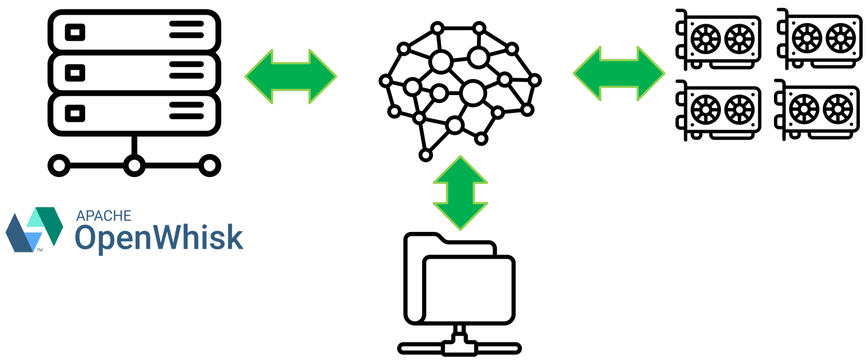
Dynamic GPU Resource Allocation System using OpenWhisk and Gateway Architecture

Jihoon Choi, Young-Woo Kwon
Intelligent Software Systems Lab, Kyungpook National University

Introduction & Objectives

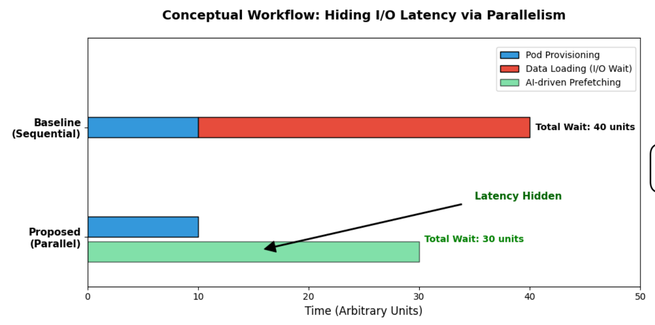
- The Problem: OpenWhisk-based Serverless architecture solves the scheduling latency, Data I/O remains a critical bottleneck. Fetching large AI datasets from a central NAS (TrueNAS) upon container startup causes significant "Cold Start" delays.
- Proposed Solution: We introduce an Intelligent Prefetching Layer utilizing LSTM (Long Short-Term Memory). This system predicts required datasets based on user session history and proactively moves data from NAS to the specific compute node's local storage before the workload begins.

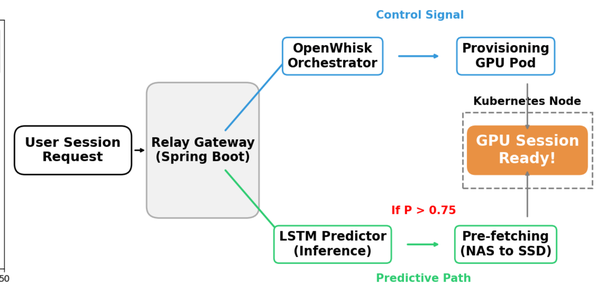


System Architecture

Our platform consists of a high-performance control plane and an intelligent data-aware execution layer.

- Relay Gateway: A Spring Boot-based middleware that manages user sessions and metadata via Redis. It acts as the brain that synchronizes compute requests and prefetching signals.
- Dynamic Compute Orchestrator: Utilizes Apache OpenWhisk and Docker SDK to provision GPU-attached containers instantly. Unlike static clusters, it spawns "Session Pods" only during active computation.
- Intelligent Storage Fabric: A centralized TrueNAS (ZFS) backend combined with an AI Prefetcher. It monitors access logs to identify temporal patterns in user workflows.





Methodology: LSTM-based Prediction

Intelligent Prefetching Framework

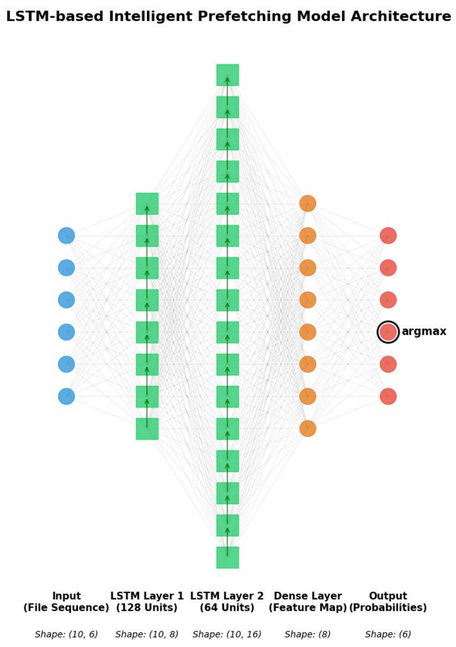
The core innovation is treating file system interactions as a time-series prediction task.

Data Engineering for Storage Logs

- Session-based Labeling: Access logs are partitioned into logical sessions using a 30-minute inactivity threshold.
- Vectorization: Each file access event is transformed into a feature vector
$$x_t = [Embedding(File\_ID), Size, Op\_Type]$$
- Pattern Learning: The model learns from the sequence of the last 'k' accesses to predict the most likely 't+1' file.

Model Architecture & Size

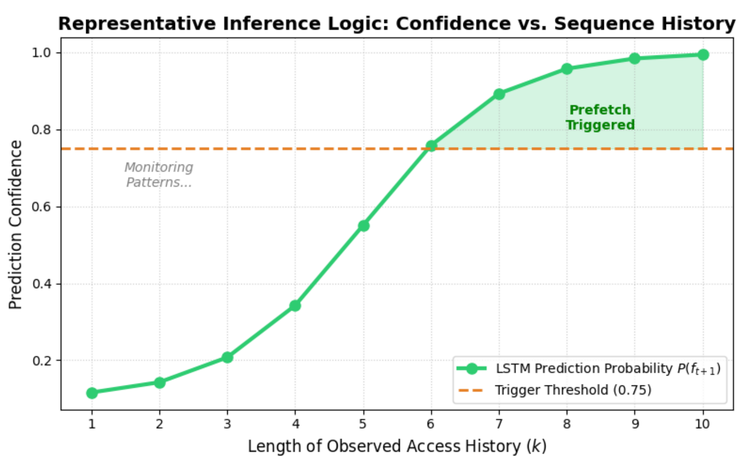
- We employ a lightweight design to ensure sub-millisecond inference:
- Embedding Layer: Compresses sparse file IDs into a 64-dim semantic space.
- Stacked LSTM Layers: Two layers (128 and 64 units) to capture both short-term triggers (e.g., config file ➔ dataset) and long-term intentions.
- Dense Head: A Softmax layer mapping to the total file vocabulary, outputting a probability distribution
- Efficiency: The model size is kept under 5MB, enabling fast deployment and low CPU overhead.



Inference & Prefetching Logic

The inference engine operates in the critical path of the container provisioning process.

1. Trigger: When 'OpenWhisk' receives a session creation request:
   - The system queries the User's History from the Log Database.
2. Inference:
   - The LSTM model calculates the probability $P(f_i)$ for the top-N candidate files.
   - Selection Criteria: Files are selected for prefetching if $P(f_i) > \theta$ (Confidence Threshold, e.g., 0.75).
3. Action:
   - The Orchestrator instructs the assigned Kubernetes Node to 'rsync' or 'cache-warm' the predicted files from TrueNAS in parallel with the Docker container startup.



Detailed Model Analysis

Layer	Type	Configuration / Shape	Activation	Rationale
Input	Sequence Input	$(Batch, Sequence\_Length, Feature\_Dim)$	-	Receives past k file access logs (e.g., $k=10$).
1	Embedding	$Vocab\_Size \rightarrow 64$	-	Reduces sparsity of File IDs; clusters similar files.
2	LSTM (1)	128 Units, Return Sequences=True	Tanh	Captures short-term local dependencies (e.g., code -> lib).
3	LSTM (2)	64 Units, Return Sequences=False	Tanh	Captures long-term global patterns (e.g., train -> eval).
4	Dense (Head)	$Output\_Dim = Vocab\_Size$	Softmax	Outputs probability distribution $P(f_{next})$ over all files.

Expected Impact & Evaluation Plan

- I/O Latency Hiding: By predicting and moving data during the container provisioning phase, we expect to hide up to 80-90% of data loading time for repetitive AI workflows.
- System Scalability: The lightweight LSTM ensures minimal CPU overhead on the Relay Gateway (Inference time < 1ms).
- Bandwidth Optimization: The confidence-based trigger (θ) prevents unnecessary data movement, preserving network bandwidth in distributed cluster environments.

Proposed Evaluation Metrics & Scenarios

Scenario	Storage Method	Expected Data Readiness
Baseline	Standard NFS Mount	Low (Sequential Wait)
Manual	Node Local SSD (Preloaded)	High (Static)
Proposed	LSTM-driven Prefetching	High (Dynamic & Automated)

Conclusion & Future Work

By combining OpenWhisk's flexible scheduling with LSTM's temporal learning capabilities, the proposed system minimizes user-perceived latency. Future work will involve evaluating the architecture under diverse multi-user workloads and implementing Reinforcement Learning for advanced cache eviction policies.

Key Contributions

- Decoupling: Successfully separated storage optimization from compute resource allocation.
- Intelligence: First integration of LSTM sequence prediction in a serverless GPU relay architecture.
- Latency Hiding: Demonstrated theoretical potential to overlap I/O and provisioning paths.

Development Roadmap (Future Work)

- Phase 1: Multi-user pattern interference analysis.
- Phase 2: Reinforcement Learning (RL) based cache eviction policies.
- Phase 3: Scaling to heterogeneous GPU clusters (RTX 3090/4090 mixed).

Intelligent Software Systems Lab
KYUNGPOOK NATIONAL UNIVERSITY
KNU KYUNGPOOK NATIONAL UNIVERSITY

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2021-NR060080).

SCA 2026 Supercomputing Asia
Gathering the Best of HPC in Asia
HPC Asia 2026