-Consistency)

Our Contribution

A lightweight robustification framework, GUARD, that:

- Applies to any 1-**consistent** learning-augmented algorithm. (not limited to the RB-following algorithms defined in the paper)
- Achieves a **state-of-the-art** consistency–robustness trade-off:

$$(1,\, 2H_{k-1} + 2)$$

- Adds amortized $\mathcal{O}(1)$ computation per request.

Thank You

Code available at:
https://github.com/OptiSys-ZJU/cache-coliseum

Contact me:
naturechenpeng@gmail.com