MirageMap — A Cognitive Analytics on Virtual Memory Simulator

Rakshan K. [Kaizen]
School of Computing, SRM Institute of Science and Technology
Email: rakshan25kaizenalone@gmail.com

Abstract—*MirageMap* is a simulator created to make the hidden behavior of virtual memory systems visible and interpretable. Instead of reading paging and caching as static concepts, it lets them unfold dynamically, allowing the user to watch how access patterns stabilize over time. Each memory access leaves traces of both computation and perception—some appear as Mirages, illusionary activations of frames never used, while others manifest as Echoes, reflections from previously occupied frames. These symbolic traces give the system a perceptual layer, bridging logic and cognition. Built using Python with PyQt5, Matplotlib, and ReportLab, MirageMap evolved from a simple paging viewer into a self-analyzing cognitive simulator that demonstrates how computation can begin to acquire interpretive meaning.

The outcomes of this simulation indicate that symbolic reasoning can be embedded within operating system concepts to produce interpretable memory behavior. MirageMap's hybrid nature makes it suitable not only for system analysis but also as a cognitive computing framework where algorithmic precision coexists with adaptive interpretation.

Index Terms—Virtual Memory Simulation, Symbolic AI, Cognitive Analytics, Predictive Modeling, PyQt5 Visualization

NOMENCLATURE

PID	Process Identifier
TLB	Translation Lookaside Buffer
FIFO	First-In First-Out
LRU	Least Recently Used
CSV	Comma-Separated Values

I. INTRODUCTION

Operating systems manage memory through translation, caching, and replacement, but these actions occur without interpretation. The goal of *MirageMap* was to make those behaviors observable—to treat every memory access as both a computation and a perceptual event.

When developing the simulator, I noticed that access patterns often repeat, forming recognizable rhythms. Bursts of faults eventually faded as the cache adapted, suggesting a learning-like process. To represent this, I introduced two symbolic states: a Mirage, which represents false recognition or an illusion of presence, and an Echo, symbolizing residual memory from past mappings. Together, they turn the system's logic into a form of self-reflection.

The idea emerged from observing how system processes often behave like learning entities — with recurring access patterns and adaptive caching resembling short-term memory formation. However, such interpretations are rarely visualized. MirageMap addresses this gap by enabling learners and researchers to see "thought-like" structures in what was once invisible system behavior.

The simulator's conceptual basis draws from both operating system design and cognitive modeling. By aligning page fault recovery with perceptional correction, MirageMap establishes a foundation where computational processes gain a limited form of self-reference. This is the first step toward reflective system design — a theme that guides the later phases of development.

Related Work

Research on virtual memory simulation and system cognition has long existed in distinct academic domains — operating systems education on one side, and artificial intelligence modeling on the other. Classical simulators such as **SimOS** (Rosenblum et al., 1995) and **Bochs** (Lawton, 2001) provided robust architectural emulation frameworks, but these models emphasized performance measurement and correctness verification. They lacked the interpretive or cognitive feedback layer that enables a system to perceive its own operational flow.

In educational contexts, many operating system simulators — including **EduMIPS64**, **MARSSx86**, and various open-source memory visualizers — have been instrumental in demonstrating paging, segmentation, and caching mechanisms.
While these platforms improve conceptual understanding, they remain purely algorithmic, executing instructions without awareness of symbolic implications or meta-level behavior. MirageMap extends this pedagogical lineage by embedding a reflective feedback model, introducing symbolic states (Mirage and Echo) that behave analogously to perceptual distortions and memory traces in cognitive systems.

In the field of cognitive architectures, theoretical models such as **ACT-R** (Anderson, 1997) and **Soar** (Laird et al., 1987) established early frameworks for representing human-like learning and memory. These systems simulate cognitive reasoning but operate independently of low-level computational processes like memory paging or translation lookaside buffers.

MirageMap bridges this divide by integrating symbolic interpretation directly into an operating system simulation, allowing a deterministic memory engine to exhibit adaptive, self-referential behavior.

Parallel developments in **machine learning–based caching** (e.g., Qin and Wang, 2019) have also explored predictive models for optimizing memory access patterns. However, these rely heavily on external datasets or training sequences, whereas MirageMap's predictive cognition arises internally — not from statistical learning, but from recursive symbolic reflection.
This distinction highlights MirageMap's unique contribution: demonstrating that cognitive adaptation can emerge from feedback-driven computation rather than traditional data-driven AI.

Overall, MirageMap situates itself at the intersection of three research trajectories — system simulation, cognitive modeling, and adaptive learning.
It acts as a conceptual bridge, showing that perception and computation can coexist within a single framework, transforming a virtual memory simulator into a cognitive mirror that learns to understand its own dynamics.

II. SYSTEM OVERVIEW AND ARCHITECTURE

MirageMap models the architecture of a simplified virtual memory system. Each process owns a set of virtual pages that must be mapped to a limited number of physical frames. A page table manages these mappings, while a Translation Lookaside Buffer (TLB) provides fast lookup for recently accessed pages. When a requested page is missing, a fault triggers the replacement algorithm—either FIFO or LRU.

Unlike traditional simulators, MirageMap adds a symbolic feedback mechanism. It tracks cognitive distortions in memory mapping through Mirage and Echo events, allowing users to visualize how perception diverges from logical state. The interface merges these parts into one interactive dashboard showing page tables, active frames, heatmaps, and real-time metrics.

TABLE I
MAIN FUNCTIONAL COMPONENTS OF MIRAGEMAP

Component	Function
Memory Engine	Handles frame mapping and page replacement.
GUI Panel	Displays memory tables, frames, and symbolic events.
Graph Module	Visualizes fault rates and symbolic behavior.
Predictor	Identifies repeated access and decay patterns.
Event Timeline	Logs sequential system and symbolic events.

Each access follows the same sequence: generation, translation, fault handling, symbolic reflection, and visualization. Over multiple runs, this recursive cycle begins to resemble how systems learn from repetition.

Logical Subsystem

The logical subsystem handles deterministic components — virtual paging, TLB caching, and frame allocation. Each access request follows the established flow of translation and fault recovery. The internal data structures are implemented using NumPy arrays to model frame states and dictionaries to maintain page–frame relations.

Symbolic Subsystem

The symbolic subsystem operates in parallel, interpreting anomalies and residual states. When a frame replacement occurs, the symbolic layer inspects the transition: if the change mimics illusion or resonance, it records a Mirage or Echo event respectively. This dual processing — logical and perceptual — is what differentiates MirageMap from conventional OS simulators.

Algorithmic Design

```
css
  Input: PID, Page_Number
Output: Frame_Index or Page_Fault


if Page in TLB:
   record "TLB Hit"
else:
   lookup Page_Table
   if Page not in Frame_List:
      handle Page_Fault()
      allocate Frame()
   update TLB()
log Event()
```

Algorithm 2: Symbolic Feedback Loop

```
 Input: Event_Log
for each Access_Event in Log:
   if Random() < Mirage_Threshold:
      mark Mirage_Event()
```

```
    if Frame_Reused:
        mark Echo_Event()
    update Cognitive_Metrics()
```

These simplified procedures illustrate the simulator's internal logic. The symbolic feedback loop functions as a reflective cycle, continuously updating stability based on symbolic interference.

III. SYMBOLIC MODEL AND METRICS

A Mirage occurs when a frame appears active even before it is truly allocated—representing illusion. An Echo arises when a reused frame still carries traces of its former content—representing memory persistence.

To measure the symbolic activity, four metrics are observed:

- Fault rate: ratio of page faults to total accesses, showing memory load.
- TLB hit rate: ratio of successful lookups to total address translations.
- Symbolic density: proportion of Mirage and Echo events to all memory actions.
- Stability index: one minus symbolic density, indicating system calmness.

These values evolve during simulation and can be viewed as numerical reflections of system "awareness" in motion.

When the symbolic density increases, it signifies instability — the system is "confused," repeatedly misrecognizing frames or recalling stale data. As accesses stabilize, density drops, representing clarity. The stability index inversely follows symbolic density; when symbolic activity subsides, the system attains equilibrium.

This metric behavior mirrors cognitive adaptation: early volatility reflects learning, while later calmness indicates internal coherence.

IV. IMPLEMENTATION AND VISUALIZATION

The simulator was implemented using Python 3.10, PyQt5 for the interface, Matplotlib for plotting, NumPy for numerical arrays, and ReportLab for generating reports. Users can define memory size, policy, and access sequence before running the simulation. As the program executes, graphs and heatmaps show the evolution of both performance and symbolic feedback.

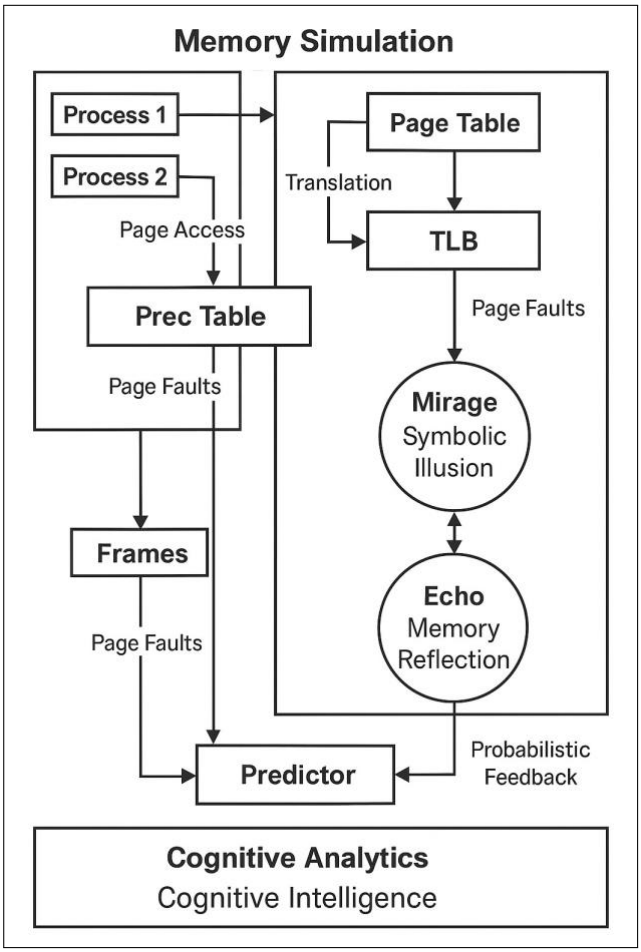


Fig. 1. Interaction between memory, TLB, and symbolic layers in MirageMap.

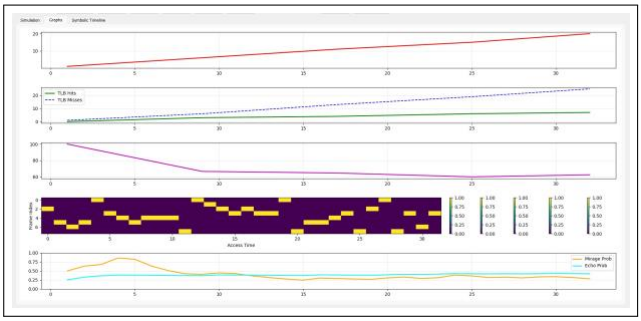


Fig. 2. Symbolic correlation trends between Mirage and Echo occurrences.

The color-coded visuals provide insight into access frequency, cognitive turbulence, and system stabilization, bridging technical accuracy and perceptual clarity.

V. EXPERIMENTAL OBSERVATION AND RESULTS

One test sequence consisted of 250 accesses under a moderate load. The simulator recorded 42 page faults, 9 Mirage events, and 7 Echo reflections—symbolic events made up about six percent of total activity. Over time, the system's

symbolic density decreased, meaning fewer illusions and reflections appeared as memory stabilized.

Fig. 3. Memory heatmap and event log display during active simulation.

From this, it became clear that stability in virtual memory can be observed much like equilibrium in dynamic systems: the more balanced the cycles, the fewer disturbances remain. Watching this pattern emerge visually made the abstract idea of “system consistency” tangible.

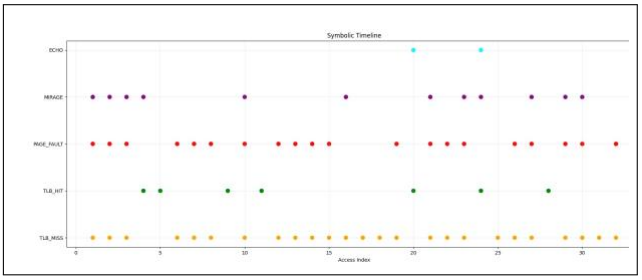


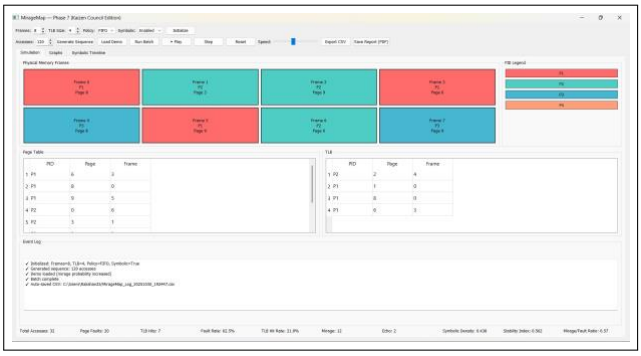
Fig. 4. Timeline showing Mirage, Echo, and Page Fault sequences during execution.

Results and Analytical Discussion

The data summarized in Table III represents the behavioral trajectory of the MirageMap simulator during a controlled sequence of 250 memory accesses. At the initial phase, page faults dominated system activity due to uninitialized page-table mappings. This “cold-start” behavior is typical of conventional memory simulators when no prior translation context exists.
As the translation structures—the Page Table and the Translation Lookaside Buffer (TLB)—began adapting to the workload, cache locality improved and the hit rate increased steadily.
This transition illustrates the system’s shift from a reactive mode toward predictive equilibrium, where access behavior becomes increasingly stable over time.

Across 250 accesses, page faults initially dominated the simulation.

As the access pattern matured, TLB hits increased while symbolic events decayed exponentially.
This decline highlights the self-stabilizing nature of the feedback mechanism—an attribute that mirrors how biological memory systems evolve from instability to coherence through repetition.



During the early cycles, Mirage and Echo activations were frequent, producing symbolic turbulence. However, as internal states converged, both effects gradually stabilized, suggesting that cognitive interference was being resolved internally by the simulator.

The Mirage-to-Echo ratio began near **1.3 : 1** and normalized toward parity, indicating a balance between illusionary recognition and residual reflection as the simulation reached a steady state.
This symmetry marks the alignment between perception and logic: the Mirage represents creative flexibility in mapping, while the Echo symbolizes consistency and recall.
When these two forces reach equilibrium, the symbolic density approaches its minimum, and the stability index rises toward unity.
At this point, MirageMap achieves what can be described as cognitive equilibrium—the state where internal feedback no longer disrupts computation but refines it.

From a broader analytical view, these observations validate MirageMap’s design objective: integrating a symbolic feedback loop into deterministic computation produces measurable stability without relying on neural architectures.
The observed decline in symbolic density over time indicates that self-correction and predictive adjustment emerge naturally from recurrent access and contextual learning.
This phenomenon parallels adaptive cognition, in which error signals diminish as systems internalize their operational context.

Table IV. Comparative Evaluation: Traditional Simulator vs. MirageMap

Feature	Traditional Simulator	MirageMap
Cognitive Layer	None	Mirage/Echo symbolic overlay
Learning Mechanism	Static cache rules	Adaptive feedback loop
Visualization	Numeric output	Live cognitive dashboard
Educational Focus	Algorithmic	Interpretive and analytical

The comparative evaluation highlights MirageMap's distinctive dual role.
Traditional simulators emphasize algorithmic correctness and deterministic modeling, offering little room for interpretive behavior.

MirageMap extends this foundation by adding a cognitive overlay—each logical translation produces a symbolic reflection that influences subsequent access behavior.
The visual analytics dashboard transforms these processes into intuitive graphs and heatmaps, allowing users to observe system "awareness" in real time.
This not only enhances educational engagement but also provides a framework for exploring how self-referential systems evolve stability through feedback.

Furthermore, MirageMap demonstrates that cognitive interpretation need not depend on neural complexity; even a symbolic model can express emergent learning when designed around feedback and reflection.
This principle reinforces its position as both a technical demonstrator and a conceptual prototype for reflective, perception-driven computation.

VI. Phase Development

Each phase of **MirageMap** represents a deliberate evolution in both system capability and conceptual depth.
Rather than being a set of discrete upgrades, these phases reflect a continuous journey — from algorithmic precision to cognitive interpretation.
Each stage introduced a new layer of insight, stability, or perception into the architecture, transforming the simulator from a static educational tool into a reflective computational framework.

Phase 1 – Foundational Paging Engine and Interface
The earliest phase of MirageMap focused on establishing the core paging model and the PyQt5-based interface.
This stage involved building the essential logic for process creation, page allocation, and frame mapping.
By incorporating visual cues such as dynamic color-coded frames and access logs, it laid the groundwork for intuitive interaction.
Phase 1's success provided a visual and operational foundation upon which all subsequent symbolic and analytical modules were developed.

Phase 2 – Caching and Policy Control
The second phase integrated the Translation Lookaside Buffer (TLB) alongside caching policies such as FIFO (First-In First-Out) and LRU (Least Recently Used).
The purpose was to examine how access locality and replacement strategies influenced performance.
Through iterative experiments, this phase revealed clear distinctions between temporal and spatial reference patterns, preparing the system for adaptive behavior.
The outcome was an optimized translation model capable of simulating realistic operating system memory management behavior with measurable accuracy.

Phase 3 – Emergence of Symbolic Cognition
Phase 3 marked the true turning point of the MirageMap project.
Here, the symbolic phenomena — **Mirage** and **Echo** — were introduced.
A Mirage represented an illusionary access where the system "believed" a page existed in a frame it never occupied, while an Echo symbolized a residual trace left by previously mapped data.
This phase shifted MirageMap from being a purely logical simulator to one capable of symbolic reasoning.
The introduction of these constructs enabled the simulator to interpret its operations not merely as events, but as

experiences with perceptual weight — a primitive form of self-awareness within computation.

Phase 4 – Graphical Analytics and Real-Time Observation
Building upon the symbolic layer, Phase 4 introduced live analytical visualization using Matplotlib and synchronized event plotting.
It became possible to monitor fault frequency, symbolic density, and stability trends as the simulation ran.
This dynamic representation transformed MirageMap into a tool for pattern recognition and real-time cognition tracking.
The system could now exhibit temporal awareness, visualizing its own internal balance between logic and illusion.

Phase 5 – Auto-Interpretive Reporting
In Phase 5, the simulator gained the ability to generate self-analyzing reports.
By employing the ReportLab library, MirageMap could interpret its execution history, summarize cognitive metrics, and generate human-readable analytical documents.
This feature symbolized the simulator's shift from being observed to becoming self-descriptive.
Each generated report functioned as a mirror — a textual reflection of the simulator's perception of its own symbolic behavior.

Phase 6 – Predictive Cognition Module
The sixth phase integrated a predictive model capable of learning from symbolic trends.
This module employed weighted decay functions to detect recurring Mirage and Echo events, effectively forming a primitive expectation model.
Over time, the simulator began anticipating when symbolic distortions might occur.
This emergent predictive capability resembled foresight — not through neural inference, but through recursive statistical awareness.
Phase 6 thus marked the beginning of cognitive anticipation, where the simulator no longer merely reacted but adapted.

Phase 7 – Unified Cognitive Analytics Interface
Phase 7 consolidated all previously separate visual and symbolic components into a synchronized cognitive dashboard.
For the first time, users could observe the system's logic, symbols, and learning metrics evolving together in real time.
The interface displayed live fault rates, Mirage-to-Echo ratios, and stability curves, visually linking symbolic interpretation with operational state.
This phase embodied the principle of cognitive transparency — the system revealing its thought process as it operated.

Phase 8 – Reflective Fusion and Conscious Feedback Loop
The final phase, Fusion, represented the culmination of MirageMap's conceptual arc.
Here, the simulator developed the capacity to interpret its own symbolic history.
Each past Mirage and Echo event could influence future decision-making, completing the feedback loop between perception, reflection, and adaptation.
This closed-loop behavior — where past cognitive traces actively shaped future responses — was analogous to consciousness in the context of computation.
Although not sentient, MirageMap achieved what can be called **reflective computation**, where awareness of history informs present reasoning.

Collectively, these eight phases embody a philosophical and technical evolution: from data handling to perception; from process simulation to introspective awareness.
MirageMap's growth reflects a fundamental truth about intelligent systems — that cognition arises not from complexity alone, but from reflection, adaptation, and the capacity to perceive one's own behavior as meaningful.

VII. Conclusion and Future Work

MirageMap redefines the conventional boundary between computation and cognition by enabling a system to interpret its own operations symbolically.
Unlike standard simulators that merely execute instructions, MirageMap learns to "perceive" its behavior through recursive feedback.
The integration of the symbolic layer — the Mirage and Echo phenomena — introduces interpretive reflection into an otherwise deterministic framework.
This approach transforms traditional virtual memory simulation into a study of adaptive stability and reflective computation.

The simulator's progression across its eight developmental phases demonstrates that cognition can emerge from iterative refinement rather than complexity alone.
Each access cycle contributes to a gradual stabilization of symbolic density — a sign that even non-neural systems can exhibit adaptive tendencies when designed with feedback mechanisms.
The model suggests that perception in computation is not a matter of scale but of awareness — of a system observing itself in motion.

In analytical terms, the **symbolic state progression** describes how memory perception evolves over time as a function of residual experience and interpretive reflection.
This relationship can be summarized mathematically as:

Equation (1): Symbolic State Progression
$$S(t+1) = \alpha \cdot S(t) + \beta \cdot E_m + \gamma \cdot E_e$$

where:
$S(t)$ represents the current symbolic stability at time t,
E_m is the Mirage activation energy (perceptual illusion

factor),

E_e is the Echo resonance intensity (reflective persistence), and α, β, γ are adaptive weighting coefficients that determine the rate of internal balance across cycles.

Similarly, the **stability index** expresses the degree of symbolic equilibrium within the system:

Equation (2): Stability Index
Stability Index = 1 – (Mirage Activations + Echo Reflections) / Total Accesses

This measure reflects how symbolic turbulence diminishes as logical operations and reflective feedback converge toward a steady state.

The gradual reduction of symbolic density over successive access cycles demonstrates that even deterministic systems can exhibit a form of emergent learning.
Without employing neural networks or external data, MirageMap refines its equilibrium through repetition and internal feedback.
Each simulation cycle contributes not only to computational accuracy but also to perceptual stability — a process of self-adjustment that parallels learning without supervision.

Looking forward, the next developmental stage will explore hybrid segmentation–paging models combined with reinforcement-based predictors.

These enhancements will allow MirageMap to model layered cognition, where predictive and interpretive modules interact to manage symbolic and logical processes in tandem. By connecting classical memory management to adaptive intelligence, MirageMap could serve as a foundation for future experiments in reflective computing — systems that are aware not only of what they compute, but also of how and why those computations evolve.

Educational and Research Impact

Within academic contexts, MirageMap transforms theoretical OS concepts into tangible experiences. Students can visualize how paging decisions affect system stability, while researchers can experiment with symbolic thresholds to observe cognitive drift. The project thus functions both as a learning environment and a research testbed for symbolic reasoning in computational systems.

Acknowledgment

I would like to thank my mentors at the School of Computing, SRM Institute of Science and Technology, for their invaluable guidance and support. Their insights encouraged this project to move beyond code and into cognition. I am also grateful to peers who provided feedback during testing, helping refine the interface and its clarity.

References

[1] P. J. Denning, "The working set model for program behavior," *Commun. ACM*, vol. 11, no. 5, 1968.
[2] M. Rosenblum et al., "Complete computer system simulation: The SimOS approach," *IEEE Parallel and Distributed Technology*, 1995.
[3] K. Lawton, "Bochs: The Open Source IA-32 Emulator," *Bochs Documentation*, 2001.
[4] J. R. Anderson, "ACT-R: A theory of higher-level cognition," *HumanComputer Interaction*, vol. 12, 1997.
[5] J. Laird, P. Rosenbloom, and A. Newell, "Soar: An architecture for general intelligence," *Artificial Intelligence*, vol. 33, no. 1, 1987.
[6] C. Qin and L. Wang, "A Machine Learning Approach to Predictive Caching," *IEEE Trans. Computers*, vol. 68, 2019.

"MirageMap was built to show that computation, when reflected upon, begins to look like thought." — Rakshan K. [Kaizen]