International Journal of Computational and Electronic Aspects in Engineering
Volume 7, Issue 5, October 2026, pp 533 - 545
https://doi.org/10.26706/ijceae.7.5.20261001

Intelligent Cloud Storage Tiering Using Random Forest and LSTM for Access Pattern Prediction

Anmar B. Salih[1,*], Soud A. Mohamed Amen[2]

[1] Department of Artificial Intelligence Engineering Techniques, Technical Engineering College for Computer and AI, Northern Technical, Mosul, Iraq.
[2] Department of Missions and Cultural Relations, Northern Technical University, Mosul, Iraq.

*Correspondence: anmar.salih@ntu.edu.iq



Abstract: Operationally, it is increasingly burdensome for firms dealing with huge amounts of data to pay for cloud storage. Studies have revealed that a large part of the objects stored in the cloud are never or rarely accessed after the first upload, but due to bad placement policies, they continue to remain in the high-cost storage tiers. In this paper, we present a hybrid machine learning approach that combines Long Short-Term Memory (LSTM) networks with a Random Forest (RF) classifier for the automation of storage tier assignment in cloud object storage systems. The LSTM modelled temporal access patterns to predict a seven-day future access count for each object. This prediction was then used as an attribute along with eight other engineered attributes from raw access logs, and finally, a Random Forest classifier was trained to map this attribute set to one of the three storage tiers. We conduct our experiments on the Microsoft AzureFunctionsBlobDataset2020, which is publicly available. This dataset consists of around 10.7 million blob access events over 287,432 unique objects, collected for a duration of 14 days. The proposed RF+LSTM pipeline obtains classification accuracy of 91.2% and weighted F1-score of 0.903. Based on the Azure Blob Storage pricing, our cost simulation framework estimates a 34.2% cost savings over a static baseline Hot-tier policy, and a 19.4% savings over a time-based threshold policy. An ablation study demonstrates that the access feature contribution to overall accuracy and estimated cost savings by the LSTM is 5.9% and 7.4%, respectively. Scope of dataset, assumptions made in labelling and the difference between simulated and actual deployment savings are discussed openly as limitations.

Keywords: : Cloud Storage Management · Data Tiering · Random Forest · LSTM · Access Pattern Prediction · Cost Optimization · AI-Driven Storage.

Article – Peer Reviewed
Received: 10 August 2026
Accepted: 2 October 2026
Published: 31 October 2026

Copyright: © 2025 RAME Publishers
This is an open access article under the CC BY 4.0 International License.

https://creativecommons.org/licenses/by/4.0/

Cite this article: Anmar B. Salih, Soud A. Mohamed Amen "Intelligent Cloud Storage Tiering Using Random Forest and LSTM for Access Pattern Prediction", International Journal of Computational and Electronic Aspects in Engineering, RAME Publishers, vol. 7, issue 5, pp. 533-545, 2026.
https://doi.org/10.26706/ijceae.7.5.20261001

1. Introduction

The amount of data stored in cloud environments worldwide is increasing rapidly. As per IDC DataAge report [1] the global datasphere has been growing to 175 ZB by 2025, with cloud infrastructure's share increasing year by year. Rising costs of storage for enterprises as well as cloud-native organizations are directly related to this growth. These costs are billed on a monthly basis, depending on both the volume of data being retained and the access tier where it lies. Major cloud providers have designed their pricing models based on storage cost matched with access frequency. In Microsoft Azure Blob Storage, for instance, the Hot tier allows relatively low-latency access and is charged at a higher per-gigabyte storage rate, making it appropriate for frequently accessed data.

The Cool tier minimizes storage costs but results in higher read operation charges. The Archive tier has the lowest storage cost but takes time to rehydrate the data before access. This is based on sound economic reasoning: a seldomly accessed object should not occupy an expensive Hot-tier slot, and a frequently accessed object should not be lost deep in Archive storage where its retrieval will be delayed and costly.

In reality, most deployments were either storing all data in Hot storage by default or applying very basic time-based rules, such as 30 and 90 days, to move objects to Cool and

then Archive, respectively, regardless of actual access patterns by the workload. This is not a great fit for real workloads where access is heterogeneous: some objects can be considered "hot" for months, some are heavily accessed only for particular events, and some are accessed much later in time relative to creation [2, 3]. Machine learning-based tiering provides a more principled alternative. If, indeed, access patterns of a workload can be predicted from historical access patterns and metadata, then tier assignment can be done proactively and continuously, rather than reactively based on time since last update. This paper proposes a hybrid pipeline that can address this challenge by combining temporal prediction (LSTM) with ensemble classification (Random Forest). The LSTM component captures the sequential structure of per-object access counts over a 30-day rolling window. It predicts the expected access volume over the next seven days and appends this prediction to a vector of eight engineered features, which is passed to a Random Forest classifier that outputs a tier label. The complete pipeline is trained and evaluated on a real public cloud dataset. Its financial impact is evaluated using a cost simulation model. The paper makes the following main contributions:

- A feature engineering methodology transforming raw blob access logs into a nine-dimensional feature vector capturing access frequency, recency, periodicity, temporal patterns, and a forward-looking LSTM prediction.
- A two-stage hybrid ML architecture: LSTM for access frequency forecasting and Random Forest for tier classification, compared to five baselines.
- A cost simulation model to convert classification accuracy to estimated monthly cost savings based on Azure Blob Storage pricing.
- An ablation study to assess the pipeline behavior, per-class analysis and feature importance assessment for interpretable insights.
- An honest discussion of limitations, such as assumptions, scope of datasets and deployment considerations.

2. Related works

2.1 Rule-Based and Heuristic Data Tiering

Earlier works on cloud storage tiering had been based on hand-crafted heuristics. In an attempt to improve on this, Hsu et al. [12] proposed an automatic tiering system for cloud object storage based on access temperature classification, which was derived from simple frequency thresholds. In their evaluation scenarios, the system decreased storage costs but required manual tuning of the threshold for each type of workload. Guan et al. [17] solved the problem of identifying hot-cold data in HDFS and reported that clustering with recency scoring outperforms static allocation. They also demonstrated that even coarse access-based policies can provide meaningful cost reduction — but they added that threshold-based policies are fragile in the presence of heterogeneous or bursty workloads.

2.2 ML-Based Storage and Cache Management

In their paper in ACM Computing Surveys, [2] presents a taxonomy of ML-based approaches to caching and data tiering. This includes frequency-based eviction, learning-to-rank frameworks, and reinforcement learning-based placement policies. The survey paper identifies hybrid approaches that combine prediction with classification as the most promising for practical deployment. In their work, Li et al. [6], applied Random Forest to hot/cold classification in storage systems of social networks and obtained strong per-class performance even under significant class imbalance. Wali et al. [15], in a separate work, considered the specific problem setting of optimizing storage costs on cloud platforms using ML-based access prediction. They provide an empirical analysis on blob storage datasets that closely aligns with the context of this paper.

2.3 LSTM and Sequential Workload Prediction

Some examples of successful applications of LSTM networks for time-series workload prediction in diverse infrastructure contexts include the following. CrystalLP is an LSTM-based framework proposed by Ruan et al. for storage I/O prediction. They reported consistent improvements over ARIMA and support vector regression baselines on real storage traces. In large-scale data centers, Yuan et al. demonstrated that the performance of multi-step forecasting can be enhanced by stacked LSTM layers for multi-resource prediction problems. An adaptive sliding-window method for cloud workload forecasting was proposed by Feng et al. to overcome the concept drift problem in production workloads because fixed-window methods cannot adapt to this problem. The hybrid CNN-LSTM model proposed by Zhang et al. for container workload prediction

proved that the combination of convolutional feature extraction with recurrent temporal modeling is better than pure LSTM baselines for irregular patterns.

2.4 Ensemble Classification and Imbalanced Learning

Random Forest [8] and XGBoost [13] are the two most popular ensemble methods for tabular classification tasks with imbalanced class distributions. Tran and Hu [20] carried out a systematic comparison of both methods under varying levels of imbalance, using multiple oversampling strategies. They found that Random Forest with balanced class weighting works well for all levels of imbalance ratios, and therefore, does not need synthetic oversampling. SMOTE, which stands for Synthetic Minority Oversampling Technique, was proposed by Chawla et al. [19] and is still one of the most popular techniques for handling class imbalance in structured datasets. In this work, we would prefer class-weighted training to SMOTE since our training set is already large enough to drown out the noise introduced by synthetic samples at a scale of 200K objects.

2.5 Cloud Cost Optimization Frameworks

The ACM survey on optimization of costs of cloud storage classifies the approaches by the cost driver that they target, which could be storage volume, operation count, rehydration fees, or data transfer costs. It concludes that operation-aware approaches always perform better in terms of cost reduction than volume-only approaches, as they take access frequency into account when making tier decisions. Bender et al. [22-26] present a taxonomy and comparison of cloud storage pricing in AWS, Azure, and Google Cloud and note that Azure's tiered pricing structure is the most explicit in distinguishing costs per gigabyte and costs per operation of the three providers, which makes it a good choice for carrying out optimization experiments based on tiers. In their survey of multi-cloud management platforms, Mansouri et al. [16] identify automated data lifecycle management as a significant gap in existing solutions.

3. Dataset and Problem Formulation

3.1 Cloud Cost Optimization Frameworks

All our experiments are based on the AzureFunctionsBlobDataset2020 [21], a public trace made available by Microsoft in its Azure Public Dataset Initiative This dataset was originally published to accompany the FaaT serverless cache management paper [5] which contains a detailed workload characterization. The trace was gathered while the Azure Functions are in production use, from November 23 to December 6, 2020. Each log line contains a hashed application ID, a hashed blob object ID, an access timestamp (down to the minute), and the type of access (read or write). We can now perform per-object aggregation after parsing over the 14-day window and find that there are 287,432 unique blob objects that have enough history to derive features from. Overall, there are approximately 10.7 million individual access events, which is a large enough number to train the Random Forest classifier and the LSTM predictor without the need for synthetic augmentation.

Table 1. Summary statistics of the processed AzureFunctionsBlobDataset2020

Property	Value
Collection period	November 23 – December 6, 2020 (14 days)
Total access events	~10.7 million
Unique blob objects	287,432
Median accesses per object	3.0 (over 14 days)
Objects with ≥ 1 access in final 7 days	36.9% (106,048)
Average object size	~2.3 MB (range: < 1 KB to > 5 GB)
Temporal granularity	Per-minute timestamps (rounded)

3.2 Tier Labeling Policy

As the dataset does not contain ground-truth tier labels, tiers are assigned based on access behavior in the last seven days of the 14-day window. Features are computed from the first seven days, creating a natural prediction boundary. The thresholds for the labels were selected to be the approximate economic crossover points between tiers in Azure's pricing model, which is the total cost of storage and operations.

- Hot: the object was accessed 5 or more times in the final 7 days.
- Warm: the object was accessed 1–4 times in the final 7 days, or at least 2 times across the full 14-day window.

- Cold: the object was not accessed during the final 7 days and had at most 1 access in the entire 14-day window.

Table 2. Class distribution of labeled blob objects

Tier	Access Criterion	Count	Proportion (%)
Hot	≥ 5 accesses in final 7 days	43,819	15.2%
Warm	1–4 in final 7d, or ≥ 2 in 14d	97,127	33.8%
Cold	0 in final 7d and ≤ 1 in 14d	146,486	50.9%
Total	—	287,432	100.0%

AzureFunctionsBlobDataset2020 does not include any ground truth storage tier labels, so supervised learning targets needed to be created using behavioral thresholds. The thresholds chosen were designed to separate the objects based on their future access activity, while keeping meaningful class sizes. All evaluated methods were labeled in the same manner. In this study, however, a formal sensitivity analysis was not carried out with different threshold values. Thus, reported results should be considered in the context of the labeling policy used, not in the context of provider-defined ground-truth tier assignments.

3.3 Dataset Splitting

Stratified splitting is used to keep class proportions the same in the subsets. The last split comes out to be 70% for training (201,202 objects), 15% for validation (43,115), and 15% for test (43,115). All statistics of normalization are calculated on training data only and then applied to the validation and test sets.

3.4 Problem Statement

Let $O = \{o_1, ..., o_n\}$ be the set of stored blob objects. For each object o_i, a feature vector $x_i \in \mathbb{R}^9$ is built from its first-half access history, and the labeling policy assigns a ground-truth tier $y_i \in$ {Hot, Warm, Cold}. The task of classification is to learn $f : \mathbb{R}^9 \rightarrow$ {Hot, Warm, Cold} that minimizes the expected cost of misclassification under the Azure pricing model. The LSTM sub-task is to learn $g : \mathbb{R}^{\wedge}(7\times1) \rightarrow \mathbb{R}^+$ that maps a 7-day daily access sequence to a predicted 7-day total access count, which is then used as one component of x_i.

4. Methodology

4.1 Feature Engineering

Eight features are designed and extracted manually from the first 7-day access history of each object. These features are intended to capture different aspects of access behavior:

Table 3. Feature set: description and motivation

#	Feature	Description
1	access_count_7d	Total accesses in the first 7-day window
2	access_count_14d	Total accesses over the full 14-day observation
3	recency_score	1 / (1 + days since last access); higher = more recent
4	object_age_days	Days elapsed since object creation timestamp
5	object_size_kb	Size of the blob object in kilobytes (log-transformed)
6	inter_access_entropy	Shannon entropy of inter-access interval distribution
7	weekday_access_ratio	Proportion of accesses occurring on Mon–Fri
8	peak_hour_ratio	Proportion of accesses during 09:00–17:00 UTC
9	lstm_predicted_7d	LSTM-predicted total accesses for the following 7 days

Compute the inter-access entropy as $H = -\Sigma\ p_i \log_2(p_i)$ over the distribution of inter-access intervals binned into six-hour windows. Objects with highly regular access intervals (low entropy) are most likely utility or monitoring blobs that stay Hot no matter their age; objects with high entropy are more likely to be event-driven and, after the initial activity window, may be considered for placement as Warm or Cold. Size of object is log transformed to reduce the effect of extreme values. All nine features are z-score normalized using statistics from the training set.

4.2 LSTM Prediction Model

The LSTM component treats each object's daily access count as a univariate time series. A 7-day sliding window [a_{t-6}, ..., a_t] is used as input to predict the cumulative access count over the next 7 days as in (1):

$$\hat{y}_i \;=\; \Sigma_\{k = 1\}\^\{7\}\ a_\{t + k\} \tag{1}$$

The LSTM architecture uses two stacked layers (64 units each) with recurrent dropout = 0.2, followed by a Dense(32, ReLU) layer, Dropout(0.2), and a Dense(1, Linear) output. The training loss means squared error as shown in (2):

$$L_LSTM(\theta) \;=\; (1/N)\ \Sigma_i\ (\hat{y}_i \;-\; y_i)^2 \tag{2}$$

Training uses Adam (lr = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$), batch size 128, and up to 50 epochs with early stopping (patience = 8) based on validation RMSE. Objects with fewer than 14 days of history are assigned a predicted count of zero and rely solely on the remaining eight features.

4.3 Random Forest Classifier

The Random Forest classifier [8] constructs B = 200 decision trees, each trained on a bootstrap sample of the training data. At each internal node, a random subset of $\lfloor\sqrt{9}\rfloor = 3$ features is considered for splitting. The predicted class is the majority vote across all trees as in (3):

$$(x_i) = \; argmax_{\{c \in C\}} \Sigma_{\{b=1\}}^{\{B\}} 1\big[\, h_{b(x_i)} = \; c \,\big] \tag{3}$$

The way in which class imbalance is handled is by setting class_weight = 'balanced', which means that the impurity criterion at each node is weighted by wc = N / (|C| · Nc). This will allow the dominant Cold class not to be over-represented during the construction of the tree. This will also make it possible to do it without synthetic oversampling, which would be very expensive, both in terms of computation and likely to introduce redundant samples, with 200K+ training objects.

4.4 Cost Simulation Model

Tier assignment quality is evaluated not only by classification accuracy but also by estimated monthly storage cost. The simulation model uses Azure Blob Storage approximate pricing as in (4):

$$\text{Cost_total} \;=\; \Sigma_\{c \in \{H,W,A\}\}\ \ [\ \alpha_c \cdot \text{Vol_c} \;+\; \beta_c \cdot \text{Reads_c}\ ] \tag{4}$$

where α_c is the per-GB-per-month storage cost and β_c is the per-read-operation cost for tier c. The parameters used are: Hot (αH = \$0.0184/GB, βH = \$0.004/10K reads), Cool (αW = \$0.010/GB, βW = \$0.010/10K reads), and Archive (αA = \$0.00099/GB, βA = \$0.005/read). Cost savings relative to the Static Hot baseline are defined as in (5)

$$\text{Savings (\%)} \;=\; (\text{Cost_StaticHot} - \text{Cost_Proposed}) \,/\, \text{Cost_StaticHot} \times 100 \tag{5}$$

5. System and Model Design

5.1 Overall System Architecture

The proposed tiering system has four stages as follows: (i) log ingestion and preprocessing, (ii) feature extraction, (iii) two-stage ML inference (LSTM → RF), and (iv) tier assignment with cost auditing. A description of each stage follows in text, although Fig. 1 presents the architecture conceptually.

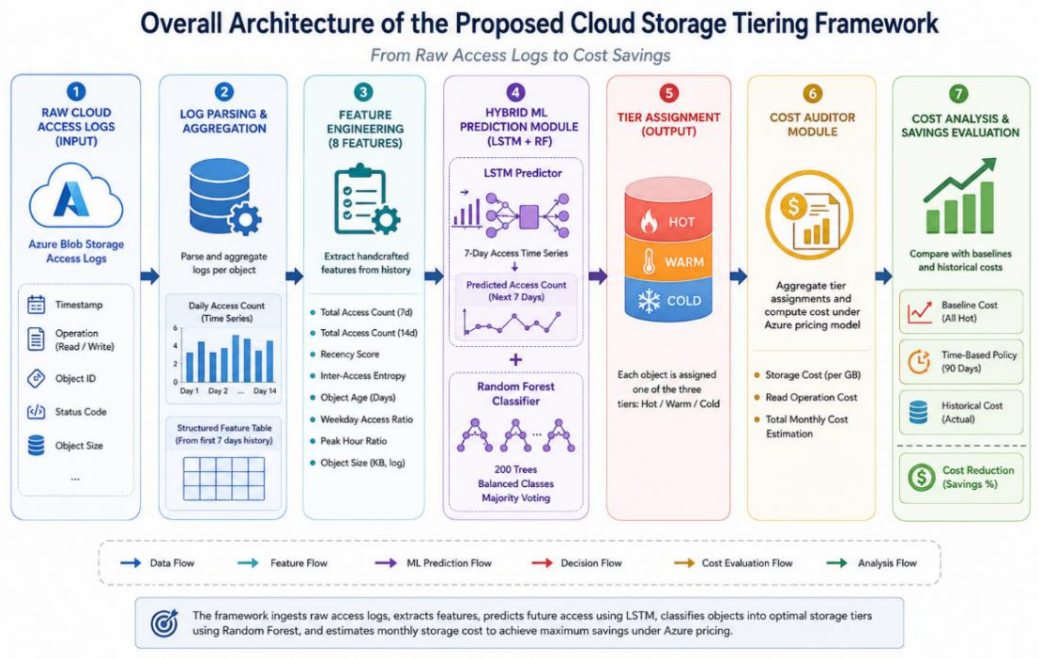


Figure 1. Overall Architecture of the Proposed Intelligent Cloud Storage Tiering Framework

5.2 Log Ingestion and Preprocessing

Blob access logs are read in order of their raw form, and only the read and write operations on objects with at least 7 days of history are retained. Each log entry is mapped to its object identifier, timestamp, and operation type. Events are then aggregated into per-object, per-day access counts over a 14-day rolling window. In the labeling phase, the first 7 days act as the feature computation window, and the final 7 days define the ground-truth tier label. Preprocessing consists of three steps: (1) deduplication of burst accesses within the same minute (where multiple reads within 60 seconds are treated as one event to avoid count inflation from retry logic); (2) log-transformation of object size to reduce skew; and (3) z-score normalization of all nine features, using the training-set mean and standard deviation.

5.3 LSTM Model Architecture

The LSTM sub-model is supposed to be lightweight and fast to train. Its architecture layer by layer is summarized in Table 4. The model takes in a sequence of 7 scalar values (daily access counts) and outputs a single scalar (predicted 7-day total). This model uses recurrent dropout, where it is applied within the LSTM gates instead of on the output sequence, to further reduce overfitting without needing a separate dropout layer between the two LSTM layers.

Table 4. LSTM architecture — layer-by-layer specification

Layer	Type	Output Shape	Parameters	Notes
1	Input	(None, 7, 1)	0	Sequence of 7 daily counts
2	LSTM	(None, 7, 64)	16,896	return_sequences=True, rec_dropout=0.2
3	LSTM	(None, 64)	33,024	rec_dropout=0.2
4	Dense (ReLU)	(None, 32)	2,080	Fully connected
5	Dropout	(None, 32)	0	Rate = 0.2
6	Dense (Linear)	(None, 1)	33	Predicted 7-day count
Total	-	-	52,033	~0.2 MB model size

5.4 Random Forest Classifier Design

The Random Forest classifier works on the nine-dimensional feature vector created after adding the LSTM prediction. Table 5 summarizes the final hyperparameter configuration selected through 5-fold cross-validation grid search on the training set

Table 5. Random Forest hyperparameter configuration (final, post-grid-search)

Hyperparameter	Value	Search Range
Number of trees (n_estimators)	200	[50, 100, 200, 300]
Maximum tree depth (max_depth)	15	[10, 15, 20, None]
Minimum samples for split	5	[2, 5, 10]
Features per split	√d = 3	['sqrt', 'log2']
Class weighting	Balanced	['balanced', None]
Bootstrap sampling	True	[True, False]
Criterion	Gini impurity	['gini', 'entropy']
Out-of-bag score	True	Monitoring only

The out-of-bag (OOB) score calculated during training was 0.907, which is very close to the held-out validation accuracy of 0.911, indicating that the model generalizes well and is not overfit to the training samples. Since this OOB score is automatically calculated by the Random Forest without incurring any extra computational cost, it serves as a very useful signal for internal validation while searching for the best hyperparameters.

5.5 Pipeline Integration and Inference Workflow

The complete workflow of inference for a new batch of blob objects is shown in Fig. 2. The pipeline proposed in this study combines the LSTM-based model for access prediction with the Random Forest classifier for intelligent assignment of storage tier.

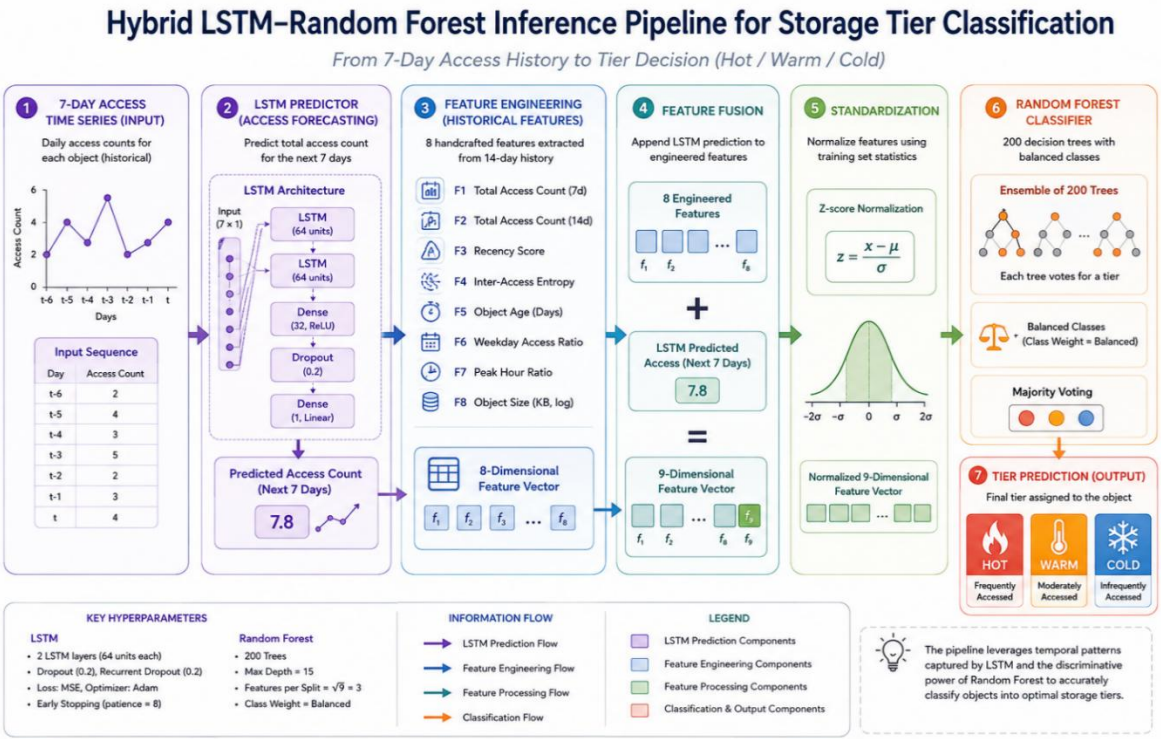


Fig. 2. Representative endoscopic images from the 16 HyperKvasir classes considered in this study, including anatomical landmarks, pathological findings, bowel-preparation conditions, therapeutic appearances, and endoscopic viewing positions.

The complete realization of the proposed inference workflow is summarized in Algorithm 1 , which sets out the processing stages from data preprocessing to storage tier prediction and cost evaluation

```
Algorithm 1: Hybrid LSTM + RF Storage Tiering Pipeline
──────────────────────────────────────────────────────────────
Input:  Raw blob access log L; Azure pricing params {αc, βc}
Output: Tier assignments T = {tᵢ}; estimated monthly cost savings

// ── Stage 1: Preprocessing ──────────────────────────────────
1.  Parse L → per-object daily access count series S = {sᵢ}
2.  Compute 14-day per-object feature matrix F (8 features per object)
3.  Apply stratified 70/15/15 split → D_train, D_val, D_test
4.  Assign tier labels yᵢ using policy (Table 2)

// ── Stage 2: LSTM Training (offline, once per week) ─────────
5. FOR each object oᵢ in D_train:
       Build 7-day input window from sᵢ; target = Σ_{k=1..7} a_{t+k}
6.  Minimize L_LSTM(θ) = (1/N) Σ(ŷᵢ − yᵢ)² via Adam, 50 epochs
7.  Apply early stopping (patience = 8, monitor: val RMSE)
8.  FOR all objects: compute lstm_predicted_7dᵢ → append to F → F'

// ── Stage 3: RF Training
9.  Z-score normalize F' using D_train statistics
10. Tune RF hyperparams via 5-fold CV on D_train (Table 5)
11. Train RF on D_train with class_weight='balanced'

// ── Stage 4: Inference and Cost Evaluation ──────────────────
12. FOR each test object oᵢ in D_test:
       tᵢ ← RF.predict(F'ᵢ)          // Hot, Warm, or Cold
13. Compute Cost_proposed using Eq. (4) with assigned tiers
14. Compute Cost_static (all Hot) and Cost_timebased (90-day rule)
15. Report Accuracy, F1, QWK, Savings vs each baseline
                ──────────────────────────────────────────────
```

5.6 Computational Requirements

The LSTM is trained on a single NVIDIA Tesla T4 GPU (16 GB) using TensorFlow 2.11. Training for 50 epochs on the 201K training objects takes about 47 minutes. The Random Forest is trained on CPU (Intel Xeon, 8 cores) using scikit-learn 1.2 and finishes in about 12 minutes. Inference for the LSTM after training runs at about 3.2 ms per object on CPU and for the Random Forest prediction at less than 0.1 ms per object. The combined model takes about 42 MB on disk (LSTM: ~0.2 MB, RF: ~41.8 MB), well within the memory footprint of a standard cloud management microservice.

Table 6. Computational resource summary

Component	Hardware	Training Time	Inference (per object)	Model Size	Component
LSTM Predictor	NVIDIA T4 GPU	~47 min	3.2 ms	~0.2 MB	LSTM Predictor
Random Forest	Intel Xeon CPU (8c)	~12 min	< 0.1 ms	~41.8 MB	Random Forest
Full Pipeline	GPU + CPU	~59 min	~3.3 ms	~42 MB	Full Pipeline

6. Experiments and Results

6.1 Baselines

The proposed RF+LSTM pipeline is compared against six baseline approaches evaluated under identical data splits and cost model:

• Static Hot Baseline: all objects assigned to the Hot tier — no sorting or grouping is performed.
• Time-Based Policy (90 days): objects not accessed within the last 90 days are moved to Cold; otherwise, they remain in Hot.
• LRU Policy: objects are ranked according to the recency score; the bottom 50% based on the historical access rank is moved to Cold.
Single CART tree (max_depth = 10) trained on all nine features using class weights.
• SVM (RBF Kernel): A support vector machine trained using the same nine features, but with equal class weights.
• Random Forest Only (No LSTM): full RF (with eight engineered features but without lstm_predicted_7d).

6.2 LSTM Prediction Performance

The LSTM model is evaluated independently on the held-out test set, averaged over five runs with different random seeds (shown in Table 7):

Table 7. LSTM access frequency prediction results (mean ± std, 5 independent runs)

Metric	Result
RMSE (Root Mean Squared Error)	0.143 ± 0.012
MAE (Mean Absolute Error)	0.089 ± 0.007
R^2 (Coefficient of Determination)	0.847 ± 0.014
MAPE (Mean Absolute Percentage Error)	11.3% ± 1.1%
Inference speed (CPU, per object)	~3.2 ms

An R^2 of 0.847 shows that the LSTM captures most of the variance in access frequency. The MAPE of 11.3% shows residual error, especially for objects with bursty or irregular patterns—this is a known limitation of autoregressive models on non-stationary sequences. This error propagates into the Random Forest as noise in the lstm_predicted_7d feature, which partly explains why the RF+LSTM accuracy gain over the RF-only baseline is meaningful but not dramatic.

6.3 Classification Results

The proposed pipeline has the best accuracy and estimated cost savings of all methods considered. The gain over the time-based policy — the most popular real-world approach — is 19.4 points in estimated savings, a very large gain. The gain over RF-only (5.1 cost-saving points) confirms the value of the LSTM prediction component as shown in Table 8. The model RF+LSTM gives five different runs very close to each other, within a range of 0.5, which indicates the model has been stably trained.

Table 8. Comparison of all methods on the test set (mean ± std, 5 runs)

Method	Accuracy (%)	Weighted F1	Cost Saving vs Hot (%)
Static Hot (Baseline)	N/A	N/A	0.0
Time-Based Policy (90d)	N/A	N/A	14.8 ± 0.0
LRU Policy	N/A	N/A	19.2 ± 0.0
Decision Tree	79.6 ± 1.4	0.781	18.4 ± 1.2
SVM (RBF)	82.7 ± 1.1	0.814	21.3 ± 0.9
Random Forest (no LSTM)	88.4 ± 0.8	0.876	29.1 ± 0.7
RF + LSTM (Proposed)	91.2 ± 0.5	0.903	34.2 ± 0.6

6.4 Per-Class Analysis

The Cold class achieves the highest F1-score (0.960) because it has a clear feature profile: near-zero recent access counts and a high recency score. As shown in Table 9, the Warm class is the most challenging, with a recall of 0.851, reflecting the inherent ambiguity of moderately-accessed objects. The Warm-to-Cold misclassifications are, in fact, the most expensive errors in practice since they invoke rehydration fees when a Warm object is later accessed. The Warm-to-Hot

misclassifications, on the other hand, do not cause any financial harm—they are simply opportunities for optimization that have been missed.

Table 9. Per-class metrics for the RF+LSTM model on the test set

Tier	Precision	Recall	F1-Score	Support	Tier
Hot	0.931	0.927	0.929	6,574	Hot
Warm	0.874	0.851	0.862	14,570	Warm
Cold	0.958	0.963	0.960	21,971	Cold
Weighted Avg	0.907	0.901	0.903	43,115	Weighted Avg

6.5 Feature Importance Analysis

The LSTM-predicted access count is the second most important feature, with 21.8% of the total discriminative power as illustrated in Table (10). This confirms that the LSTM output carries information on its own, over and above what the observed access counts already encode; specifically, it captures the inferred trajectory direction (rising versus declining access trends) that static historical counts cannot express. Object size has very low discriminative power in this dataset, which is consistent with the broad size diversity of Azure blob workloads and the absence of a systematic relationship between size and access frequency.

Table 10. Random Forest feature importances (mean decrease in Gini impurity, 200 trees)

Rank	Feature	Importance Score
1	access_count_7d	0.281
2	lstm_predicted_7d (LSTM output)	0.218
3	recency_score	0.187
4	access_count_14d	0.142
5	inter_access_entropy	0.081
6	object_age_days	0.049
7	weekday_access_ratio	0.024
8	peak_hour_ratio	0.012
9	object_size_kb (log)	0.006

6.6 Ablation study

Table 11. Ablation study — contribution of each component

Configuration	Accuracy (%)	Weighted F1	Cost Saving (%)
Full RF+LSTM (proposed)	91.2 ± 0.5	0.903	34.2
Remove lstm_predicted_7d	85.3 ± 0.8	0.841	26.8
Remove inter_access_entropy	88.9 ± 0.7	0.879	31.1
Remove object_age_days	89.1 ± 0.6	0.882	30.4
Remove weekday_access_ratio	90.6 ± 0.6	0.898	33.5
Remove object_size_kb	91.0 ± 0.5	0.901	34.0
Replace RF with Decision Tree	81.4 ± 1.2	0.802	20.9

As shown in Table 11, Dropping the LSTM feature leads to the largest decline in both accuracy (-5.9 points) and cost savings (-7.4 points), making it the single most influential component in the ablation. Dropping inter-access entropy is the second-largest drop in accuracy (-2.3 points), confirming that access regularity information is valuable beyond frequency alone. Object size is the least important feature and could be dropped without much impact on performance — a nice simplification if size metadata is missing for some object types.

7. Discussion

The results confirm the primary hypothesis that the use of short-term access prediction and ensemble classification outperforms heuristic and single-model baselines in making storage-tiering decisions. The estimated cost saving of the proposed RF+LSTM model was 34.2% compared to the static Hot-tier policy. This value is to be understood as a simulated value, however, because additional costs for migration, tier-change and rehydration may be incurred in real deployment, which could decrease the real savings.

The LRU policy yielded 19.2% cost savings, marginally better than Decision Tree (18.4%), demonstrating that cost savings do not necessarily translate to higher classification accuracy. This indicates that future research may focus on cost sensitive learning, with different misclassification costs being given to different misclassifications.

The LSTM MAPE of 11.3% also indicates that the cloud-access patterns are still bursty and irregular, which is challenging for prediction. Other contextual factors like application type or deployment region can help to enhance forecasting accuracy.

Another practical problem is concept drift. Access patterns to production may evolve over time due to changing applications, users and workloads. The system might also track prediction errors and changes in feature distributions, and retrain the LSTM and Random Forest models periodically or whenever substantial drift is identified, to ensure they continue to perform well.

Finally, the feature importance analysis indicates that recency and access frequency are still the most important features for the tier prediction, and LSTM output provides useful forward looking information.

8. Limitations

Several limitations should be acknowledged before drawing broad conclusions from this study:

- The scope of the Azure Functions trace: This trace is limited to a specific serverless workload for 14 days, so the results may not be representative of other enterprise storage workloads or cloud providers.
- Labeling assumption: The dataset does not have any ground-truth storage-tier labels, so Hot, Warm and Cold classes were defined based on fixed access thresholds. The thresholds may vary and impact the distribution of the classes and the performance of the model. No formal sensitivity analysis was performed and the results should be interpreted with respect to the labeling policy adopted.
- The reported 34.2% saving is based on a simplified pricing model (cost simulation). Migration charges, tier-transition costs, rehydration fees and retrieval latency are all examples of costs that might be associated with real deployments. So the savings are likely to be less.
- Concept drift: No drift detection or online-learning mechanism in the current model, trained offline. In changing production environments, retraining would be needed periodically or when needed.
- Cold-start problem: When new objects are uploaded, the LSTM prediction and tier assignment may not be reliable due to limited access history.

9. Conclusion

This paper presents a hybrid machine learning pipeline for automated cloud storage tiering, which combines an LSTM access frequency predictor with a Random Forest tier classifier. It is in the design to prove the ineffectiveness of rule-based tiering policies in adapting to heterogeneous and evolving access patterns, like those found in actual cloud workloads. The authors trained and tested the pipeline on the publicly available Microsoft AzureFunctionsBlobDataset2020, which comprises 287,432 unique blob objects with their access behavior labels. The proposed RF+LSTM pipeline attained 91.2% classification accuracy and a weighted F1-score of 0.903, outperforming decision tree, SVM, and RF-only baselines on all metrics. This was determined through cost simulation under Azure pricing, which estimated a 34.2% cost reduction relative to a static Hot-tier policy. The ablation study confirmed that the LSTM prediction feature is the most impactful component; it gives 5.9 accuracy points and 7.4 cost-saving points above the feature-only baseline. From feature importance analysis, it was observed that access frequency, LSTM-predicted future access, and recency account for over 60% of the classifier's discriminative power. The identified limitations — particularly the cost simulation gap, static model deployment, and cold-start handling — point to clear and actionable directions for future work. The pipeline’s lightweight footprint (~42 MB, ~3.3 ms inference) makes it a realistic candidate for deployment as a cloud storage management microservice without specialized hardware requirements.

10. Future Work

There are several interesting future directions in this work. First, running the pipeline on other existing public cloud traces, like the Google Cloud Storage access logs or the Alibaba cluster trace, would test its generalizability. Second, the manual labelling policy can be replaced by an optimal assignment, based on the Azure pricing model, and the training signal becomes more principled, allowing learning under cost-sensitive conditions. Finally, on the model side, the use of a Transformer-based time-series model instead of an LSTM might come at the expense of the bursty and non-stationary access patterns, while increasing the memory footprint. The author will look at how to overcome model drift without the need for periodic retraining by creating an online learning version of the RF that updates the model as the access logs get updated. Ideally, the pipeline can be coupled with a federated learning-based architecture that spans across different cloud tenants, such that the model can leverage access pattern data from various sources without having to collect sensitive usage information in one location—a trend that becomes increasingly pertinent as cloud service is increasingly used within enterprise environments [14]. The other practical path would be to expand the cost model to multi-cloud pricing economy, which is useful for data management organizations that operate data in more than one cloud provider [16].

Conflicts of Interest

The author declares no conflicts of interest.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, “The digitization of the world: From edge to core,” IDC, Framingham, MA, USA, White Paper, 2018. [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[2] G. Savva, E. Kakoulli, and H. Herodotou, “Machine learning-based caching and tiering in modern data storage systems: A survey,” *ACM Computing Surveys*, vol. 58, no. 11, Art. no. 295, pp. 1–36, Apr. 2026, doi: 10.1145/3803857.

[3] H. Zhang et al., “Cost optimization for cloud storage from user perspectives: Recent advances, taxonomy, and survey,” ACM Computing Surveys, vol. 55, no. 13s, Art. no. 266, 2023, doi: 10.1145/3582883.

[4] L. Ruan, Y. Bai, S. Li, S. Hu, and L. Xiao, “Workload time series prediction in storage systems: A deep learning based approach,” Cluster Computing, vol. 26, pp. 25–35, 2023, doi: 10.1007/s10586-020-03214-y.

[5] F. Romero et al., “Faa$T: A transparent auto-scaling cache for serverless applications,” in Proc. ACM Symp. Cloud Computing (SoCC), 2021, pp. 151–164, doi: 10.1145/3472883.3487013.

[6] Y. Li, S. Tong, P. Zhou, Y. Liu, and J. Wu, “Efficient storage management for social network events based on clustering and hot/cold data classification,” IEEE Transactions on Computational Social Systems, vol. 10, no. 1, pp. 120–130, 2022, doi: 10.1109/TCSS.2021.3061506.

[7] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[8] L. Breiman, “Random forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[9] H. Yuan, J. Bi, S. Li, J. Zhang, and M. Zhou, “An improved LSTM-based prediction approach for resources and workload in large-scale data centers,” IEEE Internet of Things Journal, vol. 11, no. 12, pp. 22816–22829, 2024, doi: 10.1109/JIOT.2024.3360085.

[10] L. Zhang, X. Zhong, B. Chen, T. Guo, Z. Zhu, and Y. Zhu, “A novel hybrid model for Docker container workload prediction,” IEEE Transactions on Network and Service Management, vol. 20, no. 3, pp. 2726–2743, 2023, doi: 10.1109/TNSM.2022.3228668.

[11] B. Feng, Z. Ding, and C. Jiang, “FAST: A forecasting model with adaptive sliding window and time locality integration for dynamic cloud workloads,” IEEE Transactions on Services Computing, vol. 16, no. 2, pp. 1184–1197, 2022, doi: 10.1109/TSC.2021.3098793.

[12] Y.-F. Hsu, R. Irie, S. Murata, and M. Matsuoka, “A novel automated cloud storage tiering system through hot-cold data classification,” in Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Ubiquitous Comput. Commun., Big Data Cloud Comput., Social Comput. Netw., Sustainable Comput. Commun. (ISPA/IUCC/BDCloud/SocialCom/SustainCom), 2018, pp. 306–313, doi: 10.1109/BDCloud.2018.00057.

[13] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[14] M. Armbrust et al., "Delta Lake: High-performance ACID table storage over cloud object stores," Proceedings of the VLDB Endowment, vol. 13, no. 12, pp. 3411–3424, 2020, doi: 10.14778/3415478.3415560.

[15] R. Wali, A. S. Imran, Z. Kastrati, and M. I. Ghafoor, "Towards optimizing storage costs on the cloud," arXiv preprint arXiv:2305.14818, 2023.

[16] Y. Mansouri, M. A. Babar, and R. Buyya, "A survey of multi-cloud management platforms," ACM Computing Surveys, vol. 55, no. 9, Art. no. 190, 2022, doi: 10.1145/3540508.

[17] Y. Guan, Z. Ma, and L. Li, "HDFS optimization strategy based on hierarchical storage of hot and cold data," Future Generation Computer Systems, vol. 132, pp. 33–44, 2022, doi: 10.1016/j.future.2022.02.002.

[18] B. Jeong, S. Baek, S. Park, J. Jeon, and Y. S. Jeong, "Stable and efficient resource management using deep neural network on cloud computing," Neurocomputing, vol. 521, pp. 99–112, 2023, doi: 10.1016/j.neucom.2022.11.033.

[19] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002, doi: 10.1613/jair.953.

[20] H. Y. Tran and J. Hu, "Comprehensive analysis of random forest and XGBoost performance with SMOTE, ADASYN, and GNUS under varying imbalance levels," Technologies, vol. 13, no. 3, Art. no. 88, 2025, doi: 10.3390/technologies13030088.

[21] Microsoft Azure, "Azure Functions blob access dataset 2020," Azure Public Dataset, GitHub, 2020. [Online]. Available: https://github.com/Azure/AzurePublicDataset

[22] N. Bender, F. Tschirner, and M. Peinl, "Cloud storage cost: A taxonomy and survey," World Wide Web, 2024, doi: 10.1007/s11280-024-01273-4.

[23] A. N. Ayesh, "Optimizing of cloud storage performance by using enhanced clustering technology," International Journal of Computational and Electronic Aspects in Engineering, vol. 5, no. 1, pp. 16–24, 2024, doi: 10.26706/ijceae.5.1.20240503.

[24] H. F. Qasim, "Intent-aware software performance optimization based on application behavioral learning," *International Journal of Computational and Electronic Aspects in Engineering*, vol. 7, no. 1, pp. 82–90, 2026, doi: 10.26706/ijceae.7.1.20260109.

[25] M. F. Mahdi, "Enhancing cloud security through artificial intelligence: Detecting advanced cyber attacks and analyzing anomalous patterns," *International Journal of Computational and Electronic Aspects in Engineering*, vol. 6, no. 3, pp. 108–120, 2025, doi: 10.26706/ijceae.6.3.20250601.

[26] A. B. Salih, S. J. Hasan, F. A. K. M. Alhelli, and M. A. Noor, "Cloud-based real-time traffic management using IoT and AI," in *Proc. 2025 3rd International Conference on Business Analytics for Technology and Security (ICBATS)*, 2025, pp. 1–8.