Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Hierarchical File-Level Cache Access Prediction for Distributed Scientific Experiments: A Case Study in High-Energy Physics with US CMS

Sarat Sasank Barla

sbarla2@huskers.unl.edu

University of Nebraska–Lincoln

Venkat Sai Suman Lamba Karanam

Bowling Green State University

Byrav Ramamurthy

University of Nebraska–Lincoln

Derek Weitzel

University of Nebraska–Lincoln

Research Article

Keywords: High Energy Physics (HEP), Compact Muon Solenoid (CMS), Large Hadron Collider (LHC), eXtended Root Deamon (XRootD), Any Data, Anytime, Anywhere (AAA)

Posted Date: June 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9809323/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Hierarchical File-Level Cache Access Prediction for Distributed Scientific Experiments: A Case Study in High-Energy Physics with US CMS

Sarat Sasank Barla[1], Venkat Sai Suman Lamba Karanam[2], Byrav Ramamurthy[1], Derek Weitzel[1]

[1]School of Computing, University of Nebraska–Lincoln, Lincoln, NE, USA.

[2]Department of Computer Science, Bowling Green State University, Bowling Green, OH, USA.

Contributing authors: sbarla2@huskers.unl.edu; vlambak@bgsu.edu; ramamurthy@unl.edu; dweitzel@unl.edu;

Abstract

Modern High-Energy Physics (HEP) workflows operate in highly distributed computing environments in which large-scale experimental data must be efficiently accessed across geographically dispersed resources. In Compact Muon Solenoid (CMS) ecosystem, data produced at CERN is distributed aross a tiered infrastructure and is accessed through federated systems such as XRootD, Any Data, Anytime, Anywhere (AAA), and cache nodes or data origins that are often deployed near compute facilities. While this architecture offers scalability and broad accessibility, the current caching strategies are inherently reactive – for example, Least Recently Used (LRU) and Least Frequently Used (LFU) often fail to capture the temporal and structural characteristics of scientific data access workloads.

In this work, we present a hierarchical predictive framework for cache optimization that combines dynamic workload understanding, directory-level forecasting, mathematical ranking, and file-level refinement. We realize this effort with the following steps. (1) We first performed data preprocessing and access-pattern analysis on cache records collected from SoCal Repo, which is part of the CMS experiment, spanning a nine-month period in Jan.-Sept. 2025 comprising `7.2 million` cache records. (2) We then developed an an improved Deep Neural Network (DNN) to predict future directory accesses. To this end, we improve the

resulting learned model trained on the cache records by integrating a probabilistic formulation based on popularity and recency, and systematically compare alternative temporal scoring strategies, including exponential decay, harmonic mean, and linear moving average. (3) On the directories thus predicted, we perform file-level prediction using a second model – another much simpler DNN – that incorporates the file-level temporal features of popularity, recency, and file size as predictive features. (4) The resultant full pipeline is evaluated against cache-oriented performance metrics, including hit rate and precision, to assess its effectiveness in improving data locality and access efficiency.
Overall, this study shows a 5%-10% improvement in cache hit rates, more than 50% precision accuracy in directory-level predictions, and provides a structured and practical approach toward predictive caching for distributed scientific infrastructure.

Keywords: High Energy Physics (HEP), Compact Muon Solenoid (CMS),Large Hadron Collider (LHC), eXtended Root Deamon (XRootD), Any Data, Anytime, Anywhere (AAA)

1 Introduction

The unprecedented growth of data generated by modern scientific instruments in recent years has made efficient data access a cardinal challenge in large-scale distributed computing environments. In High-Energy Physics (HEP), experiments such as the Compact Muon Solenoid (CMS) [1] at the Large Hadron Collider (LHC) [2] produce massive volumes of data – often over several petabytes annually – that must be processed, distributed, and analyzed by geographically dispersed computing facilities connected via high-speed Wide-Area Networks (WAN). In CMS, data produced at CERN is distributed across a multi-tiered infrastructure composed of mainly four tier, namely, Tiers-0–3. The user analysis jobs are often submitted through frameworks such as CRAB [3], access these datasets through federated mechanisms including XRootD [4] and AAA [5], while caching framework such as XCache [6] are used to reduce repeated wide-area transfers by improving data locality. Although this distributed architecture provides scalability and flexibility, the underlying caching mechanisms are largely static and hence they do not adapt to dynamic workload behavior, heterogeneous data demand, and limited cache capacity [7–10]. Commonly, repeated requests for popular datasets (files and directories) can incur access latency and network overhead that can be avoided with adaptive data locality mechanisms. Traditional cache management approaches, such as Least Recently Used (LRU), Least Frequently Used (LFU) [11], and other heuristic policies are currently used because of their simplicity and low overhead. However, such methods are fundamentally reactive, they make the retention and eviction decisions only after data is accessed, without accounting for the future demand. Experiments running on these distributed scientific infrastructures are often characterized by temporal locality, shifting popularity, and structured directory-level access patterns – here reactive policies often fail to retain

the most valuable content in the cache [12, 13]. As a result, there is exigency in predictive approaches that can adaptively anticipate future accesses and thereby improve cache efficiency.

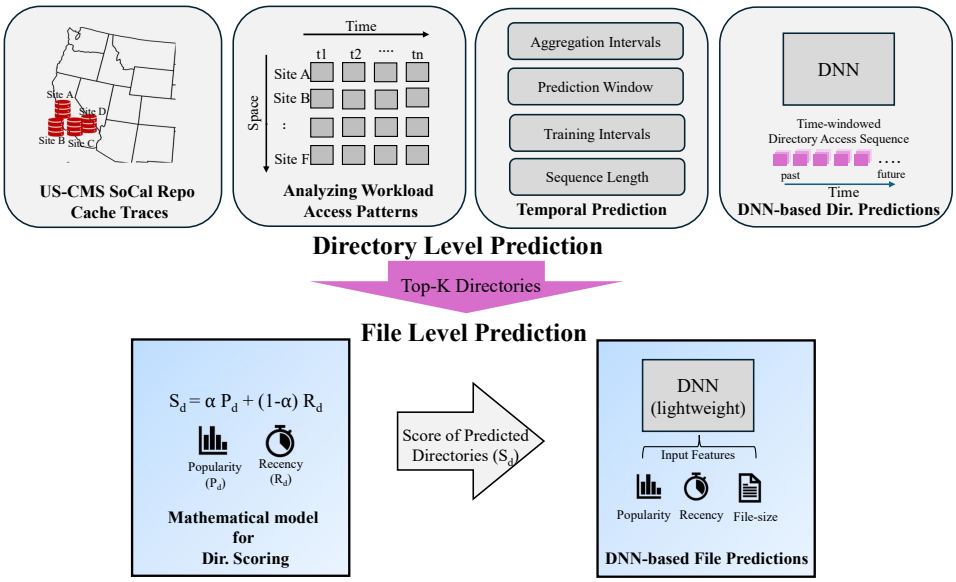


Fig. 1 The figure presents the proposed hierarchical predictive framework for distributed scientific caching. Following a data preprocessing and filtering step, the temporal correlations at the filesystem directory-level are extracted and passed to a DNN which makes directory-level future access predictions. On the top-K directories thus predicted, a statistical modeling is applied over popularity and recency, and then passed down to a file-level access prediction based on a much simpler DNN.

To address these challenges, this work presents a hierarchical predictive framework for distributed scientific caching illustrated in Fig. 1. We first begin by extensive data collection of cache access traces from HEP-oriented workflows – we collected `7.2 million` cache records from the SoCal Repo, which is part of the CMS experiment, spanning a nine-month period in Jan.-Sept. 2025. Our proposed approach first performs a data preprocessing and workload understanding to extract meaningful access patterns from raw logs. It then predicts future directory accesses using an improved Deep Neural Network (DNN) model, capturing coarse-grained structural behavior in the filesystem namespace. We then complement this learning-based stage with a mathematical model based on popularity and recency is introduced to rank likely future accesses. Since recency modeling plays an important role in cache behavior [], this study further compares multiple temporal scoring formulations– exponential decay, harmonic mean, and linear moving average. Finally, we perform file-level prediction within the predicted directories using a simpler DNN model that incorporates features such as popularity, recency, and file size.

The overall objective of this work is to improve data locality and cache effectiveness in distributed HEP data systems by combining hierarchical prediction and temporal access modeling. The proposed framework is evaluated using cache-oriented metrics, including hit rate and precision, in order to assess its ability to better align cached content with future access demand. By integrating directory-level forecasting, mathematical ranking, and file-level refinement, this study provides a structured approach to predictive caching in large-scale scientific workflows.

We summarize the main contributions of this work below.

1. We present a large-scale analysis of CMS cache access traces from the SoCal Repo located in Southern California, which covers nine-month duration – January to September 2025 – comprising approximately `7.2 million` cache records. Our analyses identifies temporal access patterns, cache-node heterogeneity, directory-level and file-level reuse behavior as important aspects of predictive caching for distributed HEP workflows.
2. We propose a hierarchical framework to predict cache accesses in distributed compute systems. Our framework first forecasts future directory-level activity and then applies a second forecasting at the file level. This two-layer design reduces complexity of direct file-level forecasting while still adhering to the inherent hierarchical structure of CMS data access paths.
3. We develop and evaluate a mathematical ranking approach over popularity-recency metrics for cache access prediction. This approach compares multiple temporal scoring strategies, including exponential decay, harmonic decay, and linear moving average – all of which were found to be critical to study how different recency assumptions affect cache access prediction performance.
4. We present a file-level cache prediction approach as the second stage after directory-level prediction. This neural ranking stage predicts file-level cache access within the predicted directories by the directory-level approach. This stage uses file-level features such as popularity, recency, and file size. This stage is designed to enable finer-grained selection of cache files which are most likely to be accessed in the near future.
5. We evaluate the hierarchical framework using metrics such as directory hit rate, file hit rate, byte hit rate, and precision. Our evaluation showed that the proposed approach improves cache relevance and achieves approximately 5%–10% improvement in cache hit rates under the CMS workloads analyzed from the SoCal Repo.

The remainder of this paper is organized as follows. Section 2 presents the background and system context, including the CMS data ecosystem, tiered data distribution, federated data access, and cache infrastructure. Section 3 describes the problem formulation and overall methodology. Section 4 discusses experimental results and the inference, followed by conclusions and future directions in Section 5.

2 Background and Related Work

2.1 CMS Data Production, Distribution, and Access Workflow

The Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) produces and processes extremely large volumes of scientific data. This data originates from proton-proton collision events recorded by the CMS detector, as well as from large-scale Monte Carlo simulation campaigns used to model expected physics processes. After data is collected or simulated, it passes through multiple stages of reconstruction, reduction, storage, distribution, and analysis. The resulting workflow forms a globally distributed data ecosystem in which physicists access files from many computing sites rather than from a single centralized storage system.

As the raw and fully reconstructed datasets are very large, CMS provides multiple derived data tiers with different levels of detail. The Analysis Object Data (AOD) provides detailed reconstructed event information but is relatively large. MiniAOD reduces this size while retaining commonly used physics objects, and NanoAOD offers an even more compact format optimized for fast analysis. Once datasets are produced, they are stored and distributed across the Worldwide LHC Computing Grid (WLCG), a global federation of computing and storage resources. In the traditional tiered model, CERN acts as the primary Tier-0 site. Large national or regional Tier-1 centers provide storage, processing, and large-scale data distribution. Tier-2 sites typically support simulation production and user analysis workloads.

The movement and placement of CMS datasets are coordinated through data management systems such as Rucio which manages dataset replicas, file locations, transfer rules, and data placement policies across distributed storage endpoints. Rather than requiring users to know the physical location of every file, Rucio maintains metadata about where datasets and files are available and helps orchestrate transfers between sites. In the user-level analysis, CMS scientists submit distributed jobs using tools such as CRAB, the CMS Remote Analysis Builder. The system divides the task into many smaller jobsand these jobs run at different sites depending on resource availability, data locality, and scheduling decisions. The remote data access is enabled by the XRootD data federation. If a job requests a file that is not available locally, the XRootD federation can redirect the request to a remote site that hosts the file. However, remote reads may increase latency, consume wide-area network bandwidth, and create additional load on storage servers and network paths.

To reduce the overhead of repeated remote reads, caching layers such as XCache are deployed at selected sites or regional locations. An XCache server can store files that have been recently or frequently accessed through the federation. When a user job requests a file, the request may first be served by the local or regional cache if the file is already present. This results in a cache hit and avoids fetching the file again from a remote origin. If the file is not present in the cache, the cache retrieves it from the appropriate remote storage endpoint and may keep a copy for future requests. In this way, caches act as intermediate data-access accelerators between user jobs and remote storage systems.

A simplified user request flow can be described as follows. A CMS user prepares an analysis task using CRAB and selects a dataset, for example in MiniAOD or NanoAOD

format. CRAB submits multiple jobs to the distributed computing infrastructure. When a job starts, it attempts to read its assigned input files. If the file is available from local storage, the job reads it directly. If the file is not local, the request may be routed through the XRootD federation. The federation identifies a source that can provide the file, and the data may be streamed remotely to the job. If an XCache instance is present along the access path, it checks whether the requested file is already cached. A cache hit allows the file to be served quickly from the cache, while a cache miss causes the file to be fetched from a remote origin and then potentially stored for later reuse.

2.2 Conventional Cache Management Strategies and In-network Caching

Classical cache replacement strategies such as Least Recently Used (LRU), Least Frequently Used (LFU), and their variants remain widely used because they are simple, interpretable, and computationally lightweight [14, 15]. These methods are designed to exploit temporal locality or historical frequency without requiring explicit modeling of future behavior. In many traditional workloads, such heuristics provide reasonable performance and are straightforward to implement.

However, scientific data-access workloads differ from conventional web or operating-system cache settings in several important ways. They often exhibit bursty access behavior, evolving demand phases, repeated reuse of structured namespaces, and strong dependency on workflow-driven access patterns rather than purely independent object requests. Under such conditions, reactive policies may fail to retain the most operationally useful content. In particular, policies based only on recency may overreact to transient bursts, while policies based only on frequency may fail to adapt quickly to newly active datasets. These limitations motivate the exploration of predictive alternatives that explicitly estimate future demand rather than relying solely on past replacement history.

In-network caching addresses several limitations of traditional cache deployment by placing storage resources closer to regional users and computing facilities, thereby reducing repeated wide-area transfers, access latency, and backbone bandwidth consumption. Prior deployments such as the SoCal Repo demonstrate these benefits, reporting substantial reuse of cached data and approximately 1.4 PB of network traffic volume savings during the observed period [16]. However, in-network caching improves the placement and delivery infrastructure, it lacks in the decision process governing what content should be retained, evicted, or proactively staged. It does not fully address the predictive challenge of identifying future file- and directory-level demand under evolving CMS workloads. This motivates learning-based cache intelligence that can incorporate temporal access dynamics, semantic file-path structure, and continual adaptation to guide proactive and workload-aware caching decisions.

2.3 Other ML-based Caching strategies

Early learning-based cache strategies commonly used supervised models such as decision trees, random forests, gradient-boosted trees, support vector machines, and

regression models to classify whether an object should be retained, evicted, or prefetched based on features such as access frequency, recency, object size, reuse distance, and historical request counts [17–19]. Such models provide better adaptability than fixed heuristic policies because they can learn nonlinear relationships between workload features and cache utility. However, they generally depend on manually engineered features and often treat each request as an independent decision. As a result, they provide limited ability to capture long-range temporal dependencies, evolving access sequences, and abrupt workload shifts that are common in distributed scientific data environments.

Several works have further applied learning-based cache replacement and admission policies, including approaches that learn when to follow recency-oriented or frequency-oriented behavior. For example, LeCaR combines LRU and LFU decisions through online learning, while Hawkeye learns cache behavior by approximating Belady-style optimal replacement decisions in processor-cache settings [20, 21]. Reinforcement-learning-based cache replacement methods such as Glider have also shown that learned policies can outperform static replacement rules under certain workloads [22]. Although these methods demonstrate the value of adaptive cache intelligence, many of them are designed for web caches, processor caches, or content-distribution settings where objects are smaller, request streams are denser, and the semantics of file paths or scientific datasets are not central to the prediction task. Therefore, their direct applicability to CMS-style distributed caching is limited, where files are large, accesses may be sparse or bursty, and incorrect caching or prefetching decisions can incur significant storage and network-transfer costs.

Deep learning methods have been introduced to address some of these limitations by modeling temporal access behavior more directly. Recurrent neural networks, particularly LSTM- and GRU-based models, are well suited for cache-access prediction because they can learn sequential dependencies from historical request streams and retain information about prior access behavior over time [23, 24]. This property is important in CMS workflows, where future file or directory demand may depend not only on immediate previous accesses but also on longer-running analysis campaigns, dataset releases, and recurring user activity patterns. Attention-based models further strengthen this capability by allowing the model to emphasize the most relevant portions of an access history rather than compressing the entire sequence into a fixed hidden state [25]. Such models are particularly useful when access logs contain mixed short-term bursts and longer-term dependencies, as they can selectively focus on influential time steps, directories, or file groups when estimating future demand.

2.4 Scope of this Work

The present work is situated at the intersection of scientific data-access optimization, predictive caching, and hierarchical workload modeling. Unlike conventional reactive cache policies, the proposed framework explicitly models future access behavior. It is motivated by the observation that CMS-style workloads exhibit meaningful structure at multiple levels, accesses are temporally correlated, file paths are organized hierarchically, and a relatively small subset of directories and files often accounts for a

large portion of repeated demand. These properties make the workload well suited to a hierarchical predictive approach.

Accordingly, this work first performs directory-level prediction in order to identify likely active regions of the namespace, and then refines prediction at the file level using popularity- and recency-aware ranking. In addition to the learning-based component, the framework also studies a mathematical formulation based on popularity and recency, and compares alternative temporal scoring mechanisms including exponential decay, harmonic mean, and linear moving average. This combination allows the study to examine both interpretable analytical ranking and learned fine-grained prediction within a unified pipeline.

This paper extends our earlier conference version by broadening both the analytical scope and the predictive framework. First, the journal version includes a more detailed workload understanding and preprocessing study, with expanded characterization of cache nodes, temporal behavior, access concentration, and reuse patterns. Second, it introduces a mathematical popularity–recency formulation together with a comparative analysis of multiple temporal scoring methods, thereby providing a more systematic treatment of recency modeling in predictive caching. Third, the present work extends the prediction pipeline beyond directory-level forecasting to include file-level prediction within the predicted directories, enabling finer-grained cache relevance assessment. Finally, the evaluation is expanded using cache-oriented verification metrics such as hit rate, byte hit rate, and precision, providing a more comprehensive view of practical predictive performance in distributed scientific cache systems.

3 Methodology and Approach

In this section, we first present our data preprocessing and workload characterization steps used to transform the raw cache logs into a structured representation suitable for modeling. We then present the directory-level prediction approach, followed by the mathematical popularity-recency formulation and the comparative analysis of temporal scoring methods. Finally, we describe the file-level prediction model within the predicted directories and the evaluation methodology used to verify the performance of the overall framework.

3.1 Data Preprocessing

As a first step, we converted the raw cache access traces into a structured analytical dataset suitable for predictive modeling. The source data consists of multi-month CMS XCache logs spanning from January through September 2025. The dataset comprises ∼5M access records collected across 25 cache nodes, encompassing nearly 560,000 unique files and ∼14K unique directories, with a cumulative read volume of approximately 2,344 TB and an average file size of about 2.5 GB. Each record in the trace captures an individual access event and includes attributes such as timestamp, cache node identifier, logical file name, number of bytes read, and file size. Although these fields provide rich operational detail, they must be standardized and transformed before meaningful modeling can be performed.

The preprocessing pipeline begins by loading all monthly log files and concatenating them into a unified dataset. The timestamp field is parsed into a standardized datetime representation, from which additional temporal descriptors are derived, including date, hour of day, day of week, and ISO week number. The logical file name is then mapped to its corresponding directory path, enabling the workload to be examined at both file and directory granularity. In parallel, byte-level fields such as `read_bytes` and `file_size` are converted into numeric form to support downstream traffic and storage analysis.

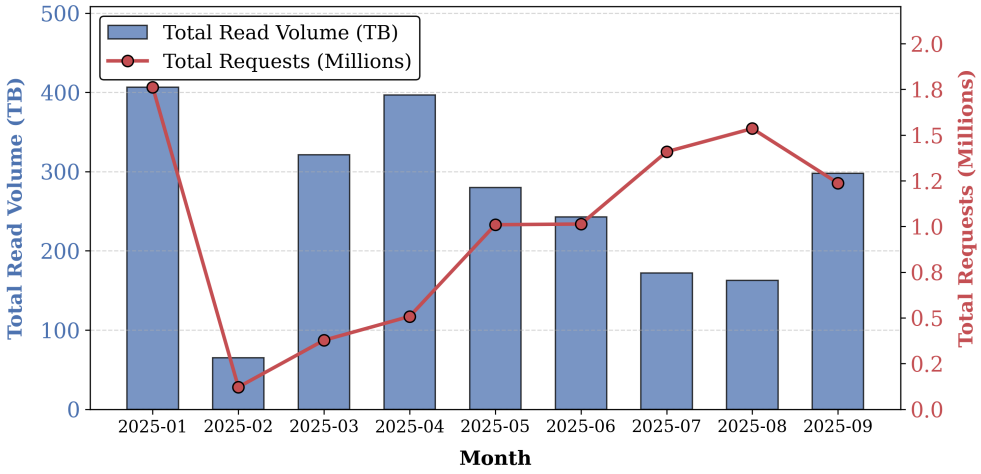


Fig. 2 Monthly CMS XCache Traffic with total number of requests and total read volume from January to September 2025

Besides basic cleaning and enrichment, we also performed a detailed workload understanding analysis to characterize the structure of the dataset before formal modeling begins. Cache nodes are profiled in terms of total requests, total bytes served, unique files, unique directories, and active days, allowing the workload to be examined across different sites and traffic levels. This analysis is important because it determines whether all nodes should be pooled together or whether modeling should focus on dominant high-traffic nodes. Our analysis showed a substantial heterogeneity across nodes and we run our subsequent experiments on selected high-traffic UCSD Tier-2 cache nodes (see Fig. 5)

Temporal access behavior is also examined during this preprocessing stage in order to identify the natural time scale of reuse. By analyzing hourly and daily request patterns, we assessed whether the future behavior should be modeled in fine-grained or coarser time intervals (see Figs. 3, 6). This analysis directly influences the choice of our aggregation intervals used later in the directory-level prediction stage. Similarly, directory-level access concentration is measured to determine whether the workload exhibits a long-tail pattern with a relatively small number of highly popular directories accounting for a substantial portion of requests (see Fig. 7). Such concentration is

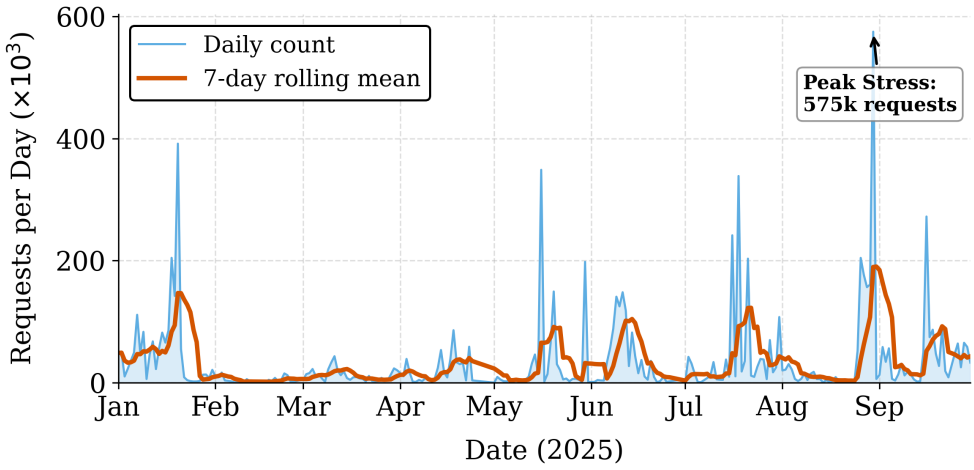


Fig. 3 Daily XCache Request Volume with rolling mean from January to September, 2025

important because it justifies a top-K directory prediction strategy and supports the feasibility of selective prefetching.

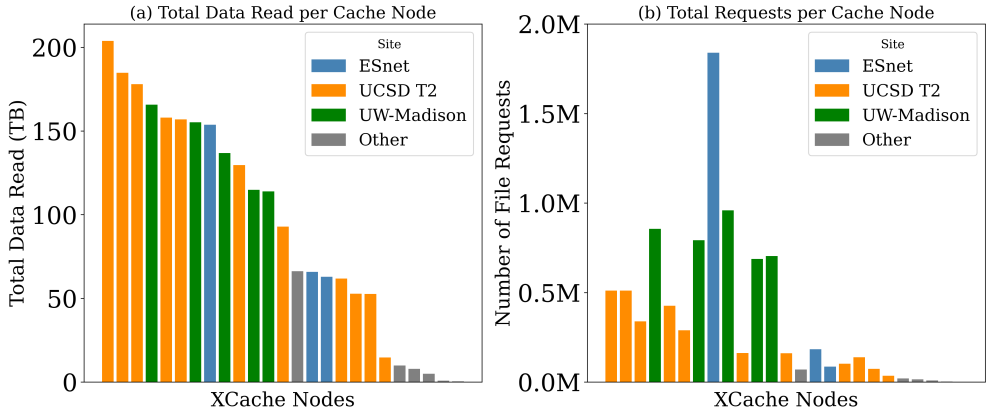


Fig. 4 Left-side: Total data read per cache node from January of September, 2025. Right-side: Total requests per cache node from January of September, 2025

Finally, file reuse characteristics are analyzed to determine whether predictive caching is operationally worthwhile. If files are rarely reused, prefetching and cache prediction would provide limited benefit. In contrast, repeated access to files within relatively short time windows suggests that future requests can be anticipated from historical traces (see Figs. 8, 9). In the provided workflow, this reuse analysis serves as a key justification for the overall predictive framework. The preprocessing stage

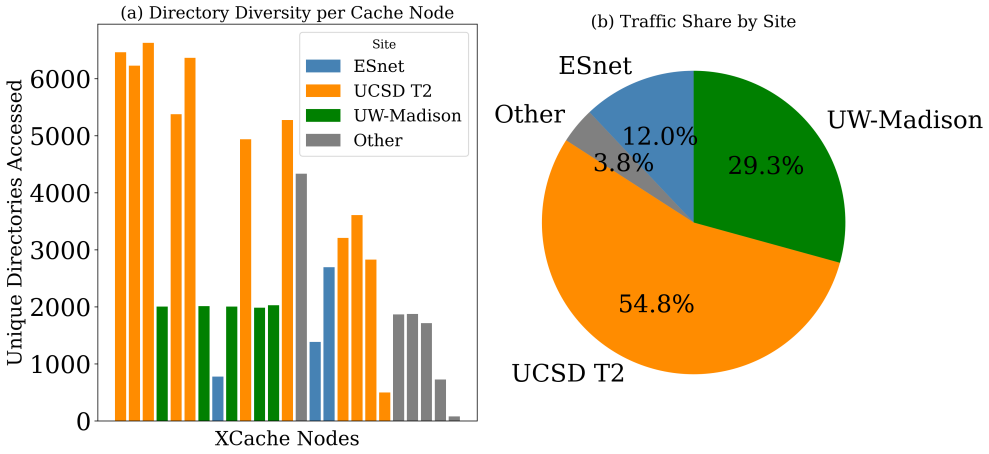


Fig. 5 Left-side: Directory diversity per cache node from January of September, 2025. Right-side: Traffic shared by each cache site from January of September, 2025

not only allows the data for modeling, but also establishes the design choices for the experiments used in the later stages of the methodology.

3.2 Directory-Level Prediction

Following data preprocessing, we performed predictions at the directory-level initially in order to identify the parent directory paths that most likely to experience future activity. This design is analogous to the hierarchical organization of CMS data, where related files are grouped within common directory paths that often reflect shared datasets, campaigns, or processing contexts. Predicting at the directory-level provides a structured and scalable alternative to directly forecasting accesses over the full global file space. The workflow first narrows attention to selected high-traffic cache nodes, with a particular emphasis on UCSD Tier-2 nodes identified during the workload characterization stage. This node selection (see section 3.1) is intended to focus the predictive study on operationally significant access traces while avoiding discrepencies from very sparse or low volume nodes. Within this filtered setting, the data is partitioned using a time-based split, where an initial history window is used for training and a subsequent future window is used for testing.

To model directory activity, the access stream is aggregated into configurable time intervals. Multiple aggregation intervals are considered, including 30 minutes, 1 hour, 2 hours, 4 hours, and 6 hours, in order to assess how temporal granularity influences predictability. Despite using raw request counts alone, the workflow employs a sequence-aware frequency view of directory activity, which captures how consistently a directory appears across the intervals. This formulation is intended to better reflect temporal recurrence and access persistence, rather than allowing short-lived bursts of raw requests to dominate the ranking. The output of this stage is a set of top-ranked candidate directories predicted to be active in a future prediction window. Multiple

prediction windows are explored, including 1, 4, 6, 8, 12, and 24 hours, to understand the relationship between predictive lead time and achievable cache relevance.

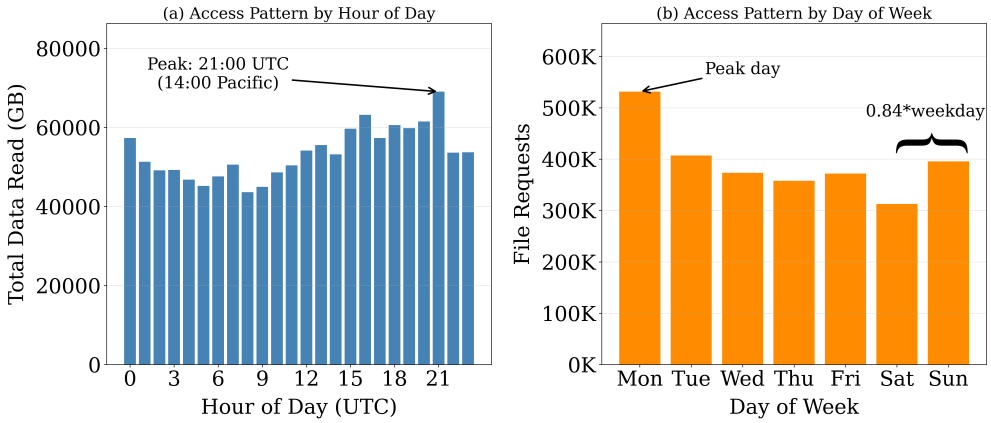


Fig. 6 Temporal Patterns of UCSD XCaches from January to September, 2025

For the model creation, to capture the complex and non-linear temporal dynamics of HEP data workflows, we fundamentally updated the directory prediction architecture from our previously proposed model [7]. Our initial framework relied on a sequential recurrent architecture, utilizing stacked LSTM layers followed by a GRU to process historical access patterns. While effective for short-term sequence modeling, recurrent architectures suffer from sequential bottlenecking and struggle to capture long-range contextual dependencies within highly sparse datasets. To address these limitations, we transitioned to a Deep Neural Network (DNN) driven by Transformer-based Multi-Head Attention mechanisms. In this updated architecture, raw directory access sequences are first mapped into a dense continuous space using categorical embeddings, supplemented by positional encodings to explicitly preserve temporal sequence awareness. We replace the LSTM and GRU layers with stacked Multi-Head Attention blocks and Layer Normalization, allowing the model to evaluate the contextual significance of all historical time-steps simultaneously rather than sequentially. Finally, the sequence features are aggregated via Global Average Pooling and passed into an autoregressive cascade of Dense softmax output heads, independently predicting the top-5 (see Fig. 7) most probable directories for upcoming data workflows. We applied this model on all the four time intervals (1 day, 1 week, 15 days, 1 month) in quest of optimal training time for the data across all the months (see Table 1, Fig. 12) This evolution significantly mitigates gradient degradation across long time-horizons while improving the model's ability to recognize complex, repeating access patterns.

By operating first at the directory level, the methodology reduces the complexity of the file-level prediction problem and introduces a natural hierarchical structure into the overall framework. In this sense, directory-level prediction serves as the first

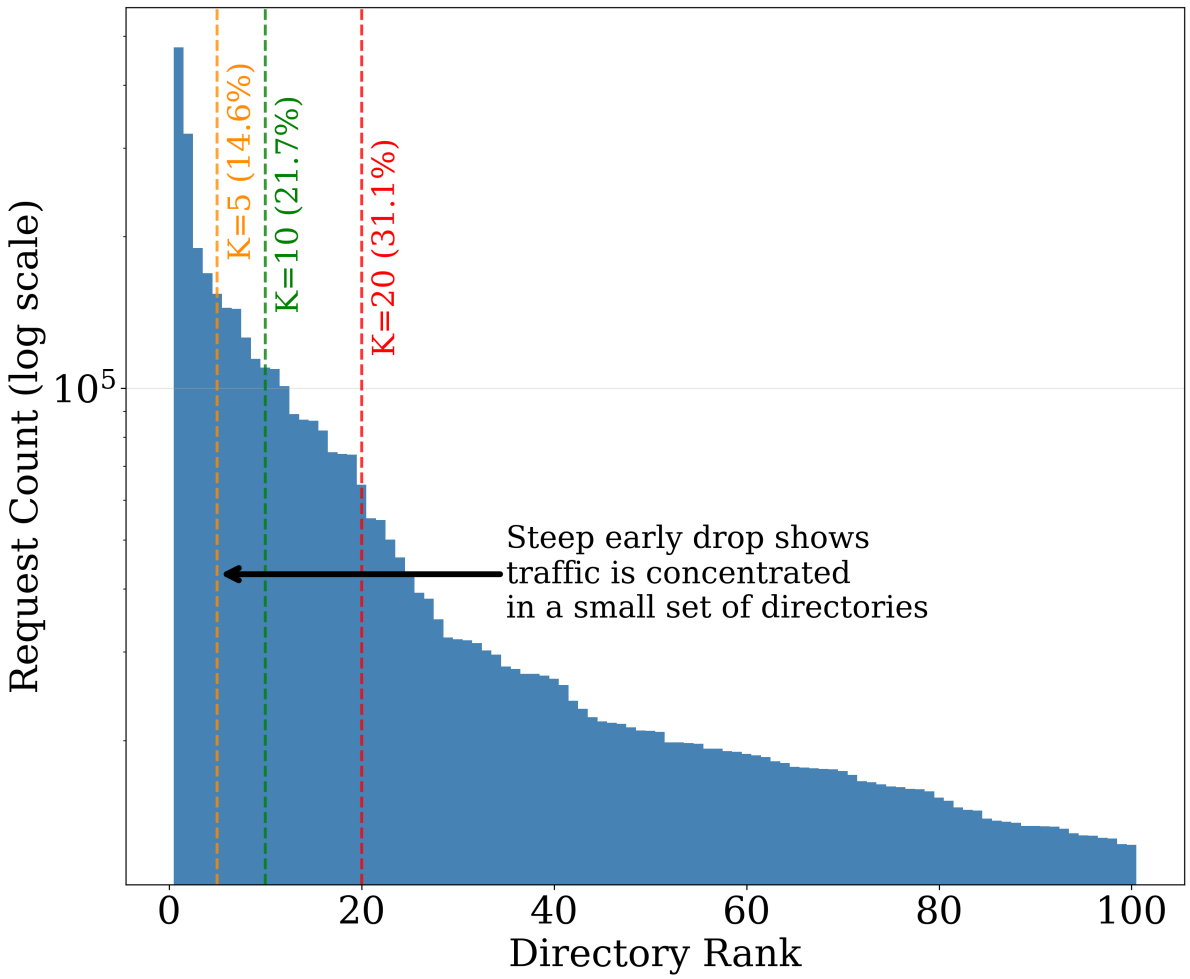


Fig. 7 Long-tail distribution of directories arranging in the descending order of their popularity on the X-axis and Request counts on the y-axis

decision layer, identifying where future demand is likely to emerge before finer-grained file selection is performed.

3.3 Mathematical Modeling using Popularity and Recency

In addition to the sequence-aware directory forecasting model, the proposed methodology incorporates a mathematical ranking formulation based on two fundamental access signals: *popularity* and *recency*. These signals capture complementary characteristics of cache-access behavior. Popularity measures the historical access strength of a directory or file, whereas recency measures how recently the object has appeared in the workload. The motivation for combining these two quantities is that future

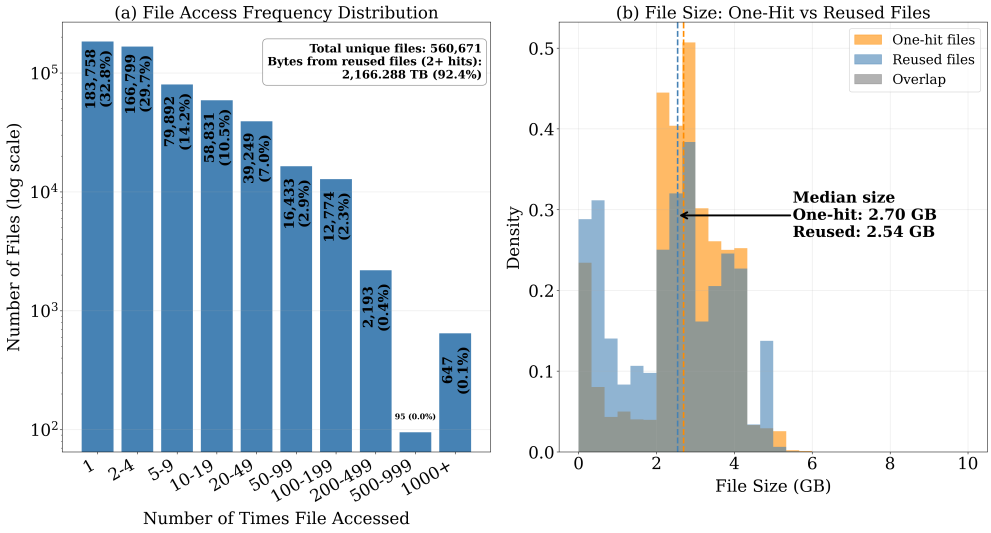


Fig. 8 Left-side: File access frequency distribution from 1 to 1000+ reaccess. Right-side: One-time hit and reused files based on the filesizes

cache accesses are typically influenced by both long-term reuse patterns and short-term temporal locality. Frequently accessed content may remain important because it is associated with active scientific workflows, commonly reused datasets, or repeatedly executed analysis tasks. Similarly, recently accessed content may be accessed again in the near future due to bursty job execution or temporal clustering of user requests.

Let $\mathcal{O}$ denote the set of candidate objects to be ranked, where an object $o \in \mathcal{O}$ may represent either a directory or a file depending on the prediction stage. For each object o, the historical popularity is computed from its observed access frequency within a training window. If N_o denotes the number of accesses to object o, and $\sum_{j\in\mathcal{O}} N_j$ denotes the total number of accesses across all candidate objects, then the normalized popularity score is defined as

$$P_{\text{pop}}(o) = \frac{N_o}{\sum_{j\in\mathcal{O}} N_j}. \tag{1}$$

This normalization converts raw access counts into a probability-like quantity, allowing objects with different absolute access frequencies to be compared within the same candidate space. Objects that appear frequently in the historical trace receive higher popularity scores, while rarely accessed objects receive lower scores. However, popularity alone may overemphasize stale objects that were frequently accessed in the past but are no longer active in the current workload.

To address this limitation, a recency component is introduced. Let t_{current} denote the prediction time and let t_o^{last} denote the most recent timestamp at which object o was accessed. The elapsed time since the last access is given by

$$\Delta t_o = t_{\text{current}} - t_o^{\text{last}}. \tag{2}$$

A smaller value of Δt_o indicates that the object was accessed more recently and is therefore more likely to be relevant under temporal locality. To convert this elapsed time into a bounded recency score, an exponential decay function is used as discussed in As discussed in Observation 1:

$$P_{\text{rec}}(o) = \exp(-\lambda \Delta t_o), \tag{3}$$

where λ is a decay-rate parameter that controls how rapidly the recency score decreases as the object becomes older. A larger value of λ makes the model more sensitive to recent accesses, while a smaller value allows older accesses to retain influence for a longer period. In this formulation, an object accessed very recently receives a score close to one, whereas an object that has not been accessed for a long time receives a score closer to zero.

The final probabilistic score combines popularity and recency through a weighted linear formulation:

$$S(o) = \alpha P_{\text{pop}}(o) + (1 - \alpha) P_{\text{rec}}(o), \tag{4}$$

where $\alpha \in [0, 1]$ controls the relative contribution of popularity and recency. When α is closer to one, the ranking favors historically frequent objects. When α is closer to zero, the ranking becomes more responsive to recently accessed objects. This weighting provides a flexible mechanism to balance persistent workload behavior and short-term access dynamics. The resulting scores are then normalized across all candidate objects to obtain a probability distribution:

$$P(o) = \frac{S(o)}{\sum_{j \in \mathcal{O}} S(j)}. \tag{5}$$

The normalized score $P(o)$ represents the relative likelihood that object o will be accessed in the upcoming prediction window, according to the combined popularity-recency model. Candidate objects are ranked in descending order of $P(o)$, and the top-K objects are selected for downstream cache-aware decision making.

Within the proposed hierarchical workflow, this mathematical model serves two major purposes. First, at the directory level, it provides an interpretable mathematical baseline against which the sequence-aware deep learning model can be compared. This is important because access traces in distributed scientific computing environments often exhibit both recurring historical patterns and short-lived bursts. A purely frequency-based method may fail to adapt to workload shifts, while a purely recency-based method may overreact to temporary spikes. The combined formulation provides a balanced and transparent alternative.

Second, after the directory-level model identifies candidate directories, the popularity-recency formulation is used to refine file-level selection within those predicted directories. Specifically, for each predicted directory d, the set of files contained within that directory is collected as $\mathcal{F}_d$. For each file $f \in \mathcal{F}_d$, the same scoring process is applied using file-level access counts and file-level last-access timestamps:

$$S(f \mid d) = \alpha P_{\text{pop}}(f \mid d) + (1 - \alpha) P_{\text{rec}}(f \mid d). \tag{6}$$

Here, $P_{\text{pop}}(f \mid d)$ represents the normalized popularity of file f among files in directory d, and $P_{\text{rec}}(f \mid d)$ represents the recency-based likelihood of that file. The conditional notation emphasizes that file ranking is performed only within the set of files belonging to the predicted candidate directory, rather than across the entire global file space. This restriction substantially reduces the prediction search space and makes file-level inference more computationally tractable.

The ranked file scores can be used in two ways. In the simpler setting, the score $S(f \mid d)$ is used directly to select the most likely files for proactive caching or prefetching. In the learning-based setting, popularity and recency are used as input features to a downstream file-level prediction model, along with other file-specific attributes such as file size, access count, or temporal statistics. In this way, the mathematical model functions both as an interpretable standalone ranking mechanism and as a feature-generation component for the final prediction stage.

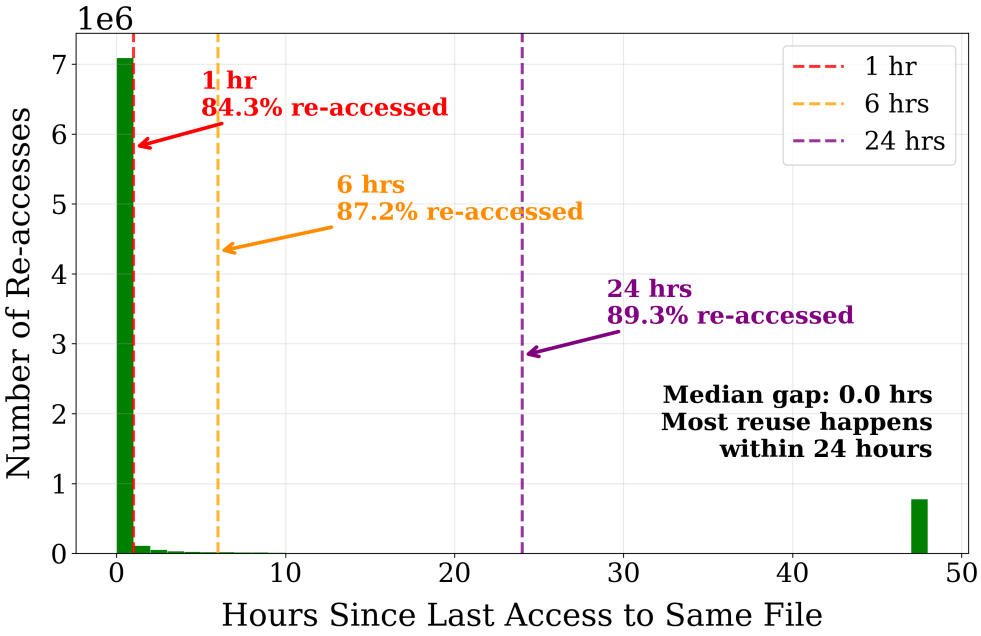


Fig. 9 File reaccess gaps at different time intervals from 1hr, 6hrs, 24 hrs to 50 hrs

Algorithmically, the popularity-recency ranking process can be summarized as follows. For each prediction window, the access history is first filtered according to the selected training interval. The access count and most recent access timestamp are computed for each candidate object. Popularity scores are then obtained by normalizing access counts, while recency scores are computed using an exponential decay function over the elapsed time since the last access. The two scores are combined using the weighting parameter α, normalized across the candidate set, and sorted to obtain the final top-K ranking. This process is computationally lightweight because

it requires only aggregate counts and timestamps, making it suitable for operational cache-management environments.

Overall, the popularity-recency model introduces an interpretable mathematical layer between directory-level prediction and file-level refinement. It complements the deep learning component which is referred to as the black box in operation, by providing a mathematically transparent scoring mechanism that captures both persistent reuse and temporal locality. This combination is particularly useful for distributed cache systems, where access behavior is dynamic, workload patterns evolve over time, and cache-management decisions must be made efficiently with limited computational overhead.

3.4 Comparative Analysis of Temporal Scoring Methods

As the temporal locality is a key characteristic of cache-access behavior, recency must be modeled carefully in the proposed prediction workflow. Hence we wanted to test different recency models on our data before selecting one and applying it. As shown in Fig. 9, file re-access patterns are often influenced by the time elapsed since the previous request. However, different mathematical formulations make different assumptions about how quickly the relevance of a past access should decrease. Therefore, rather than relying on a single fixed recency representation, this stage compares three temporal scoring methods. They are 1) exponential decay, 2) harmonic decay, and 3) moving average linear decay. The objective is to analyze how each formulation affects candidate ranking, file selection, and cache-oriented prediction performance.

Let t denote the current prediction time and let t_i^{last} denote the most recent access time of candidate file i. The elapsed time since the last access is defined as

$$\Delta t_i = t - t_i^{\text{last}}. \tag{7}$$

A smaller Δt_i indicates that the file was accessed more recently, while a larger Δt_i indicates that the file has remained inactive for a longer period. Each temporal scoring method maps this elapsed time into a recency score R_i, where higher values indicate stronger expected relevance for future access.

The first method is the exponential decay model, which assigns rapidly decreasing weights to older accesses. It is defined as

$$R_i^{\text{exp}} = \exp(-\lambda \Delta t_i), \tag{8}$$

where $\lambda > 0$ is the decay-rate parameter. A larger value of λ causes the score to decrease more sharply, making the model highly sensitive to recent activity. This formulation is useful when the workload exhibits strong short-term temporal locality, because files accessed very recently receive substantially higher scores than files accessed further in the past. However, the rapid decay may reduce the influence of files that are accessed periodically but not continuously.

The second method is harmonic decay, which reduces the recency score more gradually:

$$R_i^{\text{harm}} = \frac{1}{1 + \beta \Delta t_i}, \tag{9}$$

where $\beta > 0$ controls the rate of decay. Compared with exponential decay, the harmonic function has a longer tail, meaning that older accesses retain some predictive value for a longer period. This behavior is useful in scientific cache workloads where certain files may be reused after moderate delays due to repeated workflow stages, batch job scheduling, or iterative analysis campaigns. Harmonic decay therefore provides a more stable scoring mechanism when access patterns are not dominated only by immediate reuse.

The third method is a linear moving average decay model. This approach assigns a linearly decreasing score within a fixed temporal window and assigns zero relevance beyond that window. Let W denote the maximum temporal horizon within which a past access is considered useful. The linear recency score is defined as

$$R_i^{\text{lin}} = \max\left(0, 1 - \frac{\Delta t_i}{W}\right). \tag{10}$$

In this formulation, a file accessed at the current time receives a score close to one, while the score decreases linearly as the access becomes older. Once $\Delta t_i \geq W$, the file receives a score of zero. This model approximates a hard temporal window and is useful when cache relevance is expected to expire after a fixed duration. Unlike exponential and harmonic decay, the linear model explicitly removes very old accesses from consideration after the cutoff threshold.

For candidate ranking, each temporal score can be incorporated into the popularity-recency formulation described in Sec. 3.3. For a candidate file i, the combined score is computed as

$$S_i^{(m)} = \alpha P_i + (1 - \alpha) R_i^{(m)}, \tag{11}$$

where P_i is the normalized popularity of file i, $R_i^{(m)}$ is the recency score obtained using method $m \in \{\text{exp}, \text{harm}, \text{lin}\}$, and $\alpha \in [0, 1]$ determines the relative contribution of popularity and recency. The resulting scores are then normalized across the candidate set $\mathcal{F}$:

$$\hat{S}_i^{(m)} = \frac{S_i^{(m)}}{\sum_{j \in \mathcal{F}} S_j^{(m)}}. \tag{12}$$

Files are ranked in descending order of $\hat{S}_i^{(m)}$, and the top-K files are selected as the most likely candidates for future access or proactive cache placement.

The comparison among these temporal scoring methods is meaningful because each function captures a different assumption about workload persistence.

Observation 1. *Exponential decay emphasizes fast adaptation to the most recent workload state (see Fig. 14). Harmonic decay provides a smoother and more persistent memory of past accesses. Linear moving average decay provides an explicit temporal boundary beyond which older accesses are ignored.*

Although all three functions are monotonic with respect to Δt_i, the exponential decay produces better precision rates compared to other two recency models for the

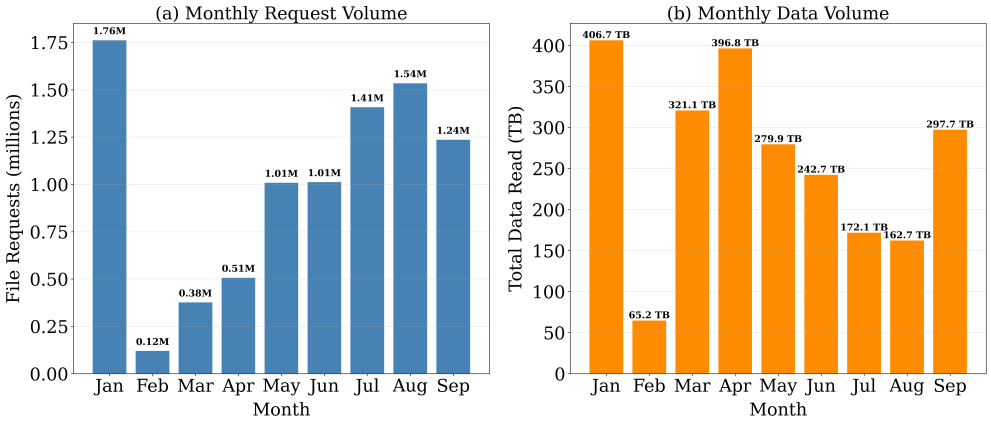


Fig. 10 Monthly data and data bandwidth volume across January to September, 2025

current tested cache data. Hence the exponential decay is selected for calculating the file receny in section 3.3 However the recency models may produce different rankings when combined with popularity, when candidate scores are close, or when threshold-based selection is used.

This comparative analysis therefore evaluates not only the mathematical behavior of the decay curves, as illustrated in Fig. 13, but also their practical impact on cache-aware prediction. In particular, the decay functions are applied to the file candidates associated with the predicted directories, and their resulting top-K rankings are compared against actual future accesses. This allows the study to determine which temporal formulation best reflects the observed reuse behavior in the cache traces and which provides the most effective balance between short-term responsiveness and long-term stability.

3.5 File-Level Prediction using DNN

After identifying likely future directories (from sec. 3.2), we refine prediction at the file level by ranking candidate files within those directories. However, we performed the file-level prediction using the mathematical model (see sec. 3.3) this stage is necessary because practical cache decisions are ultimately wanted to use the predictive and adaptive intelligence to make the optimal decisions. While directory-level prediction identifies regions of likely future activity, file-level ranking determines which specific items within those regions should be retained, prefetched, or prioritized. The file-level workflow is formulated as a neural ranking problem. Candidate files are first gathered from the set of directories predicted by the previous stage, thereby restricting the search space to contextually relevant portions of the namespace. Historical features are then computed for each candidate file. The key features used for training are file popularity, measured as historical access count, and recency, represented by the number of hours since the file was last accessed. These features are extracted from a

historical training interval, while the learning target indicates whether a file appears in a subsequent unseen target window.

A multi-layer perceptron-style neural model is then trained to estimate the probability that a candidate file will be accessed in the target interval. At inference time, the learned score (Neural Score) produced by the neural ranker is combined with a directory-level prior, allowing files in more strongly predicted directories to receive additional emphasis. The final ranked list is obtained by sorting candidate files according to this combined score and selecting the top files within each predicted directory. This file-level neural ranking stage serves as the fine-grained component of the hierarchical framework. It moves beyond fixed analytical formulas by allowing the model to learn a non-linear relationship between historical access features and future file relevance. At the same time, it remains tightly coupled to the directory-level stage, since candidate generation is conditioned on the earlier directory predictions. This hierarchical narrowing strategy improves tractability, preserves contextual information, and more closely reflects the structured organization of scientific datasets.

3.6 Verification and Evaluation

The final stage of the methodology verifies the effectiveness of the proposed framework using cache evaluation metrics. Since the goal of the work is not simply to achieve abstract predictive accuracy, but rather to improve practical cache usefulness, evaluation is performed in terms directly relevant to data access efficiency. Accordingly, the methodology measures performance at multiple levels, including directory hit rate, file hit rate, byte hit rate, and precision.

Directory hit rate evaluates whether the predicted directories overlap with the directories that are actually accessed in the target window. This metric assesses the effectiveness of the coarse-grained first stage of the hierarchical framework. File hit rate measures how well the selected candidate files match the files that are subsequently requested, thereby reflecting the accuracy of the fine-grained file ranking stage. Byte hit rate extends this notion further by weighting successful predictions according to the volume of traffic they capture, thus relating the predictions to the actual data serving burden on the cache. Precision measures the fraction of predicted files that prove to be relevant, thereby assessing how selective and accurate the ranking process is.

The evaluation is conducted using time-based train-test splits in which historical access windows are used to generate predictions for future unseen intervals. This mirrors the actual operational setting of predictive caching, where future requests must be inferred from past observations only. In the workflow reflected in the notebook, both short-horizon validation and broader multi-month analyses are used to understand the consistency of the approach across time. The resulting evaluation framework therefore assesses not only whether the proposed methods can correctly identify future accesses, but also whether they do so in a manner that would be meaningful for practical cache management. Overall, it provides an empirical basis for comparing aggregation choices, forecast windows, recency models, and file-ranking strategies, and it quantifies how effectively the hierarchical prediction pipeline can improve the relevance of cached content under realistic scientific access workloads.

4 Results and Discussion

In this section, we present the main results obtained from the workload characterization, directory-level prediction, temporal recency modeling, and file-level ranking stages of the proposed framework. We first examine the structural and temporal properties of the cache traces to understand the nature of the workload. We then discuss the performance of the directory-level prediction model under different historical training intervals, aggregation resolutions, and forecast windows. Finally, we analyze the effect of alternative recency decay models and interpret the behavior of the file-level neural ranking model.

4.1 Workload Characterization and Temporal Behavior

Figures 4, 5 shows the cache-node characterization for January 2025. The results indicate that the workload is highly skewed across cache nodes, with a small number of nodes contributing the majority of the traffic and request activity. In particular, the UCSD Tier-2 site accounts for the largest traffic share, which justifies focusing the predictive analysis on dominant high-volume nodes. The figure also shows that the busiest nodes exhibit substantial directory diversity, suggesting that the workload is both high-volume and structurally rich.

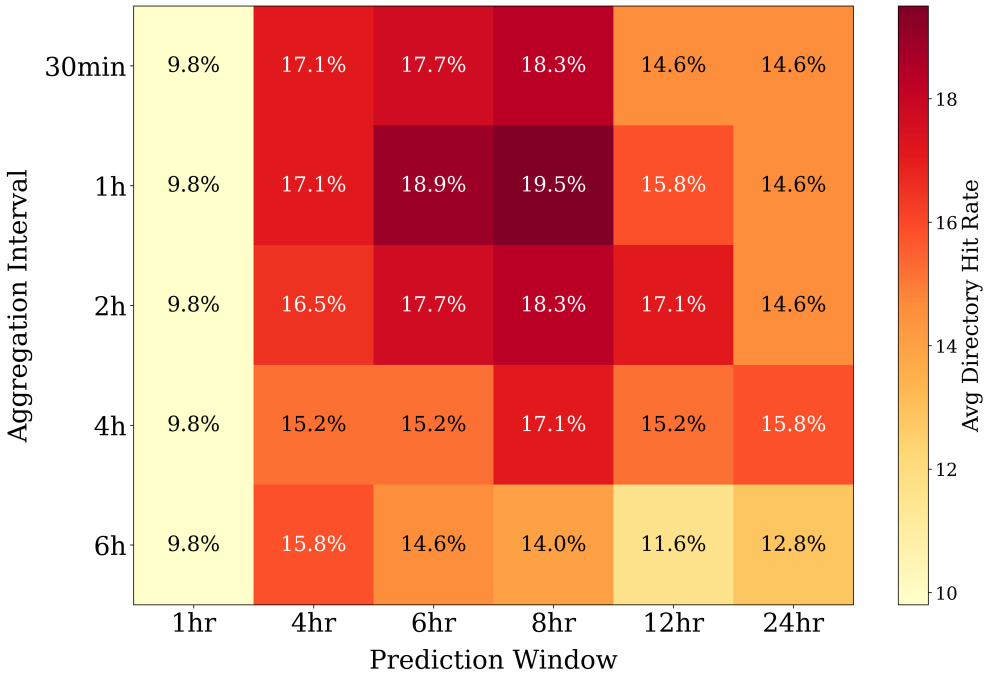


Fig. 11 Heatmap illustrating better aggregation and prediction windows

Figure 3 illustrates the daily request trend from January to September 2025. The workload is clearly non-stationary, with several pronounced bursts of activity separated by lower-traffic periods. This behavior reflects the dynamic nature of user-driven scientific analysis and highlights the limitations of purely reactive cache policies. Figure 2 further shows that the total read volume per request varies across months, indicating that workload variation occurs not only in access frequency but also in data-transfer granularity.

Additionally, we looked at the intra-day behavior and found that the hourly aggregation exhibits repeating daily structure, while the full time-series view shows short-lived spikes superimposed on broader activity cycles. These observations support the use of time-binned prediction and motivate evaluating multiple aggregation intervals in the forecasting stage.

4.2 Directory-Level Prediction Performance

Figure 11 presents the average precision-hit rate achieved under different combinations of aggregation interval and prediction window. The results show that performance depends strongly on both parameters. In general, finer temporal resolutions and intermediate forecast windows provide better results than very coarse resolutions or very long lead times. The strongest performance is observed around the 6–8 hour prediction range, particularly for the 30-minute and 1-hour aggregation settings. This suggests that the workload contains useful short- to medium-term predictability that can be exploited for cache optimization.

Table 1 presents the monthly directory-level prediction precision rates obtained under different training intervals using an 8-hour aggregation setting. Overall, the results show that prediction performance varies considerably across months and training-window configurations, indicating that directory-level access behavior is not uniform over time. The average precision across all months is highest for the 1-day training interval, with an average value of 51.6%, followed by the 15-day interval with 42.6%, the 1-week interval with 35.0%, and the 1-month interval with 33.3%. This suggests that, for the evaluated workload, shorter recent history is often more informative than longer historical windows. Such a trend is consistent with the dynamic nature of cache-access workloads, where recent access patterns may better reflect near-future directory demand than older aggregated behavior.

A key observation from the table is that July achieves the strongest overall prediction performance, with a monthly average of 89.3%. In particular, the 1-week, 15-day, and 1-month training intervals all achieve 100.0% precision for July, indicating a highly stable or repetitive directory-access pattern during that month. This suggests that the workload in July may have contained strong temporal consistency, recurring analysis activity, or concentrated access to a limited set of directories. Similarly, May also shows relatively strong performance, with a monthly average of 58.9%, and maintains consistently high precision across all training intervals. These months demonstrate that when the workload exhibits persistent directory reuse, both short- and long-duration training intervals can provide useful predictive information.

In contrast, February shows no effective prediction performance across the available intervals, with precision values of 0.0% for the 1-week, 15-day, and 1-month settings

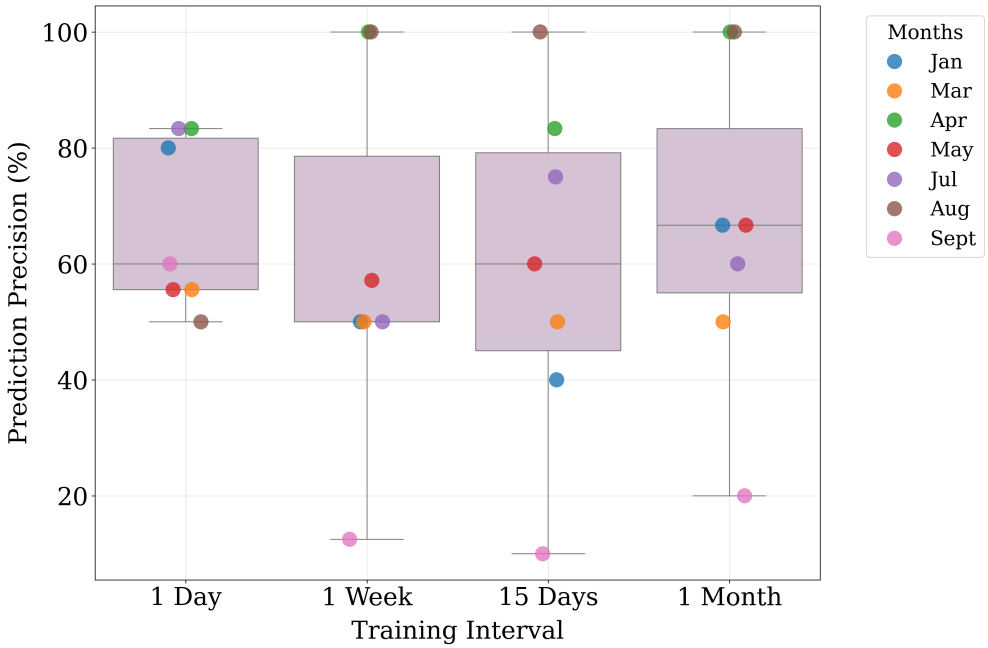


Fig. 12 Distribution of Directory Prediction Precision Across 9 Months and the prediction precision over different training intervals

and no valid value reported for the 1-day interval. This may indicate sparse activity, insufficient training samples, limited overlap between historical and target windows, or a significant shift in directory-access behavior. March and April also show relatively low monthly averages of 8.3% and 27.4%, respectively, suggesting that these months may contain weaker temporal regularity or more dispersed directory accesses. These lower-performing months highlight the challenge of predicting directory demand when workload behavior is highly variable or when future accesses do not strongly resemble recent historical patterns.

Another interesting observation is that no single training interval dominates consistently across all months. For example, the 1-day interval performs best in January, May, and September, while the 1-week interval performs best in June and July. The 15-day interval provides strong performance in January, April, June, July, and August, whereas the 1-month interval is highly effective in July but less effective in several other months. This variation suggests that the optimal training duration depends on the temporal structure of the workload in each month. Shorter windows appear more responsive to rapidly changing access behavior, while longer windows become beneficial when directory accesses are more stable or recurring.

Figure 12 further summarizes directory prediction precision across nine months. The 1-month interval shows comparatively strong and stable performance, while the 1-day and 15-day settings also remain competitive. The variation across months confirms

Table 1 Monthly prediction precision rates across different training intervals with 8-hour aggregation intervals.

Month	Directory-level Precision Rates (%)				Monthly Avg.
	1D	1W	15D	1M	
Jan	71.4	20.0	66.7	66.7	56.2
Feb	–	0.0	0.0	0.0	0.0
Mar	16.7	16.7	0.0	0.0	8.3
Apr	28.6	16.7	50.0	14.3	27.4
May	71.4	57.1	50.0	57.1	58.9
Jun	25.0	66.7	66.7	20.0	44.6
Jul	57.1	100.0	100.0	100.0	89.3
Aug	42.9	25.0	50.0	25.0	35.7
Sep	100.0	12.5	0.0	16.7	32.3
Average	**51.6**	**35.0**	**42.6**	**33.3**	**39.2**

that workload shifts influence predictive quality and suggests that adaptive training windows may be beneficial in practical deployments.

4.3 Recency-based models and their inferences

Figure 13 compares the three recency decay models considered in this work: exponential, harmonic, and moving-average-style linear decay. These functions represent different assumptions about how quickly the influence of past accesses should diminish. Exponential decay emphasizes very recent activity, harmonic decay retains a longer historical tail, and the linear model imposes a bounded relevance window.

The impact of these design choices is shown in Figure 14, which reports file prediction precision across months. Among the three methods, the moving-average-style linear model generally achieves the strongest precision, especially in the later months. Exponential decay performs competitively in some cases, whereas harmonic decay tends to be less effective overall. These results suggest that, for this workload, a bounded but moderately persistent notion of recency is more suitable than either a very aggressive short-term decay or an overly long-tailed formulation.

4.4 File-level predictions using Mathematial model

Table 2 shows the file-level ranking produced using the mathematical popularity–recency scoring model within the predicted directories. The results illustrate how the final score is influenced jointly by historical popularity and temporal freshness. For example, files in Dir A generally have high popularity values, but their recency scores decrease as Δt increases, which lowers their final ranking scores. In contrast, Dir B contains files with smaller Δt values, resulting in stronger recency scores. This allows Dir B File 1 to achieve the highest final score of 0.8449, even though its popularity is lower than several files in Dir A. This highlights the importance of recency in identifying files that may be more relevant for near-future access.

Another useful observation is that files with older access times receive lower final scores, even when they retain moderate popularity. This is visible in Dir C, where

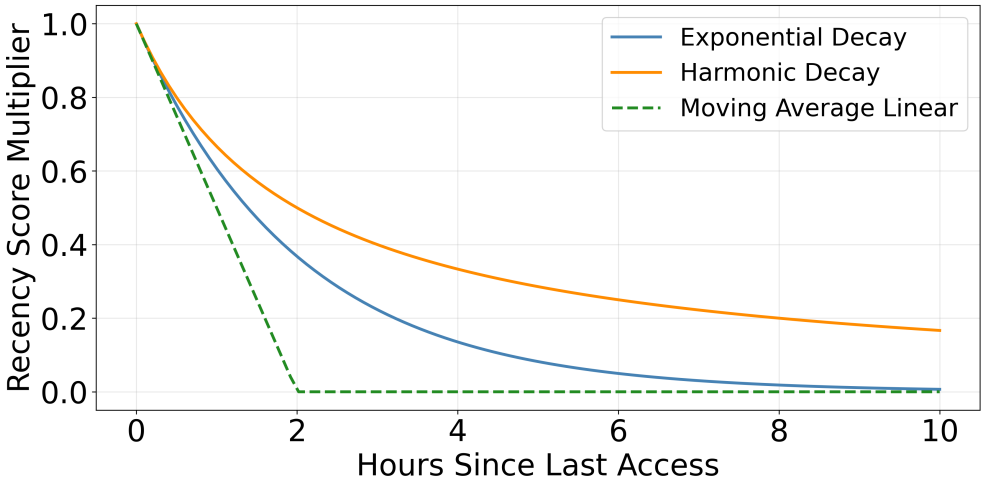


Fig. 13 Comparison of Recency Decay Models for the lambda = 0.5

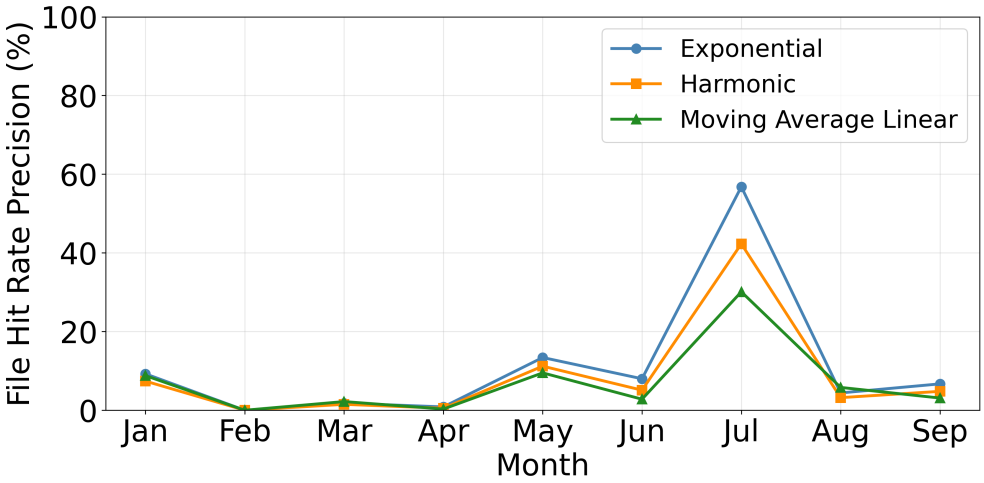


Fig. 14 File Prediction Precision Across 2025 with threshold-based, Top 5 Dirs with lambda = 0.5 per hour

the final score steadily decreases from File 1 to File 5 as both recency and popularity decline. Overall, the table demonstrates that the mathematical model can balance long-term access frequency with short-term temporal locality, making it useful for interpretable file-level ranking before applying the DNN-based prediction model.

Table 3 reports the monthly file-level prediction precision across different training intervals when compared to traditional LRU strategy. The results show that file-level prediction is more challenging than directory-level prediction, which is expected

because the file namespace is much larger and more sparse. Among the training intervals, the 1-month window achieves the highest average precision of 11.24%, followed by the 15-day window with 9.47%, the 1-week window with 6.72%, and the 1-day window with 3.22%. This indicates that file-level prediction benefits more from longer historical context, since individual files may not appear frequently enough in short training windows.

Table 2 File-level ranking scores across predicted directories using the mathematical model.

Rank	Directory	File	Δt	Popularity	Recency	Final Score
1	Dir A	File 1	1.20	0.9420	0.5488	0.7847
2	Dir A	File 2	1.45	0.8910	0.4843	0.7283
3	Dir A	File 3	2.10	0.8750	0.3499	0.6650
4	Dir A	File 4	2.50	0.8240	0.2865	0.6090
5	Dir A	File 5	3.20	0.8120	0.2019	0.5680
6	Dir B	File 1	0.05	0.7580	0.9753	0.8449
7	Dir B	File 2	0.25	0.6910	0.8825	0.7676
8	Dir B	File 3	0.42	0.6540	0.8106	0.7166
9	Dir B	File 4	0.85	0.6210	0.6538	0.6341
10	Dir B	File 5	1.10	0.6050	0.5769	0.5938
11	Dir C	File 1	0.50	0.5820	0.7788	0.6607
12	Dir C	File 2	1.80	0.5610	0.4066	0.4992
13	Dir C	File 3	2.40	0.5100	0.3012	0.4265
14	Dir C	File 4	3.60	0.4850	0.1653	0.3571
15	Dir C	File 5	4.50	0.4500	0.1054	0.3122

Table 3 Monthly mathematical-based model file-level prediction precision increase when compared to LRU across different training intervals.

Month	File-level Precision Rates (%)				Monthly Avg.
	1D	1W	15D	1M	
Jan	2.15	5.50	7.85	9.25	6.19
Feb	0.00	0.00	0.00	0.00	0.00
Mar	0.20	0.85	1.25	1.80	1.03
Apr	0.00	0.35	0.55	0.85	0.44
May	4.50	8.25	11.50	13.40	9.41
Jun	1.55	4.20	6.80	8.00	5.14
Jul	18.20	35.65	48.50	56.80	39.79
Aug	0.50	2.15	3.65	4.40	2.68
Sep	1.85	3.50	5.15	6.70	4.30
Average	**3.22**	**6.72**	**9.47**	**11.24**	**7.66**

July shows the strongest performance across all intervals, with a monthly average of 39.79%, suggesting that file accesses during this month were more repetitive and easier to predict. May and June also show moderate improvements with longer training intervals, indicating the presence of recurring file-level access patterns. In contrast, February has zero precision across all intervals, which may indicate sparse activity, limited file reuse, or a major mismatch between training and prediction windows.

Overall, these results suggest that the mathematical popularity–recency model provides a useful and interpretable baseline for file-level prediction. However, the relatively low precision values also show that file-level access behavior is highly sparse and difficult to capture using fixed scoring rules alone. This motivates the use of the proposed DNN-based file-level predictor, which can learn more complex relationships among popularity, recency, file size, and future file access behavior.

4.5 DNN-based file-level predictions

Table 4 presents the DNN-based file-level prediction precision across different training intervals when compared with the traditional LRU strategy. Compared with the mathematical popularity–recency model, the DNN-based approach shows a consistent improvement across all training intervals. The average precision increases from 3.22% to 3.85% for the 1-day interval, from 6.72% to 7.94% for the 1-week interval, from 9.47% to 11.01% for the 15-day interval, and from 11.24% to 13.06% for the 1-month interval. This improvement indicates that the DNN is able to learn more informative relationships among popularity, recency, and file size than the fixed mathematical scoring function.

A key observation is that longer training intervals continue to provide better file-level prediction performance. The 1-month interval achieves the highest average precision of 13.06%, followed by the 15-day interval with 11.01%. This suggests that file-level prediction benefits from broader historical context, since individual files may be accessed sparsely and may require longer observation windows to capture meaningful reuse behavior. July again shows the strongest prediction performance, with a monthly average of 43.73%, indicating that the file-access pattern during this month was more repetitive and easier for the model to learn.

The DNN-based method also improves performance in months with moderate activity, such as May, June, August, and September. For example, May improves from 9.41% in the mathematical model to 11.20% using the DNN, while September improves from 4.30% to 5.59%. These gains show that the neural model is not only memorizing highly active months but also improving prediction under more varied access conditions. However, February remains at 0.00%, suggesting that when there is insufficient reuse or no meaningful overlap between historical and future accesses, even the learned model cannot recover strong predictive performance.

4.6 Interpretations

Overall, the results show that the CMS cache workload contains meaningful structure at both the temporal and hierarchical levels. The node characterization confirms that traffic is concentrated at a small number of important cache sites, while the temporal

analysis reveals bursty yet partially recurring access behavior. These properties make the workload well suited for predictive caching.

The directory-level results demonstrate that useful coarse-grained predictability can be achieved when suitable aggregation and forecast settings are chosen. At the same time, the gap between directory hit rate and byte hit rate indicates that coarse prediction alone is not sufficient, thereby motivating the file-level refinement stage. The recency-model comparison and DNN-based neural ranking analysis further show that file selection benefits from combining popularity with an appropriate temporal representation rather than relying on a purely reactive policy. Taken together, these findings support the proposed hierarchical framework as a practical approach for improving cache relevance in distributed scientific data environments.

Table 4 Monthly DNN-based file-level prediction precision rates increase when compared to LRU across different training intervals.

Month	**DNN-based File-level Precision Rates (%)**				**Monthly Avg.**
	1D	**1W**	**15D**	**1M**	
Jan	2.65	6.40	8.95	10.85	7.21
Feb	0.00	0.00	0.00	0.00	0.00
Mar	0.50	1.25	2.05	2.85	1.66
Apr	0.00	0.65	1.15	1.65	0.86
May	5.30	10.15	13.45	15.90	11.20
Jun	2.10	5.60	8.30	10.25	6.56
Jul	20.80	39.50	53.20	61.40	43.73
Aug	0.90	3.20	5.15	6.25	3.88
Sep	2.40	4.75	6.85	8.35	5.59
Average	**3.85**	**7.94**	**11.01**	**13.06**	**8.97**

Finally, the DNN-based file-level predictor is better because it moves beyond fixed popularity–recency assumptions and learns non-linear relationships among multiple file-level features. While the mathematical model is useful as an interpretable baseline, the DNN provides greater flexibility by incorporating popularity, recency, and file size together in a data-driven manner. The consistent improvement across all average precision values confirms that the proposed DNN-based refinement is a stronger file-level prediction stage within the hierarchical caching framework.

5 Conclusion and Future Work

In this work, we presented a hierarchical predictive framework for cache optimization in distributed scientific data environments, with a particular focus on CMS cache access workloads. The study was motivated by the limitations of conventional reactive cache replacement policies in the presence of dynamic, bursty, and structurally organized scientific data-access patterns. To address these challenges, the proposed methodology combined workload preprocessing and characterization, directory-level prediction,

mathematical modeling based on popularity and recency, comparative analysis of temporal scoring methods, and file-level prediction within predicted directories.

The results demonstrate that the studied cache traces exhibit meaningful temporal and hierarchical structure that can be exploited for predictive caching. The workload analysis revealed strong concentration of traffic across a subset of dominant cache nodes, along with recurring temporal patterns and non-stationary access behavior. The directory-level prediction stage showed that useful coarse-grained forecasting can be achieved when suitable aggregation intervals and prediction windows are selected. The subsequent file-level refinement further highlighted the importance of combining popularity-aware and recency-aware signals in the prediction methods.

Acknowledgements. This work is sponsored in part by National Science Foundation (NSF) grants with award numbers OAC-2322369, PHY-2323298, OAC-2030508 and Department of Energy (DoE) grant with award number DE-SC0024648. We are also grateful for the US CMS SoCal Repo maintainers, specifically Diego Davila for his support and for providing access to cache access data that was used in this study.

Declarations

Funding

This work was supported in part by the U.S. Department of Energy (DOE) under the AI for HEP project. We acknowledge the support of the University of Nebraska–Lincoln and the computing infrastructure Holland Computing Center (HCC) used in this study.

Conflict of interest

We declare that we have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethics approval and consent to participate

Not applicable. This study does not involve any human participants, human data, or animal subjects.

Data availability

The cache access data used in this study is provided by the U.S. CMS cache operations community for research purposes. Due to operational, institutional, and collaboration-related restrictions, the raw datasets are not publicly available. Aggregated results and derived statistics may be made available from the corresponding author upon reasonable request, subject to approval from the relevant data providers.

Materials availability

Not applicable.

Code availability

The code used for data preprocessing, model development, mathematical-based ranking, file-level prediction, and evaluation can be made available and is subject to institutional and collaboration policies.

Author contribution

Sarat Sasank Barla contributed to conceptualization, methodology, data preprocessing, model development, experimentation, analysis, visualization, and manuscript drafting. Venkat Sai Suman Lamba Karanam contributed in conceptualization, analysis, methodology refinement, technical guidance, experimental discussions, and manuscript drafting. Byrav Ramamurthy contributed to conceptual guidance, project supervision, funding acquisition, and manuscript review and editing. Derek Weitzel contributed to data access, domain expertise, technical support, interpretation of results, and manuscript review. All authors reviewed and approved the final manuscript.

References

[1] Motta, J.: The Compact Muon Solenoid at the Large Hadron Collider, pp. 49–102. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-96288-2_2 . https://doi.org/10.1007/978-3-031-96288-2_2

[2] Evans, L.: The Large Hadron Collider. New Journal of Physics **9**(9), 335 (2007) https://doi.org/10.1088/1367-2630/9/9/335

[3] Codispoti, G., Mattia, C., Fanfani, A., Fanzago, F., Farina, F., Kavka, C., Lacaprara, S., Miccio, V., Spiga, D., Vaandering, E.: CRAB: A CMS Application for Distributed Analysis. IEEE Transactions on Nuclear Science **56**(5), 2850–2858 (2009) https://doi.org/10.1109/TNS.2009.2028076

[4] Dorigo, A., Elmer, P., Furano, F., Hanushevsky, A.: XROOTD/TXNetFile: a highly scalable architecture for data access in the ROOT environment. In: Proceedings of the 4th WSEAS International Conference on Telecommunications and Informatics. TELE-INFO'05. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA (2005)

[5] Bloom, K., Boccali, T., Bockelman, B., Bradley, D., Dasu, S., Dost, J., Fanzago, F., Sfiligoi, I., Tadel, A.M., Tadel, M., Vuosalo, C., Würthwein, F., Yagil, A., Zvada, M.: Any Data, Any Time, Anywhere: Global Data Access for Science. In: 2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC), pp. 85–91 (2015). https://doi.org/10.1109/BDC.2015.33

[6] Sedaghati, A., Hakimi, M., Hojabr, R., Shriraman, A.: X-cache: a modular architecture for domain-specific caches. In: Proceedings of the 49th Annual International Symposium on Computer Architecture. ISCA '22, pp. 396–409.

Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3470496.3527380 . https://doi.org/10.1145/3470496.3527380

[7] Barla, S.S., Karanam, V.S.S.L., Ramamurthy, B., Weitzel, D.: Predicting Future Accesses in XRootD Caching Systems using ML-Based Network Pattern Analysis. In: 2024 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), pp. 1–6 (2024). https://doi.org/10.1109/ANTS63515.2024.10898672

[8] Lamba Karanam, Venkat Sai Suman, Barla, Sarat Sasank, Ramamurthy, Byrav, Weitzel, Derek: Ml-based adaptive prefetching and data placement for us hep systems. EPJ Web Conf. **337**, 01011 (2025) https://doi.org/10.1051/epjconf/202533701011

[9] Barla, S.S., Lamba Karanam, V.S.S., Ramamurthy, B., Weitzel, D.: AI-Driven Strategies for Distributed Caching: A Case Study Using CMS Cache Data. In: Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration. PEARC '25. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3708035.3736094 . https://doi.org/10.1145/3708035.3736094

[10] Barla, S.S., Karanam, V.S.S.L., Ramamurthy, B., Weitzel, D.: Dynamic File Access Prediction for Distributed Caching via Semantic-Aware Continual Learning. In: 2025 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), pp. 1–6 (2025). https://doi.org/10.1109/ANTS66931.2025.11429989

[11] Robinson, J.T., Devarakonda, M.V.: Data cache management using frequency-based replacement. In: Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. SIGMETRICS '90, pp. 134–142. Association for Computing Machinery, New York, NY, USA (1990). https://doi.org/10.1145/98457.98523 . https://doi.org/10.1145/98457.98523

[12] Narayanan, A., Verma, S., Ramadan, E., Babaie, P., Zhang, Z.-L.: DeepCache: A Deep Learning Based Framework For Content Caching. In: Proceedings of the 2018 Workshop on Network Meets AI & ML. NetAI'18, pp. 48–53. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3229543.3229555 . https://doi.org/10.1145/3229543.3229555

[13] Peris, A., Molina, J., Hernández, J.M., Pérez-Calero Yzquierdo, A., Calonge, F.J., Serrano, P., Sikora, A., Vilalta, J.: Comparative analysis of Machine Learning-based eviction techniques and LRU mechanisms for CMS data caching. EPJ Web of Conferences **337**, 01192 (2025) https://doi.org/10.1051/epjconf/202533701192

[14] Belady, L.A.: A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal **5**(2), 78–101 (1966) https://doi.org/10.1147/sj.52.0078

[15] Megiddo, N., Modha, D.S.: Arc: A self-tuning, low overhead replacement cache. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies. FAST '03, pp. 115–130. USENIX Association, USA (2003)

[16] Sim, A., Kissel, E., Guok, C.: Deploying in-network caches in support of distributed scientific data sharing (2022). https://arxiv.org/abs/2203.06843

[17] Herodotou, H.: Autocache: Employing machine learning to automate caching in distributed file systems. In: 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW), pp. 133–139 (2019). https://doi.org/10.1109/icdew.2019.00-21 . IEEE

[18] Sun, H., Sun, C., Tong, H., Yue, Y., Qin, X.: A machine learning-empowered cache management scheme for high-performance ssds. IEEE Transactions on Computers **73**, 2066–2080 (2024) https://doi.org/10.1109/tc.2024.3404064

[19] Giommi, L., Spiga, D., Paladino, M., Kuznetsov, V., Bonacorsi, D.: Developments on the "machine learning as a service for high energy physics" framework and related cloud native solution. IEEE Transactions on Cloud Computing, 1–12 (2025) https://doi.org/10.1109/tcc.2025.3535793

[20] Vietri, G., Rodriguez, L.V., Martinez, W.A., Lyons, S., Liu, J., Rangaswami, R., Zhao, M., Narasimhan, G.: Driving cache replacement with ml-based lecar. In: Proceedings of the 10th USENIX Conference on Hot Topics in Storage and File Systems. HotStorage'18, p. 3. USENIX Association, USA (2018)

[21] Jain, A., Lin, C.: Back to the future: Leveraging belady's algorithm for improved cache replacement. In: Proceedings of the 43rd ACM/IEEE International Symposium on Computer Architecture. ISCA, pp. 78–89 (2016)

[22] Shi, Z., Huang, X., Jain, A., Lin, C.: Applying deep learning to the cache replacement problem. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-52, pp. 413–425. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3352460.3358319 . https://doi.org/10.1145/3352460.3358319

[23] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)

[24] Cho, K., Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 1724–1734 (2014)

[25] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30 (2017)