From Minutes to Seconds: Redefining the Five-Minute Rule for AI-Era Memory Hierarchies

Tong Zhang
ScaleFlux
USA

Vikram Sharma Mailthody
NVIDIA
USA

Fei Sun
ScaleFlux
USA

Linsen Ma
ScaleFlux
USA

Chris J. Newburn
NVIDIA
USA

Teresa Zhang
Stanford University
USA

Yang Liu
ScaleFlux
USA

Jiangpeng Li
ScaleFlux
USA

Hao Zhong
ScaleFlux
USA

Wen-Mei Hwu
NVIDIA
USA

Abstract

In 1987, Jim Gray and Gianfranco Putzolu introduced the five-minute rule, a simple, storage-memory-economics-based heuristic for deciding when data should live in DRAM rather than on storage. Subsequent revisits to the rule largely retained that economics-only view, leaving host costs, feasibility limits, and workload behavior out of scope. This paper revisits the rule from first principles, integrating host costs, DRAM bandwidth/capacity, and physics-grounded models of SSD performance and cost, and then embedding these elements in a constraint- and workload-aware framework that yields actionable provisioning guidance. We show that, for modern AI platforms, especially GPU-centric hosts paired with ultra-high-IOPS SSDs engineered for fine-grained random access, the DRAM↔flash caching threshold collapses from minutes to a few seconds. This shift reframes NAND flash memory as an *active data tier* and exposes a broad research space across the hardware–software stack. We further introduce *MQSim-Next*, a calibrated SSD simulator that supports validation and sensitivity analysis and facilitates future architectural and system research. Finally, we present two concrete case studies that showcase the software system design space opened by such memory hierarchy paradigm shift. Overall, we turn a classical heuristic into an actionable, feasibility-aware analysis and provisioning framework and set the stage for further research on AI-era memory hierarchy.

1 Introduction

The design of data management systems has long been influenced by the evolving capabilities and economics of storage hardware. In the 1980s, database engines relied on a two-tier hierarchy: DRAM and hard disk drives (HDDs). At the time, DRAM cost approximately $120/KB, compared to $0.10/KB for HDDs, prompting a central question: *when is it economically worthwhile to cache data in memory rather than fetch it from disk?* To answer this question,

Jim Gray and Gianfranco Putzolu introduced the *five-minute rule* in 1987 [19], a simple guideline stating that 1KB records accessed every five minutes should be kept in DRAM. Their formulation computes the *break-even access interval* at which the "rent" of caching data in DRAM equals the cost of fetching data from disk drives. Over the decades, this rule has been continuously revisited (most notably in 1997 [18], 2007 [17], and 2019 [5]) to account for the technological advancement, especially the advent of solid-state drives (SSDs) in the 2010s. Notably, the 2019 analysis concluded that a minute-scale threshold still holds between DRAM and SSD, echoing the now-familiar adage: *"Tape is dead, disk is tape, flash is disk."* Yet these studies largely retained the original economics-only perspective, omitting host-side costs, feasibility constraints, and workload characteristics; as a result they offered little practical guidance for provisioning real systems.

Fast forward to 2025, and the storage landscape is undergoing another dramatic shift. The explosive growth of AI workloads is driving demand for petabyte-scale working sets and increasingly diverse data access patterns. This has led to major industrial trends, such as work discussed in NVIDIA's Storage-Next research efforts [37, 41, 43], on unlocking the full potential of NAND flash memory, not just as a capacity tier, but as a high-throughput, cost-effective extension of memory. In response, SSD vendors are making substantial R&D investments in developing **Storage-Next SSDs** that promise a 10× increase in IOPS per dollar, with scalable IOPS behavior that boosts performance as access block sizes shrink (e.g., 50M IOPS at 512B, 10M IOPS at 4KB) [12, 38]. In parallel, the HBF (high-bandwidth flash) initiative [44] recently announced by SanDisk and SK hynix sets a target of 1TB/s throughput per NAND flash stack, signaling a clear industry roadmap toward flash devices with bandwidth approaching that of HBM. Unlike past waves of non-volatile memories (NVM) that faced fundamental material/device roadblocks, these efforts build on mature NAND flash technology, offering a plausible path to elevate NAND flash memory from a capacity tier toward an active tier of the memory hierarchy.

To reason about this trajectory, we revisit and redefine the five-minute rule from first principles. Our approach calibrates the caching decision with physics- and architecture-grounded inputs (host costs and device behavior), incorporates feasibility constraints (host IOPS capacity and DRAM bandwidth/capacity), and folds in workload characteristics (access-interval profiles and service-level targets). This yields a unified framework that (i) quantifies the trade-offs among DRAM bandwidth and capacity, host IOPS, and SSD throughput, (ii) translates those trade-offs into concrete provisioning choices across platforms and workload characteristics, and (iii) provides clear viability thresholds together with minimal-upgrade guidance for practitioners. Using this framework, we show that, under realistic architectural and device constraints, the DRAM-flash caching threshold can **collapse from minutes to just a few seconds**, effectively reshaping how memory and storage should be provisioned for modern AI workloads. To underpin our framework, we further develop *MQSim-Next*, a calibrated SSD simulator built upon MQSim [9, 46], which can be used for validation and sensitivity studies and to facilitate future architectural and system research. Building on this foundation, we finally present two specific case studies on large-scale key-value (KV) store and approximate nearest neighbor (ANN) search as initial steps toward exploring the exciting software design space opened by such memory hierarchy paradigm shift.

Together, these results argue for rethinking the memory hierarchy in the AI era by elevating NAND flash from passive storage to an active tier, and provide architects with a practical toolkit for provisioning and co-design across devices, hosts, and applications. Its major contributions are further summarized as follows:

- A **first-principle reformulation** of the five-minute rule that integrates host costs, device behavior, and DRAM bandwidth/capacity.
- A **constraint-aware refinement** that bounds usable SSD throughput via host IOPS capacity and tail-latency targets, replacing datasheet peaks with feasibility-aware IOPS.
- A **workload-aware platform framework** that combines access-interval profiles and service-level targets with system constraints to yield viability analysis and *actionable* provisioning guidance.
- An **empirical finding** that GPU-centric hosts paired with Storage-Next SSDs can shrink the DRAM-flash caching threshold from minutes to seconds, together with guidance on when host-side limits dominate.
- **MQSim-Next**, a calibrated, physics-grounded SSD simulator used to validate model assumptions and support future architectural research in this space.
- Two **illustrative case studies** presented as initial steps for exploring the vast software/algorithm research space enabled by seconds-scale caching.

This paper is organized as follows: Section 2 reviews the background and states the research questions. Sections 3–5 develop and validate the first-principles, workload-aware framework. Section 6 presents the MQSim-Next SSD simulator, and Section 7 presents the two case studies.

2 Background and Motivation

The five-minute rule is a simple heuristic for data placement (i.e., keep a page in DRAM if it is cheaper than fetching it from storage) but in practice it rarely guides provisioning. In its *economics-only* form, it ignores host-side I/O costs and relies on vendor specs; beyond economics, it omits feasibility limits such as finite processor IOPS, latency/throughput targets, and DRAM capacity/bandwidth. We briefly recap the classical rule, identify these gaps, and pose the research questions that motivate this work.

2.1 The Classical 5-Minute Rule (brief recap)

The rule makes a page-level decision: keep a page in DRAM when it is cheaper than fetching it on demand from storage. This is captured by a *break-even interval*: if the expected inter-reference time is below the interval, keep the page in memory; otherwise, place it on storage. Balancing the "rent" on extra DRAM against saved storage accesses yields:

$$\tau_{\text{break-even}} = \left(\frac{\text{\# Pages per MB}}{\text{Storage Drive IOPS}} \right) \times \left(\frac{\text{Storage Drive Cost}}{\text{Cost of 1MB DRAM}} \right). \quad (1)$$

Intuitively, the formulation states that, to keep a page of data in DRAM, the rent should be less than the cost of moving the page from the storage device based on its IOPS/\$ spec. This storage-memory-economics-only view ignores host-side costs and leans on vendor peak specs, while overlooking practical limits as detailed next in Section 2.2.

2.2 Limits of the Classical Formulation

(A) Insufficient realism. The classical rule treats host resources as free. In practice, issuing and completing I/O consumes CPU time, interrupts/polls, and DRAM channel bandwidth. Such costs were negligible for HDDs (up to 200 IOPS) but not for modern SSDs (multi-million IOPS). Prior revisits of the rule [5, 17–19] also lean on vendor peak specs, overlooking architectural factors (e.g., NAND physics, internal parallelism, and block-size effects) that determine sustained behavior and cost.

(B) Economics-only, missing feasibility. Optimizing only device prices (e.g., \$/GB, \$/IOPS) cannot certify deployability. Feasibility depends on constraints the price-only view omits: the host's achievable I/O submission/completion rate, application latency/throughput targets, and DRAM capacity/bandwidth. Ignoring these constraints can recommend configurations that cannot meet intended workloads or service quality.

Summary. These gaps make the classical rule non-actionable for design and provisioning. We therefore develop a feasibility-aware framework that models host resource usage and enforces practical system constraints, yielding accurate, actionable guidance.

2.3 Research Questions

Our goal is to turn the classical rule from a heuristic into a basis for concrete provisioning, guided by the following three questions:

• **RQ1 (calibrated economics).** Retain the economic view but make it realistic by explicitly modeling host resource usage and first-principles SSD behavior (rather than datasheet peaks). How does the break-even interval change under this calibrated model?

• **RQ2 (constraint-aware refinement).** Add feasibility constraints (i.e., processor I/O submission/completion capacity and application latency targets). How do these constraints reshape the break-even interval, and when do they dominate price-only estimates?

• **RQ3 (platform viability and guidance).** Integrate DRAM bandwidth/capacity limits with the workload's access-interval profile. Can a unified framework fusing *economics*, *workload*, and *hardware constraints* determine viability, identify the economics-optimal configuration, and, when needed, recommend upgrades?

Findings from the preceding questions indicate that the DRAM-flash caching threshold has collapsed into the seconds regime due to the drastic elevation of IOPS/\$ of the storage drives. As a result, the long-standing boundary between memory and storage blurs, leading to the following research question:

• **RQ4 (re-think data-intensive software)**: As the DRAM−flash threshold drops to seconds, how should we re-think the design of data-intensive software, and what principles should steer a re-architecture of data structures, access paths, scheduling, and consistency to deliberately exploit such a new paradigm shift for throughput, efficiency, scalability, and cost?

Addressing these four questions shapes the remainder of this paper. It establishes a unified economics/feasibility framework with interpretable metrics for provisioning and upgrades, and it opens a principled design-space exploration under seconds-scale DRAM-flash caching.

2.4 Discussion of Assumptions and Scope

All the modeling parameters in this study are derived from mature NAND flash technology and established interface roadmaps, in contrast to prior explorations that hypothesize active-memory roles for emerging NVMs. We assume that the SSD controller is adequately provisioned, so limits arise from fundamental device physics, interface bandwidth, and/or host capacity. Our goal is not to forecast product specifics, but to examine how feasibility and provisioning change once the full IOPS potential of NAND flash is unleashed. The framework is forward-looking yet physically grounded, and can be re-parameterized as devices and standards evolve.

3 Calibrated Economic Model (RQ1)

In this section, we ground the break-even rule in a calibrated economics view: define a first-order host–device model, make host I/O costs explicit, and replace datasheet peaks with architecture-derived SSD IOPS from a first-principles device model. We then derive the SSD terms and present a quantitative case study showing that, under realistic configurations, GPUs paired with Storage-Next SSDs have shrunk the DRAM-flash break-even from minutes toward seconds; feasibility limits (processor IOPS and latency targets) are added in Section 4.

3.1 System Model and Calibrated Economic Break-even

Fig. 1 presents a first-order host–device view of the I/O path: a host processor (CPU or GPU), a directly attached, multi-channel DRAM subsystem, and one or more NVMe SSDs. We assume an optimal zero-copy read path [24] to minimize host DRAM bandwidth usage:

each read incurs a single transfer over the host↔DRAM interface, with no extra kernel↔user copies. Now consider an l_{blk}-byte block accessed periodically with reuse interval τ_{intvl}. Absent caching in host DRAM, the system repeatedly retrieves this block from the SSD, incurring cumulative cost across the following three components:
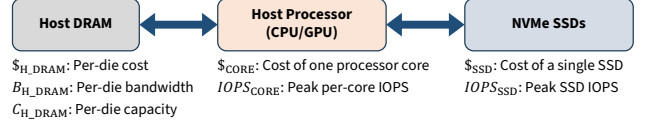

Figure 1: Simplified system architecture used to derive the new break-even interval formulation.

Host processor cost: Each I/O involves driver-level work such as queue management, DMA setup, and interrupt handling. Given the per-core IOPS capacity of $IOPS_{\text{CORE}}$[1] and per-core cost \$$_{\text{CORE}}$, we can express the cost on host processor as $\frac{\$_{\text{CORE}}}{IOPS_{\text{CORE}}} \cdot \frac{1}{\tau_{\text{intvl}}}$.

Host DRAM bandwidth cost: Each I/O transfers l_{blk} bytes into host DRAM, consuming the DRAM bandwidth. We model its cost as $\frac{l_{\text{blk}} \$_{\text{H\_DRAM}}}{B_{\text{H\_DRAM}}} \cdot \frac{1}{\tau_{\text{intvl}}}$, which is appropriate for bandwidth-bound systems common in modern AI infrastructure. If bandwidth is ample and capacity is the constraint, a capacity-based DRAM cost model would be more appropriate.

SSD access cost: Given its cost of \$$_{\text{SSD}}$ and peak random IOPS of $IOPS_{\text{SSD}}$, we can express the access cost on SSD as $\frac{\$_{\text{SSD}}}{IOPS_{\text{SSD}}} \cdot \frac{1}{\tau_{\text{intvl}}}$.[2]

Therefore, if we cache a data block being accessed with an interval of τ_{intvl} in DRAM, we can express the saved cost as:

$$\$_{\text{saving}} = \left(\frac{\$_{\text{CORE}}}{IOPS_{\text{CORE}}} + \frac{l_{\text{blk}} \cdot \$_{\text{H\_DRAM}}}{B_{\text{H\_DRAM}}} + \frac{\$_{\text{SSD}}}{IOPS_{\text{SSD}}} \right) \cdot \frac{1}{\tau_{\text{intvl}}} . \quad (2)$$

By caching the block in host DRAM, the system avoids this recurring cost. However, doing so requires reserving a portion of host DRAM capacity over time, which incurs a "rent":

$$\$_{\text{rent}} = \frac{l_{\text{blk}}}{C_{\text{H\_DRAM}}} \cdot \$_{\text{H\_DRAM}}. \quad (3)$$

The break-even interval $\tau_{\text{break-even}}$ is the access interval at which the memory rent equals the cost saved by avoiding repeated I/O operations. Solving $\$_{\text{rent}} = \$_{\text{saving}}$ yields:

$$\tau_{\text{break-even}} = \left(\frac{\$_{\text{CORE}}}{IOPS_{\text{CORE}}} + \frac{l_{\text{blk}} \cdot \$_{\text{H\_DRAM}}}{B_{\text{H\_DRAM}}} + \frac{\$_{\text{SSD}}}{IOPS_{\text{SSD}}} \right) \cdot \frac{C_{\text{H\_DRAM}}}{l_{\text{blk}} \cdot \$_{\text{H\_DRAM}}} . \quad (4)$$

This calibrated formulation preserves Gray's intuition: balance the DRAM "rent" against the cost of serving accesses from storage. It (i) explicitly charges I/O-induced host resources, and (ii) replaces datasheet peaks with SSD performance and cost derived from device behavior. Thus, it offers a more accurate economic criterion for deciding when DRAM caching is justified. In this model, $IOPS_{\text{SSD}}$

---

[1]For simplicity, we assume $IOPS_{\text{CORE}}$ is independent of the workload's read-to-write ratio; processing read and write requests incurs similar processor overhead.

[2]Consistent with the classical economic-only view, we assume the host can fully utilize the SSD's peak random IOPS for given data access block size and read-to-write ratio. Later sections incorporate hardware and workload constraints that bound usable IOPS and can change the effective cost.

and $\$_{SSD}$ are parameters rather than constants; Section 3.2 derives them from first principles using a device-level SSD model.

3.2 First-Principles SSD Modeling

We adopt a first-principles model of SSD performance and cost grounded in internal architecture and NAND device characteristics, enabling parametric exploration of how design and device choices shape IOPS and cost. As shown in Fig. 2, an SSD comprises a controller, SSD-internal DRAM for the FTL, and a NAND subsystem. The channel command time τ_{CMD} is the bus occupancy per read/write command; in conventional NAND flash with a shared 8-bit command/data bus $\tau_{CMD} \approx 1.2\,\mu s$ [39], while recent NAND flash applies SCA I/O protocol [32] to reduce τ_{CMD} to 100–200 ns, raising effective bandwidth. We model performance from sensing, programming, and command latencies, which dominate sustained service rate. Erase latency is omitted because each erase clears multiple megabytes, making its per-access cost negligible in steady state. For additional background on SSD and NAND flash, see [10, 35].
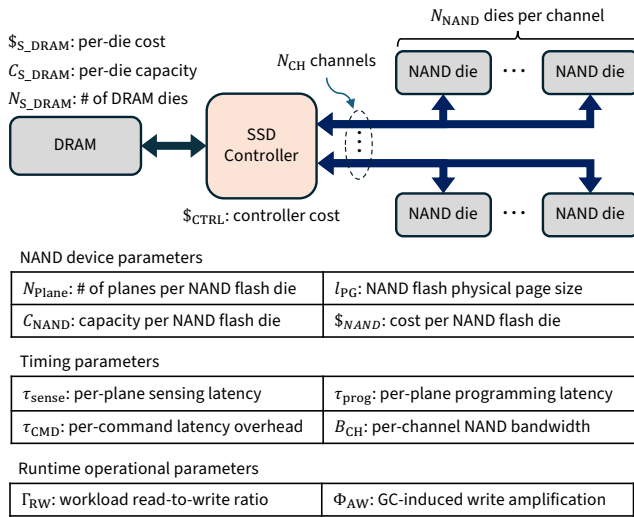

$\$_{S\_DRAM}$: per-die cost
C_{S_DRAM}: per-die capacity
N_{S_DRAM}: # of DRAM dies

N_{NAND} dies per channel

N_{CH} channels

DRAM

SSD Controller

NAND die ⋯ NAND die

NAND die ⋯ NAND die

$\$_{CTRL}$: controller cost

NAND device parameters

N_{Plane}: # of planes per NAND flash die	l_{PG}: NAND flash physical page size
C_{NAND}: capacity per NAND flash die	$\$_{NAND}$: cost per NAND flash die

Timing parameters

τ_{sense}: per-plane sensing latency	τ_{prog}: per-plane programming latency
τ_{CMD}: per-command latency overhead	B_{CH}: per-channel NAND bandwidth

Runtime operational parameters

Γ_{RW}: workload read-to-write ratio	Φ_{AW}: GC-induced write amplification

Figure 2: SSD architecture with key parameters for modeling performance and cost.

In our first-principle model, the SSD controller is provisioned so that performance limits arise *only* from the NAND flash devices. For tractability, all host-issued requests use the same block size l_{blk}. Let $IOPS_{NAND}^{(peak)}$ denote the maximum IOPS deliverable by a single NAND die, and let $IOPS_{CH}^{(peak)}$ denote the maximum IOPS sustainable by a single channel. With N_{CH} channels and N_{NAND} dies per channel, the overall peak SSD IOPS is

$$IOPS_{SSD}^{(peak)} = \frac{\Gamma_{RW}+1}{\Gamma_{RW}+2\Phi_{AW}-1} \cdot N_{CH} \cdot \min\left(N_{NAND} \cdot IOPS_{NAND}^{(peak)},\ IOPS_{CH}^{(peak)}\right), \quad (5)$$

where Γ_{RW} denote the read-to-write ratio and $\Phi_{AW} \geq 1$ captures write-amplification caused by background garbage collection (GC).

To derive the per-die peak IOPS $IOPS_{NAND}^{(peak)}$, note that a physical page must be programmed as a unit, so the controller coalesces host random writes into full-page sequential writes. Thus, within one program interval τ_{prog}, a die can commit $N_{Plane} \cdot l_{PG}/l_{blk}$

data blocks. For reads, within one sense interval τ_{sense}, a die can fetch $N_{Plane}/\lceil l_{blk}/l_{pg}\rceil$ data blocks. Combining the workload read-to-write ratio Γ_{RW} and SSD-internal write amplification Φ_{AW}, we have that, among total read/write activities over NAND flash memory, the read fraction is $R_r = (\Gamma_{RW} + \Phi_{AW} - 1)/(\Gamma_{RW} + 2\Phi_{AW} - 1)$ and the write fraction is $R_w = \Phi_{AW}/(\Gamma_{RW} + 2\Phi_{AW} - 1)$. Hence, the per-die peak IOPS is

$$IOPS_{NAND}^{(peak)} = R_r \cdot \frac{N_{Plane}}{\tau_{sense} \cdot \lceil l_{blk}/l_{pg}\rceil} + R_w \cdot \frac{N_{Plane}\, l_{PG}}{\tau_{prog}\, l_{blk}}. \quad (6)$$

Next we derive the channel-sustained peak IOPS $IOPS_{CH}$. With channel bandwidth B_{CH}, reading a size-l_{blk} block occupies the channel for $\tau_R = \tau_{CMD} + l_{blk}/B_{CH}$, so one channel can deliver up to $1/\tau_R$ random-read IOPS. A program transfers a full physical page of size l_{PG}, occupying the channel for $\tau_W = \tau_{CMD} + l_{PG}/B_{CH}$. Each page program commits l_{PG}/l_{blk} data blocks, hence each channel can support up to $l_{PG}/(l_{blk} \cdot \tau_W)$ random-write IOPS. Therefore, the peak IOPS sustainable by each channel is

$$IOPS_{CH}^{(peak)} = R_r \cdot \frac{1}{\tau_{CMD} + \frac{l_{blk}}{B_{CH}}} + R_w \cdot \frac{1}{\frac{l_{blk}}{l_{PG}}\tau_{CMD} + \frac{l_{blk}}{B_{CH}}}. \quad (7)$$

As shown in Fig. 2, the total SSD cost aggregates the controller, the population of NAND dies across all channels, and the SSD-internal DRAM used primarily for the FTL mapping table:

$$\$_{SSD} = \$_{CTRL} + N_{CH} \cdot N_{NAND} \cdot \$_{NAND} + N_{S\_DRAM} \cdot \$_{S\_DRAM}. \quad (8)$$

Here N_{S_DRAM} is the number of SSD-internal DRAM dies. If each FTL entry requires b_{FTL} bytes (e.g., 4–8 bytes) and the minimum access granularity is 512B, the maximum FTL size is

$$C_{FTL} = \frac{N_{CH} \cdot N_{NAND} \cdot C_{NAND}}{512B} \cdot b_{FTL}. \quad (9)$$

Given the capacity per DRAM die C_{S_DRAM}, the required SSD-internal DRAM die count is

$$N_{S\_DRAM} = \left\lceil \frac{C_{FTL}}{C_{S\_DRAM}} \right\rceil = \left\lceil \frac{N_{CH} \cdot N_{NAND} \cdot C_{NAND} \cdot b_{FTL}}{512B \cdot C_{S\_DRAM}} \right\rceil. \quad (10)$$

These formulations make explicit how SSD IOPS and cost scale with architectural choices and operating parameters, and they expose the coupling between performance and capacity. Unlike vendor specifications that reflect a few fixed configurations, this first-principles model supports architecture-aware reasoning across a broad design space and workload settings.

3.3 Quantitative Study

We demonstrate the framework's utility with a quantitative study of break-even intervals across realistic system configurations. Table 1 summarizes SSD parameters for three NAND types: (i) SLC devices optimized for low latency and high IOPS (e.g., Kioxia XL-Flash [45] and Samsung Z-NAND [8]); (ii) TLC operated in pseudo-SLC (pSLC) mode; and (iii) standard TLC.
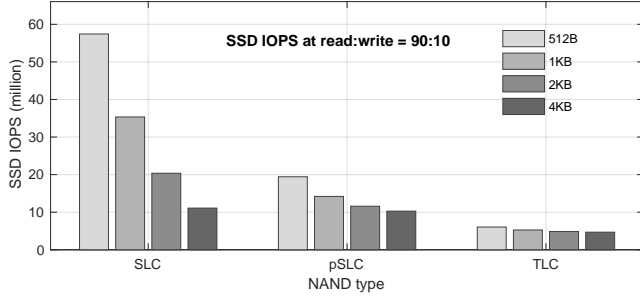
In this study, we fix $\Gamma_{RW} = 90{:}10$, reflecting read-heavy AI workloads, and conservatively set $\Phi_{AW} = 3$. Fig. 3 reports peak IOPS for SLC, pSLC, and TLC across 512B–4KB blocks. As Eq. 5 indicates, overall SSD IOPS is bounded by the smaller of the device limit $IOPS_{NAND}^{(peak)}$ and the channel limit $IOPS_{CH}^{(peak)}$. The device term depends mainly on τ_{sense} and τ_{prog} and varies only weakly with l_{blk}; the channel term depends strongly on l_{blk} and, with small

Table 1: Key SSD parameters.

	τ_{sense}	τ_{prog}	l_{PG}	N_{Plane}	C_{NAND}
SLC	$5\,\mu s$	$50\,\mu s$	4KB	6	32GB
pSLC	$20\,\mu s$	$150\,\mu s$	16KB	4	42GB
TLC	$40\,\mu s$	1ms	16KB	4	128GB

τ_{CMD}	B_{CH}	N_{CH}	N_{NAND}	$C_{\text{S\_DRAM}}$
150ns	3.6GB/s	20	4	3GB

τ_{CMD}, scales roughly as $B_{\text{CH}}/l_{\text{blk}}$. For TLC, long τ_{sense} and τ_{prog} keep $IOPS_{\text{NAND}}^{(\text{peak})}$ low, so the device side limits IOPS for all l_{blk}, producing only slight variation with block size. For SLC, very short τ_{sense} and τ_{prog} raise $IOPS_{\text{NAND}}^{(\text{peak})}$; small blocks are device-limited, while larger blocks become channel-limited, yielding a strong, though not perfectly proportional, increase of IOPS as l_{blk} decreases. pSLC falls between SLC and TLC across all sizes. Storage-Next SSDs are designed to exploit this regime: they provide scalable small-block IOPS, especially with SLC or pSLC, whereas conventional SSDs remain nearly flat at $\leq$ 4KB due to 4KB-oriented ECC/controller architecture.

Figure 3: Storage-Next SSD peak IOPS under different configurations and workload read-to-write ratio of 90:10.
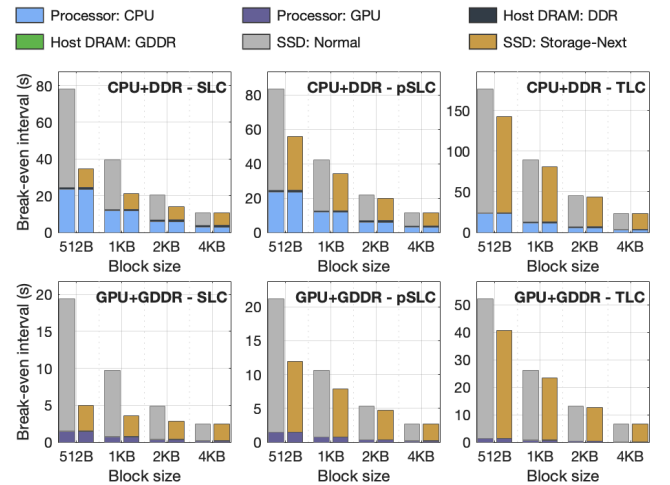
Since the break-even formula Eq. 4 has the costs on both the numerator and the denominator, we can apply normalized cost to facilitate comparison across configurations. In this work, we normalize all component costs to the NAND-die cost. Let α_{CTRL}, $\alpha_{\text{S\_DRAM}}$, $\alpha_{\text{H\_DRAM}}$, α_{CORE} denote the normalized costs of the SSD controller, SSD-internal DRAM, host DRAM, and host cores. Table 2 summarizes the normalized cost and performance assumptions for two representative platforms: CPU+DDR and GPU+GDDR. All normalized costs are estimated from a manufacturing perspective, driven by die size and process technology rather than market price, to avoid buyer-dependent pricing bias (e.g., hyperscalers vs. small enterprises). Because DDR and NAND have comparable die areas, we set DDR's normalized die cost to 1; GDDR is set to 2 due to higher pin counts, larger die/package, and tighter power/thermal limits. Based on internal chip-design experience, we set α_{CTRL} as 15, consistent with its area/complexity on mature nodes (e.g., 12–7nm). For CPUs/GPUs, costs reflect advanced-node wafer pricing and representative die sizes. We set the normalized cost of a server-class CPU core to 4, with 1 M IOPS/core; for GPUs, we set the normalized cost per SM to 3, with 4 M IOPS/SM, based on experiments with

NVIDIA's SCADA (SCaled Accelerated Data Access) platform [36] on the NVIDIA Hopper generation of GPUs. Although actual costs vary with product design, foundry pricing, and packaging, our methodology offers a more transparent and repeatable alternative to market-price snapshots.

Table 2: Normalized cost and performance parameters under different compute+memory configurations.

Platform	$\alpha_{\text{H\_DRAM}}$	$B_{\text{H\_DRAM}}$	$C_{\text{H\_DRAM}}$	α_{CORE}	$IOPS_{\text{CORE}}$	α_{CTRL}	$\alpha_{\text{S\_DRAM}}$
CPU+DDR	1	3 GB/s	3 GB	4	1M	15	1
GPU+GDDR	2	80 GB/s	2 GB	3	4M	15	1

To complete the study, we compute the break-even interval in Fig. 4. For each block size, the left bar is the *Normal-SSD* baseline (flat IOPS for $\leq$4 KB) and the right bar is the *Storage-Next SSD* (IOPS rises as block size shrinks). Following Jim Gray's formulation, we assume the system fully utilizes peak SSD IOPS, i.e., set $IOPS_{\text{SSD}} = IOPS_{\text{SSD}}^{(\text{peak})}$ when evaluating Eq. 4. Each stacked bar decomposes the interval into processor, host-DRAM, and SSD components, making clear how architectural and device parameters jointly shape placement decisions. As NAND sensing latency increases from $5\mu s$ (SLC) to $40\mu s$ (TLC), SSD IOPS/$ falls and hence SSD share of the stack grows. The break-even interval shortens with larger block sizes due to higher host-DRAM "rent". For example, under SLC on CPU+DDR it drops from ~34s at 512B to ~10s at 4KB. GPU-based systems show substantially shorter intervals than CPU-based systems; under SLC at 512B the interval falls from ~34s (CPU+DDR) to ~5s (GPU+GDDR), roughly a 7× reduction. Finally, within each block-size group, the Storage-Next bar is consistently lower than the Normal-SSD bar for sub-4KB requests, with the largest gaps in the SLC where small-block IOPS scaling dominates.

Figure 4: Break-even interval across configurations. Each stack shows contributions from host processor (CPU/GPU), DRAM, and SSD.

These results underscore the value of a first-principle framework that ties cost to architectural and physical behavior rather

than datasheet peaks. Properly engineered Storage-Next SSDs with scalable IOPS can push the break-even interval from minutes to a few seconds, and to low single-digit seconds on GPU platforms, challenging the historic memory–storage divide and positioning NAND as a viable tier in the active memory hierarchy.

4 Constraint-aware Break-even (RQ2)

The calibrated economic model above retains a key assumption from the original five-minute rule and subsequent revisits: the system can always fully utilize the SSD's peak IOPS. In this section, we drop this assumption and make the break-even interval estimation feasibility-aware by introducing two constraints that bound usable SSD throughput: (i) application-level read latency constraint, and (ii) the platform's total host IOPS capability.

To incorporate the latency constraint, we model each NAND flash channel as an M/D/1 queue [20, 27], where read requests arrive according to a Poisson process, service time is deterministic, and a single channel serves one request at a time. Given the peak SSD IOPS $IOPS_{\mathrm{SSD}}^{(\mathrm{peak})}$ (see Eq. 5 in Section 3 for its calculation) and total N_{CH} channels inside SSD, we can express the per-channel deterministic service time as $N_{\mathrm{CH}}/IOPS_{\mathrm{SSD}}^{(\mathrm{peak})}$. Meanwhile, we need to incorporate the additional sensing latency within NAND flash memory chips. Therefore, let $0 \le \rho \le 1$ denote the channel bandwidth utilization, the mean read request service latency can be calculated as

$$\tau_{\mathrm{mean}}(\rho) = \frac{N_{\mathrm{CH}}}{IOPS_{\mathrm{SSD}}^{(\mathrm{peak})}} \cdot \left(1 + \frac{\rho}{2(1-\rho)}\right) + \tau_{\mathrm{sense}} . \quad (11)$$

Moreover, following Kingman's heavy-traffic limit [20, 25], the waiting time is well-approximated by an exponential distribution, hence we can approximate the p-th percentile tail-latency as

$$\tau_{\mathrm{p}}(\rho) = \frac{N_{\mathrm{CH}}}{IOPS_{\mathrm{SSD}}^{(\mathrm{peak})}} \cdot \left(1 + \frac{\rho}{2(1-\rho)} \cdot \ln\left(\frac{1}{1-p}\right)\right) + \tau_{\mathrm{sense}} . \quad (12)$$

Let $\{\hat{\tau}_{\mathrm{mean}}, \hat{\tau}_{\mathrm{p}}\}$ denote the application-level constraints on mean read latency and p-th percentile tail read latency. Given $\{\hat{\tau}_{\mathrm{mean}}, \hat{\tau}_{\mathrm{p}}\}$, we solve for the largest $\rho \in (0, 1)$ (denoted as ρ_{max}) that satisfies $\tau_{\mathrm{mean}}(\rho_{\mathrm{max}}) \le \hat{\tau}_{\mathrm{mean}}$ and $\tau_{\mathrm{p}}(\rho_{\mathrm{max}}) \le \hat{\tau}_{\mathrm{p}}$. Accordingly, we have that the usable SSD IOPS is $IOPS_{\mathrm{SSD}} = \rho_{\mathrm{max}} \cdot IOPS_{\mathrm{SSD}}^{(\mathrm{peak})}$. In essence, the scaling factor ρ_{max} reflects the impact of application-level read latency constraints on the usable SSD IOPS. Moreover, let $IOPS_{\mathrm{proc}}^{(\mathrm{peak})}$ denote the maximum total IOPS that the host processor can practically sustain, we can further calibrate the usable SSD IOPS as

$$IOPS_{\mathrm{SSD}} = \min\left(\rho_{\mathrm{max}} \cdot IOPS_{\mathrm{SSD}}^{(\mathrm{peak})}, IOPS_{\mathrm{proc}}^{(\mathrm{peak})}/N_{\mathrm{SSD}}\right), \quad (13)$$

where N_{SSD} is the number of SSDs controlled by host processor.

Fig. 5 extends the quantitative study in Section 3.3 under the feasibility constraints discussed above in this section. We focus on SLC NAND and Storage-Next SSDs (scalable small-block IOPS). Because device service time depends on block size, we specify a separate 99th-percentile read-latency target for each block size, denoted $\tau_{\mathrm{tail\_512B}}, \tau_{\mathrm{tail\_1KB}}, \tau_{\mathrm{tail\_2KB}}, \tau_{\mathrm{tail\_4KB}}$. For simplicity, we do not set any constraint on mean read latency. Table 3 gives four tail-latency tiers chosen so that 512B, 1KB, 2KB, and 4KB all admit the same $\rho_{\mathrm{max}} \in \{0.70, 0.80, 0.90, 0.99\}$. We assume the host drives four SSDs and sweep CPU capacities $IOPS_{\mathrm{proc}}^{(\mathrm{peak})} \in \{40M, 60M, 80M, 100M\}$

(guided by $\sim$ 1M IOPS/core) and GPU capacities $IOPS_{\mathrm{proc}}^{(\mathrm{peak})} \in \{160M, 240M, 320M, 400M\}$ (guided by $\sim$ 4M IOPS/SM).

Table 3: 99th-percentile tail latency tiers per block size (Storage-Next SSD with SLC NAND), chosen to equalize the admissible utilization ρ_{max} across block sizes.

$\tau_{\mathrm{tail\_512B}}$	$\tau_{\mathrm{tail\_1KB}}$	$\tau_{\mathrm{tail\_2KB}}$	$\tau_{\mathrm{tail\_4KB}}$	ρ_{max}
$7\mu s$	$9\mu s$	$11\mu s$	$16\mu s$	70%
$9\mu s$	$11\mu s$	$15\mu s$	$23\mu s$	80%
$13\mu s$	$17\mu s$	$26\mu s$	$44\mu s$	90%
$85\mu s$	$135\mu s$	$230\mu s$	$418\mu s$	99%

Impact of host IOPS capacity. Fig. 5(a)-(b) examine sensitivity to the host-side IOPS ceiling $IOPS_{\mathrm{proc}}^{(\mathrm{peak})}$ with no latency cap ($\rho_{\mathrm{max}} = 1$). While the system is *host-limited*, increasing $IOPS_{\mathrm{proc}}^{(\mathrm{peak})}$ allows a larger share of requests to be handled within the host's processing budget, thereby shortening the break-even interval. Once the device peak $IOPS_{\mathrm{SSD}}^{(\mathrm{peak})}$ becomes the bottleneck, further increases in $IOPS_{\mathrm{proc}}^{(\mathrm{peak})}$ no longer reduce the interval. The transition from host-limited to device-limited depends on both the host IOPS budget and the block size, since Storage-Next SSD peak IOPS decline as block size grows. For example, at 512B on CPU+DDR, raising the CPU budget from 40M to 100M IOPS reduces the break-even interval from 83s to 47s; by contrast, at 4KB on CPU+DDR the interval stays near 10s regardless of CPU budget, indicating device limitation. Given GPUs' much higher $IOPS_{\mathrm{proc}}^{(\mathrm{peak})}$, the GPU-based platform operates almost entirely in the device-limited regime; moreover, due to GPU's higher IOPS/\$, GPU+GDDR maintains substantially shorter break-even intervals (well below 7s) across all block sizes.
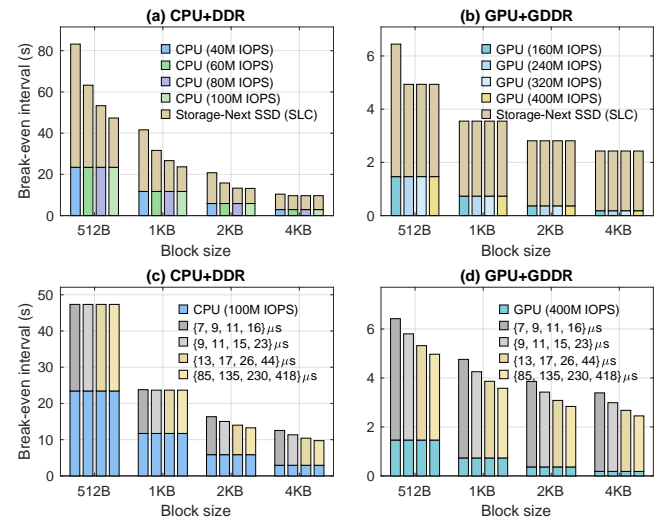

Figure 5: (a) and (b): break-even interval under different host processor IOPS capacity without latency constraint; (c) and (d) break-even interval under different tail latency constraints with fixed processor IOPS capacity.

Impact of latency constraint. Fig. 5(c)-(d) hold the host budgets fixed (CPU: 100M IOPS; GPU: 400M IOPS) and vary only the 99th-percentile tail-latency tier from Table 3. Tightening the tier (moving from the 99% row toward 90–70%) lowers the admissible SSD IOPS utilization $\rho_{\max}$ and hence usable SSD IOPS, leading to longer break-even interval. Conversely, when the fixed host budget is already the limiter for a given block size, adjusting the tail tier has little or no effect (e.g., 512B and 1KB on CPU+DDR platform). Quantitatively, the sensitivity to tail latency is modest: for 512B on GPU+GDDR, relaxing the 99th-percentile from $7\mu s$ to $85\mu s$ reduces the break-even interval by only about 1.5s.

In summary, the above results show that host processor IOPS capacity is the primary lever for shrinking the break-even interval, while latency constraints have a comparatively modest effect. Raising the host budget moves the system out of the host-limited regime, immediately lowering the SSD term and driving large, monotonic reductions in the break-even intervals, especially at small blocks where the device can sustain high IOPS. By contrast, tightening or relaxing the tail target changes the admissible utilization but typically shifts the bars far less. This asymmetry reinforces the case for using the GPU as the I/O engine: its higher IOPS capacity keeps operation away from the host ceiling and, paired with Storage-Next SSD's IOPS scalability, consistently pulls the break-even interval into the few-seconds regime.

5 Workload-Aware Platform Analysis (RQ3)

Building upon Sections 3–4, this section develops a workload-aware framework for systematic, quantitative assessment of a candidate hardware platform's *viability* and *economic optimality* for a given workload. Given the workload's access-interval profile and a fixed platform, we seek to answer the following questions:

(1) **Viability and optimality:** Does the platform, as configured, satisfy the workload's throughput and latency targets? If so, can it also operate at the economics-optimal (break-even) point?

(2) **Binding bottleneck and minimal upgrade:** If not, which hardware resource (DRAM bandwidth/capacity, SSD usable IOPS/bandwidth, or host IOPS) is the binding bottleneck, and what minimal upgrade attains viability and/or the break-even point?

5.1 Analysis Framework Development

For simplicity, we assume a single data access granularity l_{blk}. Let N_{blk} be the number of size-l_{blk} blocks in the working set (hence total size $N_{\text{blk}} \cdot l_{\text{blk}}$). To capture the workload data access characteristics, let τ_i denote the average access interval of block i, and define $\mathcal{S}(T) = \{ i : \tau_i \leq T \}$, the set of blocks whose access intervals do not exceed T. The workload also provides mean/tail latency targets and a read:write ratio. A given hardware platform has fixed host-processor IOPS budget, per-SSD peak IOPS, number of SSDs, host-DRAM bandwidth and capacity, and component-cost structure. This work solely focuses on regimes where the working set is much larger than the host-DRAM capacity.

We assume optimal DRAM caching: every block cached in DRAM has a shorter access interval than any uncached block; equivalently, there exists a threshold T with cached set $\mathcal{S}(T)$. For any T, the aggregate cached and uncached access throughputs (bytes/s) are

$$\Psi_c(T) = l_{\text{blk}} \sum_{i \in \mathcal{S}(T)} \frac{1}{\tau_i}, \qquad \Psi_d(T) = l_{\text{blk}} \sum_{i \notin \mathcal{S}(T)} \frac{1}{\tau_i}. \qquad (14)$$

We assume an efficient zero-copy I/O stack to minimize I/O-induced host-DRAM traffic. Under this model, a DRAM cache miss incurs one SSD→DRAM DMA plus one DRAM read by the processor. The resulting host-DRAM bandwidth demand is

$$B_{\text{DRAM}}^{\text{use}}(T) = \Psi_c(T) + 2\,\Psi_d(T). \qquad (15)$$

As T increases, $\mathcal{S}(T)$ expands, $\Psi_c(T)$ increases, and $\Psi_d(T)$ decreases with $\Psi_c(T) + \Psi_d(T) = l_{\text{blk}} \sum_i 1/\tau_i$ fixed; therefore $B_{\text{DRAM}}^{\text{use}}(T)$ decreases strictly with T.

We now define three thresholds, T_B, T_S, and T_C, to isolate the impacts of DRAM bandwidth, SSD bandwidth, and DRAM capacity. Because $\mathcal{S}(T)$ expands monotonically with T, the bandwidth thresholds below are thus well defined and unique whenever a solution exists.

• *DRAM bandwidth.* The DRAM-bandwidth threshold T_B is the *smallest* access-interval threshold at which the required host-DRAM traffic does not exceed the available DRAM bandwidth:

$$T_B \triangleq \min\{ T > 0 : B_{\text{DRAM}}^{\text{use}}(T) \leq B_{\text{DRAM}} \}, \qquad (16)$$

where B_{DRAM} denotes the host-DRAM bandwidth. Because $B_{\text{DRAM}}^{\text{use}}(T)$ decreases strictly with T, the solution, when it exists, is unique, and a simple existence check is $B_{\text{DRAM}} \geq l_{\text{blk}} \sum_i 1/\tau_i$. Moreover, increasing B_{DRAM} lowers T_B.

• *SSD bandwidth.* The SSD-bandwidth threshold T_S is the *smallest* threshold that confines the uncached throughput to the aggregate usable SSD bandwidth. Given the latency targets and the host-processor IOPS budget, the usable per-SSD IOPS_{SSD} is obtained as in Section 4. We define

$$T_S \triangleq \min\{ T > 0 : \Psi_d(T) \leq B_{\text{SSD}} \}, \qquad (17)$$

where $B_{\text{SSD}} = l_{\text{blk}} \cdot N_{\text{SSD}} \cdot \text{IOPS}_{\text{SSD}}$ and N_{SSD} is the number of SSDs. Since $\Psi_d(T)$ decreases with T, the solution is unique. Scaling N_{SSD}, selecting higher-IOPS devices, or increasing the host-IOPS budget raises B_{SSD} and therefore reduces T_S.

• *DRAM capacity.* The capacity threshold T_C is the *largest* threshold whose cached set fits within available DRAM:

$$T_C \triangleq \max\{ T > 0 : |\mathcal{S}(T)|\, l_{\text{blk}} \leq C_{\text{DRAM}} \}, \qquad (18)$$

where C_{DRAM} denotes host-DRAM capacity. Ordering $\{\tau_i\}$ increasingly and letting $K = \lfloor C_{\text{DRAM}}/l_{\text{blk}} \rfloor$, T_C equals the K-th smallest τ_i; operationally, at most the K most frequently accessed blocks can be cached in DRAM.

If $\max(T_B, T_S) \leq T_C$, the platform is viable for the workload. When $T_B = T_S$, DRAM and SSD bandwidths are perfectly balanced. To minimize DRAM cost while maintaining viability, the designer should select C_{DRAM} so that $T_C = \max(T_B, T_S)$. The platform operates at the economics-optimal point if $\tau_{\text{break-even}} \in [\max(T_B, T_S), T_C]$. If this condition is not met, we should diagnose the limiting path and upgrade accordingly: when $T_B > T_C \geq T_S$, the system is DRAM-bandwidth limited and the designer should increase B_{DRAM} (for example, by adding DDR/GDDR channels, raising data rates, or adopting HBM); when $T_S > T_C \geq T_B$, the storage I/O path is limiting and the designer should raise the aggregate SSD throughput B_{SSD} by scaling N_{SSD}, selecting higher-$\text{IOPS}^{(\text{peak})}$ devices, and/or increasing

the host IOPS budget if $IOPS_{proc}^{(peak)}$ is the sub-limiter; and when both T_B and T_S exceed T_C, bandwidth and capacity are jointly insufficient, so the designer should either increase C_{DRAM} until $T_C \geq \max(T_B, T_S)$ or reduce $\max(T_B, T_S)$ through bandwidth upgrades, with the choice guided by a price model or stated priority. After any upgrade, we should recompute $\tau_{break\text{-}even}$; if $\tau_{break\text{-}even} \notin [\max(T_B, T_S), T_C]$, the configuration remains viable but off-optimum, and we should further adjust the limiting resources to bring the break-even placement within reach or closer.

5.2 Quantitative Study

Extending the study in Sections 3–4, we demonstrate the workload-aware platform analysis framework on CPU+DDR and GPU+GDDR platforms. For CPU+DDR platform, we set 12 channels of DDR5-5600 (hence total 540GB/s DRAM bandwidth); for GPU+GDDR platform, we set 8 channels of GDDR6-20 (hence total 640GB/s DRAM bandwidth). We set the peak CPU IOPS capacity to 100M and peak GPU IOPS capacity to 400M. Each platform deploys four SSDs, and we consider both normal SLC SSD and Storage-Next SLC SSD. We adopt 99th-percentile read-latency constraint of $13\mu s$ (512B), $17\mu s$ (1KB), $26\mu s$ (2KB), and $44\mu s$ (4KB), corresponding to SSD IOPS utilization ρ_{max} of 90% as shown above in Section 4.

Given the target workload, we aim to make each hardware platform viable (and economics-optimal if practically possible) by provisioning the DRAM capacity C_{DRAM}. Since DRAM capacity now is a variable, we only need to calculate the metrics T_B and T_S. Accordingly, we obtain $T_v = \max(T_B, T_S)$ as the viability threshold, and $C_{DRAM}^{(V)} = |\mathcal{S}(T_v)| \cdot l_{blk}$ represents the minimum DRAM capacity for making the hardware platform viable. Given the break-even interval $\tau_{break\text{-}even}$, we obtain $T_o = \max(\tau_{break\text{-}even}, T_v)$ as the economics-optimal threshold, and $C_{DRAM}^{(O)} = |\mathcal{S}(T_o)| \cdot l_{blk}$ represents the minimum DRAM capacity for making the hardware platform economics-optimum.

We assume the workload data access interval follows a lognormal distribution with total throughput $l_{blk} \sum_i 1/\tau_i$ of 200GB/s that is smaller than but reasonably comparable with the host DRAM bandwidth. Workload data set contains 1 billion data blocks. Hence the data set size is 512GB, 1TB, 2TB, and 4TB when block size is 512B, 1KB, 2KB, and 4KB, respectively. Fig. 6 shows the simulation results of the minimum DRAM capacity $C_{DRAM}^{(V)}$ and $C_{DRAM}^{(O)}$, and the corresponding total DRAM bandwidth usage. In Fig. 6(b) and (d), each DRAM bandwidth usage bar contains two components: the top represents the I/O-induced DRAM bandwidth usage for uncached data and the bottom represents the DRAM bandwidth usage for cached data (i.e., (i.e., $2\Psi_d(T)$) and $2\Psi_c(T)$ in Eq. 15). Several bars (e.g., economics-optimal under normal SSD at 512B on CPU+DDR platform) contain only one component since the corresponding minimum DRAM capacity can already hold the entire data set and hence there are no I/O operations. Fig. 6 shows a clear inverse relation between the minimum DRAM capacity and the I/O-induced DRAM bandwidth usage: as capacity increases, more blocks remain DRAM-resident, misses fall, and the miss-path traffic term $2\Psi_d(T)$ shrinks. With less DRAM, a larger uncached set drives more SSD accesses and higher I/O-induced DRAM bandwidth.

On the CPU+DDR platform, whether paired with a normal SSD or a Storage-Next SSD, the break-even interval τ_{be} is consistently
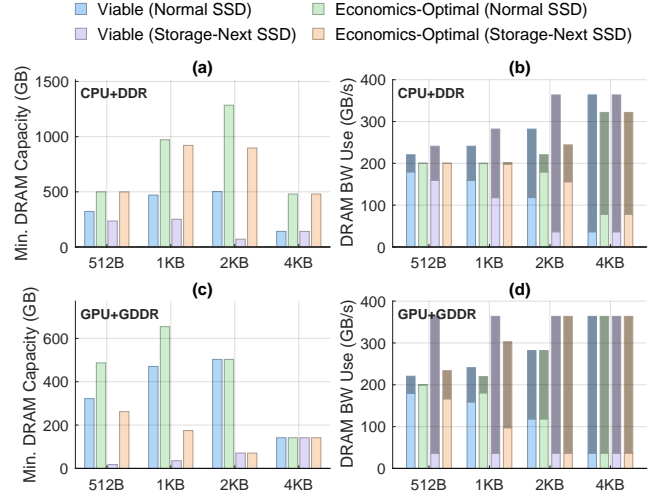
Figure 6: Minimum DRAM capacity required for the CPU+DDR or GPU+GDDR hardware platform to be viable or economics-optimal, and the corresponding DRAM bandwidth usage.

longer than $T_v = \max(T_B, T_S)$. Consequently, the economics-optimal DRAM capacity is set by τ_{be}, not by viability. At 512B and 1KB block sizes, τ_{be} is so large that achieving the economics optimum requires caching essentially the entire dataset (about 512GB and 1TB, respectively) in DRAM. As block size increases, τ_{be} decreases, so the economics-optimal cache constitutes a smaller fraction of the dataset. Because DRAM bandwidth comfortably exceeds the workload bandwidth, we have $T_v = T_S$; i.e., SSD IOPS, not DRAM bandwidth, determines the minimum DRAM capacity needed for viability. This explains why the viable DRAM capacity is lower with Storage-Next SSDs: their higher IOPS reduce T_S and therefore the required cache for viability.

On the GPU+GDDR platform with Storage-Next SSDs, both T_B and T_S are small (<5s) thanks to high GDDR bandwidth, large GPU IOPS capacity, and high usable SSD IOPS. Consequently, the viable DRAM requirement is low—especially at small block sizes, so the workload can remain viable while a larger share of traffic is serviced as I/O through GDDR. By contrast, the economics-optimal DRAM at 512B and 1KB can be much larger because the break-even interval dominates; the cost-optimal point therefore caches a substantial portion of the working set (e.g., 260GB on GPU+GDDR). At 2KB and 4KB, τ_{be} shortens and T_S becomes the governing term, so the viable and economics-optimal DRAM capacities coincide. In short, Fig. 6 highlights the fundamental advantage of combining GPUs with Storage-Next SSDs: both viability and (often) economics-optimal operation are achievable with far less DRAM than CPU+DDR.

6 MQSim-Next: Storage-Next SSD Simulator

To study the Storage-Next regime with realistic fidelity, we extend MQSim [9, 46] into MQSim-Next, preserving its validated foundations (e.g., PCIe/TLP and FTL/cache timing, request-fetch control, steady-state preconditioning, and prior calibration against
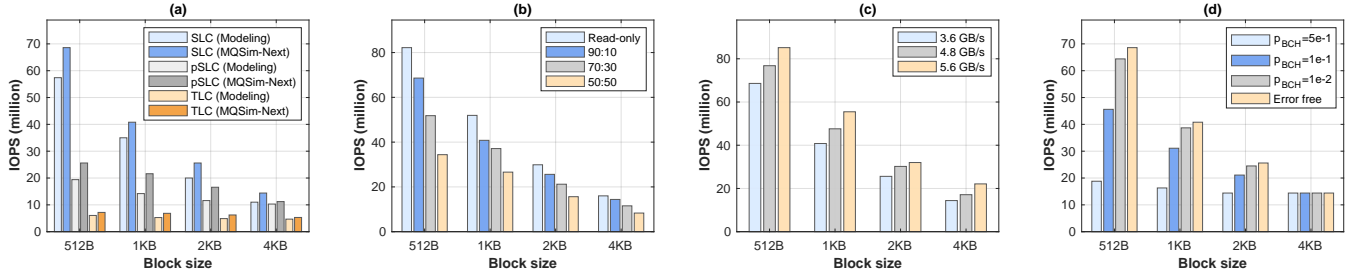
Figure 7: (a) Comparison of modeled and simulated IOPS under 90:10 read-to-write ratio, (b) simulated SLC SSD IOPS under different read-to-write ratio, (c) simulated SLC SSD IOPS under different NAND channel bandwidth, and (d) simulated SLC SSD IOPS under different BCH decoding failure rate p_{BCH}.

real multi-queue SSDs). Below, we summarize the major upgrades and enhancements made in MQSim-Next.

To better reflect modern NAND flash behavior, we introduce three back-end enhancements, together constituting a necessary extension of MQSim's timing model for contemporary parts. First, we add SCA [32] on the NAND channel, so command/address movement incurs a much shorter per-command cost; this sustains effective channel bandwidth as request sizes shrink and mirrors current device practice. Second, we add the support of *independent multi-plane reads* [23, 42], which can much better exploit intra-die parallelism in modern NAND flash. Third, we add the support of *explicit transfer-sense overlap*, allowing array sensing/programming for one request to proceed concurrently with command/address or data movement for another. Collectively, these enhancements are essential to aligning MQSim-Next with contemporary NAND timing and concurrency in the Storage-Next regime. Accordingly, we enhance the scheduler to better exploit back-end parallelism via read-prioritized, plane-aware scheduling: short reads continue to advance alongside long programs, while the channel arbiter can interleave SCA command/address bursts before data transfer to improve small-I/O utilization.

Another important upgrade is an explicit ECC model. SSDs usually protect data at 4KB codeword granularity, flattening random-IOPS for ≤4KB requests because each small read pays a full 4KB decode/transfer. To hit Storage-Next small-block IOPS, shorter protection (e.g., 512B) helps but increases redundancy and requires more parallel decoders, raising bit cost and controller complexity. Following concatenated coding [31, 33], MQSim-Next uses a two-layer scheme: a BCH inner code on each 512B sector and an LDPC outer code spanning eight such sectors (4KB). On reads of one or more contiguous 512 B blocks, the controller decodes only the needed BCH codeword(s); if all succeed, it avoids touching the 4KB LDPC codeword and thus prevents intra-SSD read amplification. On any BCH failure, the simulator escalates by fetching the full 4KB LDPC codeword, accounting for extra transfer and a base-plus-iterative LDPC latency. The simulator supports configurable BCH decoding failure probability, enabling users probe small-read tail behavior and ECC-induced read amplification.

We also extend MQSim-Next to support a much larger number of I/O request queues, which is crucial for extracting the full random-IOPS potential of Storage-Next SSDs under deep host parallelism. We configure the simulator using the parameters in Table 1

and set the PCIe interface to Gen7 ×8[3]. Fig. 7 validates the model against MQSim-Next and probes key sensitivities. In Fig. 7(a) the two align closely. MQSim-Next reports slightly higher IOPS because our model adopts a conservative write-amplification factor (Φ_{AW}=3), which raises the modeled write share and lowers usable IOPS. Both the model and simulator assume a controller-nonlimiting setup, so the observed limits reflect NAND and channel behavior rather than controller bottlenecks. In Fig. 7(b) IOPS declines as the workload becomes more write heavy due to GC traffic that competes with host I/O. At 512B (SLC), IOPS drops from about 82M (read only) to 68M (90:10), 52M (70:30), and 34M (50:50). Fig. 7(c) shows a monotonic IOPS increase with NAND channel bandwidth, revealing a channel-side ceiling even with short SLC latencies: at 512B (SLC), IOPS rises from 68M at 3.6GB/s to 77M at 4.8GB/s and 85M at 5.6GB/s. This motivates wider channels and lower per-command overheads; die-stacked I/O such as Xtacking [21] can further raise effective per-channel bandwidth. Fig. 7(d) quantifies how 512B BCH decoding failures reduce IOPS under the concatenated BCH/LDPC scheme: each failure escalates to a 4KB LDPC decoding, adding transfer and latency. Well-designed 512B BCH can keep failure rates below 1%, and performance is already near the error-free plateau at a 1% rate. In summary, MQSim-Next reproduces the first-principles trends and the key sensitivities shown in Fig. 7 (workload mix, channel bandwidth, and ECC). This alignment supports the calibrated model and feasibility framework for provisioning and design guidance, and it provides a solid foundation for the next Section on exploring the design space opened by seconds-scale DRAM–flash caching.

7 Re-think Data-intensive Software (RQ4)

By collapsing the DRAM-flash caching threshold to seconds, Storage-Next SSDs open the door to fundamentally rethinking algorithm and data structure design. We advance two complementary principles: (1) proactively re-architect algorithms and data structures to exploit ultra-high random IOPS at small block sizes (e.g., 512B), favoring fine-grained access and wide concurrency; and (2) leverage flash's far lower \$/GB than DRAM to purposely employ sparse (over-provisioned) data structures when they yield higher speed and lower complexity, even if they consume more bytes on flash. Following these principles, this section presents two case studies,

---

[3]We use Gen7 ×8 so that PCIe bandwidth does not bottleneck 4KB IOPS as the NAND channel bandwidth scales from 4.8GB/s to 5.6GB/s.
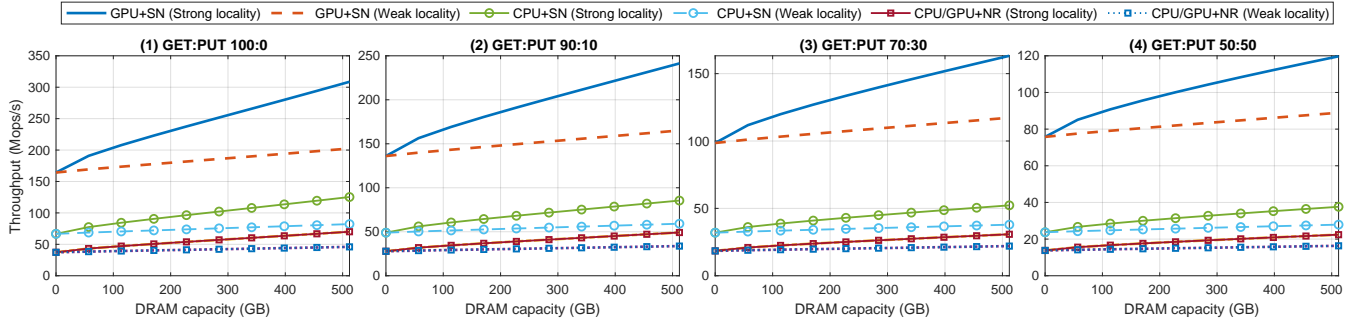
Figure 8: Achievable operational throughput of SSD-resident blocked-Cuckoo KV store under different GET:PUT ratio and DRAM capacity. Storage-Next SSDs and normal SSDs are denoted as SN and NR, respectively.

blocked-Cuckoo SSD-resident KV store and progressive SSD-centric ANN search, to demonstrate a practical path to high throughput and simpler implementation of data-intensive software.

7.1 Case Study 1: SSD-Resident KV Store

KV stores anchor modern AI stacks, powering feature lookups in recommenders, embedding caches, LLM memory layers, and session-state in serving pipelines. These workloads often involve billions of unique keys with sparse, unpredictable access patterns. To meet throughput, many systems use in-memory KV stores (e.g., Redis [2], FASTER [7], MICA [30]) with hash indexing for low latency and simplicity, but the DRAM footprint becomes economically untenable at scale. This has driven interest in hybrid DRAM/SSD engines (e.g., RocksDB [3, 13], WiredTiger [4], Bw-tree [29]) that embrace block I/O and tree-like indexing. However, even these hybrid designs often retain substantial DRAM-resident indexing and metadata (e.g., hash directories, filters, block catalogs) to keep lookup latency acceptable, which grows with key cardinality and still limits capacity per dollar.

Building on the Storage-Next context, we propose an SSD-native KV store that instantiates a blocked Cuckoo hash [26, 40] directly on the SSD, eliminating any DRAM-resident index or metadata. Meta's CacheLib [1, 6] also uses a hash index for SSD-resident KV pairs; because it serves as a cache, it can discard entries when a bucket is full. We target a persistent KV store that must not drop items, so we adopt Cuckoo hashing, using relocations rather than discards to handle bucket overflows. Each key maps to two candidate SSD-resident buckets, and each bucket matches to one SSD block. Each lookup therefore requires one or two SSD block reads (on average 1.5). To avoid insertion failure, the runtime load factor α must remain below the critical threshold α_{critical} determined by the bucket size $B = \lfloor l_{\text{blk}}/l_{\text{KV}} \rfloor$, where l_{blk} is the SSD block size and l_{KV} is the average KV-pair size (e.g., 64B). Prior work [26, 40] shows that even for modest B (≥ 4), α_{critical} typically exceeds 0.95. Insertions may trigger short displacement chains whose expected length can be estimated by $\mathbb{E}[L] \approx \frac{\alpha^{2B}}{1-\alpha^B}$, so operating well below α_{critical} keeps $\mathbb{E}[L] \ll 1$, yielding nearly constant insertion latency. We dedicate all available DRAM to caching individual hot KV pairs. We use an SSD-resident write-ahead log (WAL) for persistence and to amortize write cost by consolidating updates that target the same hash bucket. When the WAL exceeds a size threshold, the system

commits the consolidated updates into blocked-Cuckoo hash blocks and then recycles the freed log space.

For demonstration, we evaluate throughput in a realistic large-scale setup: a 5TB KV store with 80 billion 64B items, load factor 0.7, and bucket sizes matched to device class (512B on Storage-Next SSDs, 4KB on normal SSDs). All DRAM is devoted to caching hot KV pairs. We considered four different GET:PUT ratios (100:0, 90:10, 70:30, and 50:50), with 20% of PUTs as inserts and the rest updates. Access intervals follow a lognormal distribution under two locality regimes: strong ($\sigma = 1.2$) and weak ($\sigma = 0.4$). Hardware matches prior sections: CPU+DDR or GPU+GDDR, where CPU and GPU IOPS capacities are 100M and 400M, and DDR and GDDR bandwidths are 540GB/s and 640GB/s, respectively. Each platform uses four SSDs (either Storage-Next or normal), with SSD bandwidth utilization capped at 70% to reduce tail latency. Fig. 8 reports simulated achievable throughput under both device/host-IOPS and DRAM-bandwidth bounds. With normal SSDs the system is device-limited, so CPU and GPU collapse into a single curve. The results show clear dependence on data access locality and GET:PUT ratio: strong locality extracts more value from added DRAM capacity because the cache captures a larger hot set, converts more data accesses into cache hits, and collapses distinct KV pair updates and hence SSD read-modify-write operations per WAL flush. In contrast, as the write share grows, the system issues more read-modify-write operations to SSD, increasing I/O traffic and reducing the operational throughput.

Pairing GPUs with Storage-Next SSDs (GPU+SN) is especially advantageous. On read-heavy mixes, GPU+SN sustains 100+ Mops/s, comparable to in-memory KV stores such as FASTER [7]. Switching to a CPU with the same Storage-Next SSDs shifts the bottleneck to host IOPS capacity, so throughput falls even though the Storage-Next SSD can deliver more. Across the GET:PUT mixes, DRAM bandwidth becomes the limiting factor only when cache-hit rates are very high; otherwise host IOPS capacity and SSD throughput dominate. In summary, the GPU & Storage-Next combination proves essential for realizing the vision of a fully SSD-resident KV store built on blocked-Cuckoo hashing. By exploiting both the parallelism of GPUs and the IOPS scalability of next-generation SSDs, it transforms NAND flash memory from a passive storage tier into an active, memory-like substrate capable of sustaining in-memory-class KV store throughput.
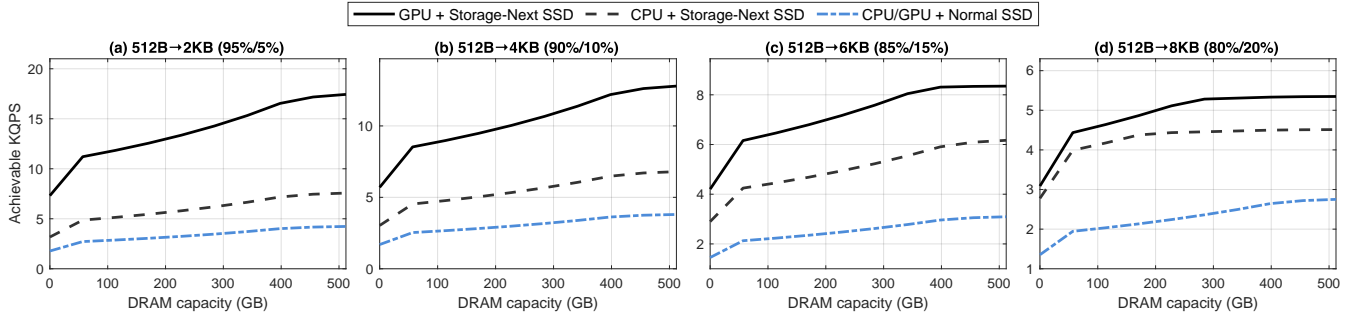
Figure 10: ANN search throughput under different full-vector length with reduced-vector length fixed as 512B.

7.2 Case Study 2: SSD-Resident ANN Search

ANN is a cornerstone of modern AI services, including recommendation, retrieval-augmented generation (RAG), and anomaly detection, yet modern workloads often involve TB/PB-scale embedding corpora, well beyond feasible DRAM capacity. Prior SSD-resident systems [16, 22] trade search quality to accommodate a minute-scale DRAM-SSD caching threshold in conventional platforms. By collapsing this threshold to a few seconds, GPUs paired with Storage-Next SSDs enable a rethink of SSD-resident ANN. As a first step, motivated by widespread use of *dimensionality reduction* in DRAM-resident ANN [11, 14, 15, 28], we propose a *two-stage progressive* SSD-resident design (Fig. 9). Each embedding is stored on SSD in both a compact, reduced-dimension form (e.g., 512B) and a full-dimension form (e.g., 4KB). At query time, reduced vectors are fetched first to prune unlikely candidates; only a small filtered set is then re-ranked using full vectors. This is effective because most distance computations simply confirm rejection: Gao et al. [15] report that over 90% of comparisons eliminate candidates, so full-dimension evaluation is often unnecessary. Reduced vectors can come from (1) linear transforms such as PCA or random projection [11, 14, 47], (2) a dual-model embedding pipeline (full and reduced), or (3) Matryoshka Representation Learning (MRL) [28], which natively supports multi-resolution vectors. On three MRL-generated corpora (MS MARCO, 20 Newsgroups, DBpedia), our experiments show the progressive scheme sustains recall above 98%.
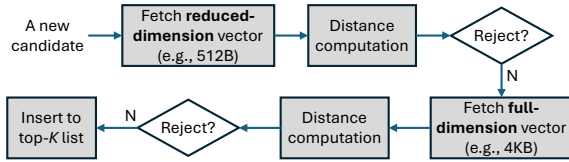

Figure 9: Illustration of two-stage progressive ANN search.

The two-stage scheme benefits directly from Storage-Next SSDs: because most accesses hit reduced-dimension vectors, the workload issues predominantly small-block random reads (e.g., 512B), which unlocks very high IOPS and lifts end-to-end throughput. For demonstration, consider an 8 billion-embedding corpus with full-dimension sizes of 2KB, 4KB, 6KB, and 8KB, respectively, while

fixing the reduced dimension at 512B. We focus on HNSW (Hierarchical Navigable Small World) [34], widely used in state-of-the-art ANN search. To improve scalability, we co-locate graph-link metadata with each node on SSD and devote available DRAM to caching the hotter upper-layer nodes. HNSW concentrates traversal by layer: layer sizes shrink rapidly with height, per-query visits per layer also decline (though more slowly), and thus per-node access intervals shorten at higher layers, making them DRAM-cache friendly. We evaluate using a calibrated, layer-aware synthetic trace that mirrors HNSW's coarse-to-fine pipeline. Under full-dimension sizes of 2KB, 4KB, 6KB, and 8KB, roughly 5%, 10%, 15%, and 20% of candidates are promoted to full-vector re-ranking, respectively. In this regime, reduced-vector fetches are IOPS-bound and benefit most from Storage-Next SSDs, while the small promoted fraction is bandwidth-bound yet amortized by the large rejection rate.

Fig. 10 shows the simulated ANN search throughput (KQPS) versus DRAM capacity under the four reduced→full vector configurations. Again, we use the same GPU+GDDR and CPU+DDR platforms as in previous studies, and each host deploys four SSDs. Under all the scenarios, GPU paired with Storage-Next SSDs delivers the highest KQPS. In the lighter-promotion cases, i.e., (a) 512B→2KB (95%/5%) and (b) 512B→4KB (90%/10%), GPU paired with Storage-Next SSDs remains bounded by SSD IOPS, rising from roughly 7-11 KQPS at small DRAM to about 13-17 KQPS at 512GB DRAM as caching reduces SSD reads per query. CPU paired with Storage-Next SSDs lies below because it is capped by CPU's 100M IOPS budget. The Normal SSD baseline is always bounded by SSD IOPS. As the full-vector promotion rate increases, DRAM traffic per query grows and bandwidth ceilings emerge. In (c) 512B→6KB (85%/15%), GPU paired with Storage-Next SSDs transitions to DRAM-bandwidth-limited beyond about 400GB, plateauing near 8.3 KQPS, while CPU paired with Storage-Next SSDs remains host-IOPS-limited (up to around 6.2 KQPS). In the heaviest mix (d) 512B→8KiB (80%/20%), GPU paired with Storage-Next SSDs becomes DRAM-bandwidth-limited at roughly 300GB, and CPU paired with Storage-Next SSDs moves from a mixed DRAM-bandwidth/host-IOPS bounded regime around 200GB to purely DRAM-bandwidth-limited thereafter. Overall, Storage-Next SSDs sustain a consistent 2-3 throughput advantage over Normal SSDs: high small-block IOPS sets the upper envelope at low caches, GDDR bandwidth determines the high-cache plateau on GPU platforms, and host IOPS capacity dictates how much of the device's potential the host platform can realize.

Overall, these results show that pairing GPU hosts with Storage-Next SSDs makes low-cost fully SSD-resident ANN practical. TB/PB-scale embedding tables can remain on flash while sustaining high recall and high KQPS, avoiding the large DRAM capacity required for in-memory retrieval. For context, DiskANN [22], an SSD-resident system from Microsoft that constructs a pruned on-disk graph and streams neighbor lists, achieves roughly 5 KQPS on billion-scale benchmarks. On our modeled hardware, the GPU+Storage-Next configuration pushes this boundary toward tens of KQPS[4]. This indicates that the memory hierarchy in the Storage-Next era can match or exceed DiskANN-class throughput while retaining HNSW-level search quality.

8 Limitations and Future Work

While this study establishes a first-principles, feasibility-aware framework for the memory-hierarchy paradigm shift enabled by Storage-Next, several simplifying assumptions and open research directions remain.

Device and cost modeling. Our analysis uses normalized cost parameters and NAND timing values representative of mature 2025-era technologies. Although process variations and future scaling nodes may change absolute numbers, the relative trade-offs, particularly the IOPS-driven collapse of the break-even interval, remain robust. Future work could integrate process-scaling models, cost-learning curves, and the implications of die stacking or 3D integration to capture the economic trajectory of next-generation NAND and controllers more faithfully.

Endurance and write economics. We model write amplification in both the analytic and simulation frameworks, but do not yet incorporate device endurance limits (e.g., retention, refresh policies) or lifetime-driven costs. Extending the framework with endurance-aware models, covering lifetime derating, refresh-induced bandwidth taxes, and energy per I/O, would elevate it to a deployment-grade, sustainability-aware provisioning tool.

Workload coverage. Our workloads focus on read-dominant, large-footprint AI and analytics under single-tenant settings. Extending to write-intensive, transactional, or multi-tenant environments will require modeling of time-varying garbage collection, compaction interference, and bursty access patterns that inflate tail latency. Factoring in update locality and write shaping will improve realism and make the viability analysis broadly applicable across storage services and data-center workloads.

System integration and topology. The framework assumes optimized local PCIe/NVMe paths and single-node coherence. Future deployments increasingly rely on multi-socket servers and disaggregated fabrics that introduce additional latency domains and queue hierarchies. Extending the analysis to these distributed or composable-memory environments will clarify how seconds-scale caching interacts with remote access and networked storage layers.

Host-side I/O optimization. Because host IOPS capacity strongly governs the seconds-scale regime, a key direction is to reduce software overheads and co-design host-device interfaces. Promising approaches include: (i) streamlining the I/O stack for reduced submission latency, and (ii) developing lightweight I/O accelerators for

queue management and completion coalescing. These efforts point toward IOPS-scalable architectures where hosts, interconnects, and storage evolve jointly.

Algorithmic design space exploration. The collapse of caching threshold to seconds blurs the traditional boundary between memory and storage, opening a broad design space for SSD-resident algorithms and data structures that treat flash as an active tier. Rather than prescribing specific mechanisms, we emphasize the scope: access-path design and scheduling at high IOPS; tier-aware data layouts and placement; lightweight ordering, consistency, and recovery tuned to seconds-scale reuse; and QoS, fairness, and isolation under multi-tenant contention. Exploring this spectrum through cross-layer co-design can yield a new class of SSD-resident systems purpose-built for the seconds-scale regime.

9 Conclusion

This work reexamines the five-minute rule from first principles and recasts it as a feasibility-aware provisioning framework for AI-era systems. Our analysis shows that, under our stated assumptions, when GPU-centric hosts are paired with Storage-Next SSDs engineered for fine-grained random access, the DRAM-to-flash caching threshold collapses from minutes to seconds, effectively promoting NAND flash to an active extension tier of memory. We implemented MQSim-Next to reproduce the key trends and support basic sensitivity studies. We further examined SSD-resident KV store and ANN search as concrete case studies, illustrating the algorithm and data-structure design space unlocked by seconds-scale caching. Overall, this work turns a heuristic into a quantitative, cross-layer framework for the AI era, laying a foundation for seconds-scale memory hierarchies that bridge device physics, system design, and algorithm co-optimization.

References

[1] 2025. CacheLib. https://github.com/facebook/CacheLib.
[2] 2025. Redis — The Real-time Data Platform. https://redis.io.
[3] 2025. RocksDB — A Persistent Key-Value Store for Flash and RAM Storage. https://rocksdb.org/.
[4] 2025. WiredTiger — High-Performance Storage Engine. https://github.com/wiredtiger/wiredtiger.
[5] Raja Appuswamy, Goetz Graefe, Renata Borovica-Gajic, and Anastasia Ailamaki. 2019. The five-minute rule 30 years later and its impact on the storage hierarchy. *Commun. ACM* 62, 11 (2019), 114–120.
[6] Ben Berg, Daniel Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, , and Gregory Ganger. 2020. The CacheLib caching engine: Design and experiences at scale. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.
[7] Badrish Chandramouli, Guna Prasaad, Donald Kossmann, Justin Levandoski, James Hunter, and Mike Barnett. 2018. Faster: A concurrent key-value store with in-place updates. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 275–290. https://doi.org/10.1145/3183713.3196898
[8] Wooseong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, Chulseung Lee, Youra Choi, Shine Kim, Dongku Kang, and Geunyeong Yu. 2018. A flash memory controller for 15µs ultra-low-latency SSD using high-speed 3D NAND flash with 3µs read time. In *IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 338–340.
[9] CMU-SAFARI. 2024. . Github. https://github.com/CMU-SAFARI/MQSim
[10] Christian Monzio Compagnoni, Akira Goda, Alessandro S Spinelli, Peter Feeley, Andrea L Lacaita, and Angelo Visconti. 2017. Reviewing the evolution of the NAND flash technology. *Proc. IEEE* 105, 9 (2017), 1609–1633.
[11] Sampath Deegalla and Henrik Bostrom. 2006. Reducing high-dimensional data by principal component analysis vs. random projection for nearest neighbor classification. In *International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 245–250.

---

[4]Our results are illustrative and model-based, not a direct performance comparison with DiskANN.

[12] DigiTimes. [n. d.]. *Samsung revives Z-NAND after 7 years to supercharge AI with 15x speed gains.* https://www.digitimes.com/news/a20250808VL210/samsung-3d-nand-technology-ai.html

[13] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 1–32.

[14] Mingjing Du, Shifei Ding, and Hongjie Jia. 2016. Study on density peaks clustering based on k-nearest neighbors and principal component analysis. *Knowledge-Based Systems* 99 (2016), 135–145.

[15] Jianyang Gao and Cheng Long. 2023. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[16] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. 2023. Filtered-DiskANN: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference.* 3406–3416.

[17] Goetz Graefe. 2007. The five-minute rule twenty years later, and how flash memory changes the rules. In *Proceedings of the International Workshop on Data Management on New Hardware.* 1–9.

[18] Jim Gray and Goetz Graefe. 1997. The five-minute rule ten years later, and other computer storage rules of thumb. *ACM Sigmod Record* 26, 4 (1997), 63–68.

[19] Jim Gray and Franco Putzolu. 1987. The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. In *Proceedings of the ACM international conference on Management of data (SIGMOD).* 395–398.

[20] Mor Harchol-Balter. 2013. *Performance modeling and design of computer systems: queueing theory in action.* Cambridge University Press.

[21] Zongliang Huo, Weihua Cheng, and Simon Yang. 2022. Unleash scaling potential of 3D NAND with innovative Xtacking® architecture. In *IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits).* 254–255.

[22] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems* 32 (2019).

[23] Ali Khakifirooz, Sriram Balasubrahmanyam, Richard Fastow, Kristopher H. Gaewsky, Chang Wan Ha, Rezaul Haque, Owen W. Jungroth, Steven Law, Aliasgar S. Madraswala, Binh Ngo, Naveen Prabhu V, Shantanu Rajwade, Karthikeyan Ramamurthi, Rohit S. Shenoy, Jacqueline Snyder, Cindy Sun, Deepak Thimmegowda, Bharat M. Pathak, and Pranav Kalavade. 2021. A 1Tb 4b/cell 144-tier floating-gate 3d-nand flash memory with 40mb/s program throughput and 13.8 gb/mm2 bit density. In *IEEE International Solid-State Circuits Conference (ISSCC),* Vol. 64. 424–426.

[24] Yousef A. Khalidi and Moti N. Thadani. 1995. *An Efficient Zero-Copy I/O Framework for UNIX.* Technical Report. USA.

[25] John FC Kingman. 1961. The single server queue in heavy traffic. In *Mathematical Proceedings of the Cambridge Philosophical Society,* Vol. 57. Cambridge University Press, 902–904.

[26] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. 2010. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.* 39, 4 (2010), 1543–1561.

[27] Leonard Kleinrock. 1975. *Queueing Systems. Volume 1: Theory.* Wiley-Interscience.

[28] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. 2022. Matryoshka representation learning. *Advances in Neural Information Processing Systems* 35 (2022), 30233–30249.

[29] Justin J. Levandoski, David B. Lomet, and Sudipta Sengupta. 2013. The Bw-Tree: A B-tree for new hardware platforms. In *IEEE International Conference on Data Engineering (ICDE).* 302–313.

[30] Hyeontaek Lim, Dongsu Han, David G Andersen, and Michael Kaminsky. 2014. MICA: A Holistic Approach to Fast In-Memory Key-Value Storage. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 429–444.

[31] Shu Lin and Daniel J. Costello Jr. 1983. *Error control coding - fundamentals and applications.* Prentice Hall.

[32] Rey Luna. 2023. Introduction to JEDEC NAND Separate Command Address (SCA) Protocol. In *Flash Memory Summit (FMS).*

[33] Florence Jessie MacWilliams and Neil James Alexander Sloane. 1977. *The theory of error-correcting codes.* Vol. 16. Elsevier.

[34] Yu Malkov and Dmitry Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[35] Rino Micheloni, Alessia Marelli, and Kam Eshghi. 2013. *Inside solid state drives (SSDs).* Springer.

[36] CJ Newburn. 2024. GPUs as Data Access Engines. Future Memory & Storage Technologies Conference (FMST).

[37] CJ Newburn, Prashant Prabhu, and Vikram Sharma Mailthody. 2025. Storage Next for AI: How to Eliminate the Memory Wall for GenAI and LLM Workloads. https://www.nvidia.com/en-us/on-demand/session/gtc25-s73012/. NVIDIA GTC 2025, Session S73012.

[38] Nikkei xTECH. [n. d.]. *Kioxia to Receive 100x Faster SSD for AI in 2027.* https://xtech.nikkei.com/atcl/nxt/column/18/00001/11065/

[39] Open NAND Flash Interface Workgroup (ONFI). [n. d.]. ONFI Specifications. https://onfi.org/specs.html.

[40] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144.

[41] Jeongmin Brian Park, Vikram Sharma Mailthody, Zaid Qureshi, and Wen-mei Hwu. 2024. Accelerating Sampling and Aggregation Operations in GNN Frameworks with GPU Initiated Direct Storage Accesses. *Proc. VLDB Endow.* 17, 6 (Feb. 2024), 1227–1240.

[42] Ted Pekny, Luyen Vu, Jeff Tsai, Dheeraj Srinivasan, Erwin Yu, Jonathan Pabustan, Joe Xu, Srinivas Deshmukh, Kim-Fung Chan, Michael Piccardi, Kevin Xu, Guan Wang, Kaveh Shakeri, Vipul Patel, Tomoko Iwasaki, Tongji Wang, Padma Musunuri, Carl Gu, Ali Mohammadzadeh, Ali Ghalam, Violante Moschiano, Tommaso Vali, Jaekwan Park, June Lee, and Ramin Ghodsi. 2022. A 1-Tb Density 4b/Cell 3D-NAND Flash on 176-Tier Technology with 4-Independent Planes for Read using CMOS-Under-the-Array. In *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers,* Vol. 65. 1–3.

[43] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, Chris J Newburn, Dmitri Vainbrand, I-Hsin Chung, Michael Garland, William Dally, and Wen mei Hwu. 2023. GPU-initiated on-demand high-throughput storage access in the BaM system architecture. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* 325–339.

[44] SanDisk Corporation. 2025. The Diversification of Flash Storage: Unlocking the Full Potential of NAND in the AI Era. In *Keynote Address, Flash Memory Summit (FMS): The Future of Memory and Storage.* Santa Clara, CA, USA.

[45] Tatsuo Shiozawa, Hirotsugu Kajihara, Tatsuro Endo, and Kazuhiro Hiwada. 2020. Emerging usage and evaluation of low latency FLASH. In *IEEE International Memory Workshop (IMW).* IEEE, 1–4.

[46] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A framework for enabling realistic studies of modern Multi-Queue SSD devices. In *USENIX Conference on File and Storage Technologies (FAST 18).* 49–66.

[47] Kilian Q Weinberger and Lawrence K Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *Journal of machine learning research* 10, 2 (2009).