LatVision: Modeling and Predicting Persisting Tail Latency in SSDs

1st Linxiao Bai
National University of Defense Technology
Changsha, China
linxiao_b@nudt.edu.cn

2nd Zhijie Jiang
National University of Defense Technology
Changsha, China
jiangzhijie@nudt.edu.cn

3rd Yuanliang Zhang
National University of Defense Technology
Changsha, China
zhangyuanliang13@nudt.edu.cn

4th Haoran Liu
National University of Defense Technology
Changsha, China
liuhaoran14@nudt.edu.cn

5th Xiangbing Huang
National University of Defense Technology
Changsha, China
xbhuang@nudt.edu.cn

6th Wang Li
National University of Defense Technology
Changsha, China
liwang15@nudt.edu.cn

7th Bin Lin
Center for Strategic Evaluation Consultation
Academy of Military China
Beijing, China
bingo186@126.com

***Abstract*—As Solid State Drives (SSDs) continue to evolve, the presence of tail latency within these devices remains a significant issue that can adversely affect overall performance. Various factors contribute to the emergence of tail latency spikes in SSDs. Current software-level management solutions primarily focus on the performance prediction of individual I/O operations, recognizing that persistent slow operations are prevalent in SSDs and tend to have a more pronounced impact. In this paper, we build a tool-LatVision to obtain I/O-related data directly from the kernel to predict persisting tail latency in SSDs by a neural network model. We conduct a comprehensive comparison and analysis of the input metrics and predictive models employed. Furthermore, we enhance LatVision's performance through the application of heuristic algorithms. Through LatVision, we achieve real-time, lightweight, and high-accuracy performance prediction for low-latency SSDs.**

***Index Terms*—Tail Latency, Performance Prediction, Neural Network Model**

This research was funded by NSFC No.62272473, the Science and Technology Innovation Program of Hunan Province (No.2023RC1001) and NSFC No.62202474.

I. Introduction

Over the last decade, Solid State Drives (SSDs) [1] have marked a significant evolution, heralding a new era of speed and reliability over their mechanical counterparts. The adoption of SSDs across various sectors—from personal computing to enterprise-level data centers—has been driven by their ability to offer drastically reduced access times and enhanced durability [2]. However, the increasing performance expectations from SSDs also spotlight a critical challenge: tail latency [3]. Tail latency refers to the unusually long response times observed during accessing SSDs, which, despite representing a small fraction of overall operations, can drastically degrade the user experience and system efficiency [4], [5].

This issue of tail latency in SSDs is particularly troubling as it often occurs sporadically but with enough frequency to impact critical operations and system throughput significantly [6]. Such latency spikes can arise from a variety of factors, including garbage collection [7] processes within the drive, firmware bugs [8], or simply due to the drive reaching its

979-8-3315-4046-3/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCC64274.2024.00020

performance limits under heavy load conditions [9]. The unpredictability and severity of these events make them a crucial area of focus for system architects and performance engineers.

Currently, the management of tail latency employs two principal approaches. On the hardware front, modifications can be made to the flash controller [10], [11] to optimize the internal mechanisms of flash memory, thereby reducing the incidence of tail latency. On the software front, flexible handling of hardware-induced tail latency [12] can be implemented to mitigate its impact. Nonetheless, existing approaches often lack efficiency in handling persisting tail latency (i.e. continuous slow I/Os), which are more consequential than isolated incidents. Focusing on individual slow predictions is not meaningful, as the overhead associated with switching, even if predicted, is substantial. Continuous slow I/Os not only persist but are also correlated with numerous performance metrics, making them predictable.

Given these challenges, this paper introduces a novel software-level tool-LatVision designed for the prediction of persisting tail latency in SSDs. The tool's architecture encompasses three main components: (1) the use of extended Berkeley Packet Filter (eBPF [13]–[15]) for data collection directly from the kernel, (2) feeding this data into a predictive model, and (3) employing a heuristic algorithm to refine the predictions.

In our selection of a predictive model, we conduct an in-depth comparison of various established classification models. We evaluate the advantages and disadvantages of each model based on both accuracy and computational load. Ultimately, MobileNet is selected as the foundational framework for our predictive model due to its light overhead and high accuracy.

In terms of metric selection, our goal is to achieve the best predictive performance with the fewest possible parameters. We decide on the following three categories of metrics for model input: (1) Recent I/O information, (2) Remaining disk space, (3) Number of active related threads. Specifically, the 'Recent I/O information' includes the number of currently pending I/O operations, the latency of the most recently completed R I/O operations, and the number of pending I/O operations at the time each of the R completed I/Os arrived. Notably, the parameter R can be customized by the user.

In summary, our paper makes three key contributions:

- To our knowledge, LatVision is the first tool designed for performance prediction of low-latency devices at the user level. We believe our study will leads to exciting discussions and questions that can spur future work.
- We present a tool-LatVision characterized by low operational overhead and high accuracy. We publish our tool on Github: https://github.com/Anonymous3pp/LatVision.
- An in-depth comparison and evaluation of model and metric selection bases, demonstrating the effectiveness of our approach in enhancing the predictability of persisting tail latency. This work not only sheds light on the underlying dynamics of SSD performance but also paves the way to help mitigate the impact of tail latency on system operations in future.

II. Background and Related Work

A. SSD Architecture

Solid State Drives (SSDs) [1] have revolutionized the storage industry by offering significant improvements in speed, reliability, and energy efficiency compared to traditional hard disk drives (HDDs).

With the continual advancement of technologies such as NVMe (Non-Volatile Memory Express) [16] and 3D NAND [17], the read and write performance of Solid State Drives (SSDs) has undergone a significant enhancement by several orders of magnitude. For instance, consider the Intel X-25E SATA SSD, which exhibited a read latency of 75 milliseconds. In contrast, the latest SSDs like the Samsung 980Pro NVMe SSD boast a remarkable reduction in read latency, requiring a mere 5 microseconds for the same operation.

The ramifications of this technological evolution are profound, as modern SSDs not only outpace traditional mechanical hard drives in terms of read/write speeds but also exhibit significant advancements in responsiveness, energy efficiency, and reliability. Consequently, SSDs have emerged as the preferred storage solution for a diverse array of computing environments, ranging from personal computing devices to enterprise-grade servers and data centers.

B. Tail Latency

Tail Latency is a common phenomenon in disk systems [4], [6], [18]. In large-scale disk arrays, tail latency can potentially lead to a decrease in overall system performance [4]. In some complex situations, it may even cause system crashes [3], resulting in very serious consequences.

The fundamental reasons for tail latency in SSDs can be attributed to two main aspects. Firstly, external

factors significantly impact SSD performance. For instance, elevated environmental temperatures may cause SSDs to operate at lower speeds due to thermal constraints or trigger thermal management mechanisms, thereby affecting disk response times. Similarly, power fluctuations or instability [19] can also negatively impact SSD performance, leading to the occurrence of tail latency.

Another crucial factor contributing to the occurrence of tail latency in SSDs is the nature of flash memory mechanisms itself [2], [8], [20]. Flash storage devices employ mechanisms such as garbage collection and load balancing to maintain their internal states, ensuring efficient data management and reliability. However, these internal mechanisms may result in the execution time of certain operations exceeding expectations, thus causing tail latency. For example, when SSDs perform garbage collection operations, they may consume significant system resources, leading to increased response times for other operations and consequently triggering tail latency phenomena.

C. Tail Latency Management

In addressing and managing tail latency, there have been significant advancements in current research efforts, effectively addressing some issues and achieving favorable outcomes. On the hardware front, Huaicheng Li's IODA [10] approach, through proactive data reconstruction, enables predictable I/O operations, while IODA introduces busy-idle time exposure and predictable latency window formulas to ensure predictable data reconstruction, especially when dealing with concurrent internal operations. Additionally, the introduction of five new fields into the NVMe interface has greatly improved the 95-99.99th percentile latency.

Furthermore, Amy Tai [11] proposes a novel method for enhancing SSD performance by employing Calibrated Interrupts to service latency-sensitive requests. This approach significantly boosts system throughput while reducing latency by 37% under merged interrupts.

In the realm of software, Mingzhe Hao introduces LinnOS [12], which utilizes optical neural networks to infer SSD performance at a fine-grained per I/O level, aiding parallel storage applications in achieving performance predictability. Moreover, LinnOS integrates machine learning into the operating system for real-time decision-making, effectively reducing tail latency.

III. Motivation

A. Overhead Trade-off

Predicting I/O operations within disk systems presents a viable solution to enhance performance and alleviate latency concerns. However, the intrinsic low-latency attributes of Solid State Drives (SSDs) introduce the potential for unwarranted overhead during prediction and processing phases. Employing exceedingly lightweight models with minimal input may lead to compromised accuracy due to resource constraints, particularly evident in high-performance environments where even a single false positive could trigger erroneous processing, potentially yielding consequences more severe than tail latency. Conversely, pursuit of higher model accuracy through the utilization of more intricate models amplifies the computational overhead during the prediction phase, thereby exerting notable pressure on SSD read-write operations. LinnOS [12] adopts the former approach, leveraging lightweight models to predict the low-speed status of each I/O passing through SSDs, redirecting those identified as low-speed for processing onto redundant devices.

In addition to the aforementioned overhead considerations, the re-routing of slow I/O operations entails a significant temporal expense. Preliminary experimental evaluations indicate that a simple re-routing switch may incur a latency ranging from 3-10μs, whereas the latency per I/O operation typically hovers around 9-20μs, rendering the re-routing latency non-negligible. The design of LinnOS [12] detects and performs possible redirection for each I/O, which indeed reduces the overhead caused by slow I/O, but also introduces frequent redirection overhead. This is not significant for a single slow I/O, and may even have a counterproductive effect in some situations (such as false positives in judgment results).

B. Continuous I/Os vs Single I/O

However, some studies on SSD tail latency have noted a certain correlation in the temporal dimension of NVMe SSDs [4]. To investigate this, we conducted validation experiments related to continuous slow I/Os. We record the delay of each I/O in the actual workload, randomly select a time window of 1000 I/Os. And then we use a labeling algorithm (detailed in Section IV-B1) to label it as fast or slow. In our tool, continuous slow I/O is defined as k consecutive individual I/Os as slow I/O. Our preliminary analysis of the data indicate that continuous occurrences of slow I/Os ($k \geq 5$) accounted for 56% of all slow I/Os. Fig.1 shows the performance

Fig. 1. Slowdown when encountering single slow I/O and continuous slow I/Os. *Explained in Section III-B.*

slowdown of Filebench [21] and Fio [22] when encountering single slow I/O and continuous slow I/O under three different workloads. As shown in Fig.1, continuous slow I/Os has a more severe impact on upper-layer software. It makes sense to focus more attention on continuous slow I/Os.

Simultaneously, in our testing, we observed that continuous slow I/Os often exhibit distinct patterns and have a strong correlation with factors such as queue length, I/O size, and waiting time. Therefore, continuous slow I/Os can be predicted reasonably. This discovery inspired us to attempt a new approach: instead of observing each individual I/O, we expanded the observation window to a group of I/Os, such as n consecutive I/Os (where n=40). Based on the characteristic information of the previous n I/Os, we utilized a model to predict whether subsequent occurrences of y slow I/Os (where $y \geq 5$) would happen. If it is predicted that multiple slow I/Os will occur consecutively, it indicates that the current device is operating at a slow speed, rather than at its normal operating speed.

C. Unsuccessful Attempts and Challenges

As a tool for addressing tail latency issues at the software level, LinnOS achieves a high level of accuracy in predicting individual I/O operations. LinnOS embeds a neural network structure within the operating system kernel, receiving feature information for each I/O operation and using this data for prediction. We extended LinnOS by enhancing its ability to predict consecutive instances of slow I/O operations. However, due to latency constraints, we considered predicting three consecutive instances of slowness as indicative of forecasting a sequence of slow I/O. Under this condition, LinnOS's accuracy decreased to 49.7%, accompanied by an additional 2.3x overhead. Therefore, predicting within the operating system kernel poses significant challenges and is not the optimal solution.

Efficiently predicting whether a device is operating at a slow pace presents a formidable challenge in our quest to design a tool employing a more granular model for forecasting. In striving for both accuracy and minimal additional overhead in our predictive endeavors, we confront several key challenges:

Acquiring Relevant I/O Data: I/O characteristics serve as the foundation for prediction; however, the availability of I/O-related metrics in user space is notably limited. Moreover, these metrics often constitute macro-level indicators, rendering them insufficient for precise prediction. To preserve the accuracy of our model, we must employ non-intrusive methods to extract I/O information from kernel space, all while ensuring that our data collection methods do not impose unacceptable overhead on I/O operations.

Determining Appropriate Metrics: To predict the current operational state of a device accurately, a multitude of metrics is at our disposal. At a macroscopic level, system metrics such as CPU utilization, memory usage, and device state indicators like temperature and utilization rates serve as potential input parameters. Delving into kernel-level I/O granularity, factors such as current I/O size, queue length, and cumulative wait time are considered. Furthermore, over a temporal span, historical I/O information is pivotal in inferring whether an SSD, for instance, is currently experiencing internal workload. Put succinctly, if preceding I/O operations have encountered prolonged latency, the likelihood of the disk operating at a sluggish pace is considerably high.

Selecting an Appropriate Model Structure: The rapid evolution of neural networks and the increasing complexity of AI capabilities pose a challenge in selecting an apt model from a plethora of options for predictive tasks. Furthermore, the accuracy and cost of the model are still trade off. Although predictions may not target every individual I/O operation, real-time responsiveness remains paramount, necessitating constant awareness of the current device's operational status. Thus, the speed of model execution stands as a critical metric for evaluating the efficacy of our tool.

IV. Design

In this section, we describe our solution to the challenges mentioned above. This section presents the final design and the principal intuitions about how we get there. Section IV-A introduces the overall workflow of LatVision. And then we will explain the design of

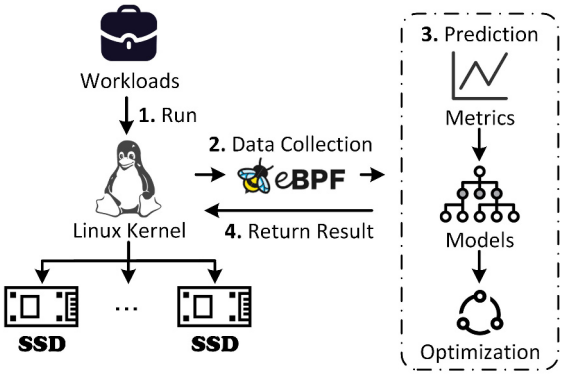

Fig. 2. Workflow of LatVision. *Explained in Section IV-A.*

LatVision from the following three parts: data collection (Section IV-B), metric selection (Section IV-C), model selection (Section IV-D) and optimization (Section IV-E).

A. Overview

The overall workflow of LatVision is illustrated in Fig.2. Our work consists of the following steps: **1.** Running the workloads on SSDs. **2.** Utilizing eBPF to collect relevant feature information. **3.** Using the collected data for prediction. **4.** Providing the predicted results to the operating system or users.

The prediction module is further divided into three key parts. Firstly, the selection of metrics, determining which metrics to use as inputs for the model. Secondly, the selection of the model, comparing the prediction accuracy of various classification models. Finally, proposing optimization algorithms for addressing false positives in the prediction model, further enhancing the model's predictive capabilities.

B. Data Collection

To achieve our goal of creating a non-intrusive predictive tool, we must contend with the limited information available at the user level. Not only do we require access to overarching device operational parameters, but also the historical data of every I/O operation over time is crucial for the efficacy of our tool. The key of this challenge lies in the real-time transmission of I/O-related information from the kernel space to the user space.

To address this issue, we employ eBPF (extended Berkeley Packet Filter) [13], [14] tools for data collection. eBPF serves as a hooking mechanism provided by the kernel. Leveraging kernel runtime eBPF tools enables the triggering of corresponding hook functions within the kernel upon the execution of I/O-related functions. These hook functions, pre-defined in nature, allow us to record pertinent I/O information and transmit it to the user space. In doing so, we acquire real-time data regarding I/O operations.

1) Labeling: Utilizing a supervised classification methodology necessitates training the model with labeled data. However, assigning actual microsecond-level latency labels to every I/O operation may inundate our dataset with excessive labels, particularly since minor delays such as 1μs may not significantly impact user experience. To mitigate this, alternative labeling strategies such as linear (e.g., 0-10μs, 10-20μs) or exponential (e.g., 2-4μs, 4-8μs) intervals have been explored. Although these labeling schemes offer a better fit, achieving high accuracy and speed in the model remains challenging even after multiple design iterations.

Taking into consideration the complexities outlined above and drawing insights from prior research on performance variance, it is observed that latency often adhere to a Pareto distribution characterized by a high alpha value. As depicted in Fig.3(a), approximately 90% of the time, latency exhibit stability, while the remaining 10% manifest as an elongated tail, indicative of sporadic delays. This Pareto distribution effectively delineates between swift and sluggish latency regions. Consequently, a plausible conjecture emerges suggesting that users are primarily concerned with the tail-end behavior of latency rather than precise microsecond values.

When determining the labeling point, we define it as the point below which latency indicates normal operation for an I/O operation, while latency exceeding this threshold signifies tail latency. We experimented with different slopes (ranging from 30° to 75° in increments of 5°) of lines tangent to the distribution curve to identify the optimal landmark position. Upon analysis, we found that the impact on predictive results was minimal when using lines with slopes between 40° and 60° tangent to the curve. Therefore, we chose the 50° line corresponding to the tangent point of the curve as the final landmark position as shown in Fig.3(b)(c).

It is worth noting that the shape of this distribution curve is influenced by various factors in the actual operating environment, including workload conditions. Hence, periodic retraining of the model is deemed necessary.

C. Metric Selection

In order to predict the occurrence of slow-running scenarios during device operation, the metrics inputted

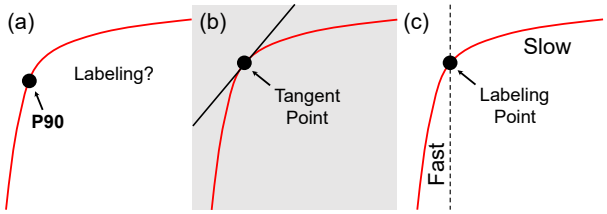

Fig. 3. Labeling point (fast/slow threshold). *As explained in Section IV-B1, the figures show the generation of labeling point. The figure format is latency CDF (Cumulative Distribution Function).*

into the predictive model are of paramount importance. Our model selects the following types of features for input:

The number of currently pending I/Os: The current count of pending I/O operations determines the level of congestion in reads and writes. If there are a significant number of I/O operations awaiting processing, it indicates that the device is likely experiencing a high level of activity, thereby increasing the probability of encountering tail latency.

The latency of the R most recently completed I/Os: The recent latency of the R most recent I/O operations also reflects the current operational state of the device. Here, we set the value of R to 20, which will also be subject to discussion in subsequent experimental tests.

The number of pending I/Os at the time when each of the R completed I/Os arrived: Predicting the internal state of the device from the number of pending I/Os at the time of completion of each recent I/O operation. In essence, if recent I/O operations experience prolonged latency without many pending I/Os, the model can infer internal contention due to device-level activities such as garbage collection, internal flushing, or wear leveling, potentially leading to tail latency.

Device space utilization rate: This macroscopic indicator aids in assessing whether the device may engage in activities such as garbage collection, providing significance to the prediction of device slow states. As macro metrics, data changes are not significant in the short term, so in order to save time, we collect data every 5 seconds.

System-level metrics: The number of active threads relevant to the current device. For an active thread that has previously sent I/O requests to the device, it is considered relevant to the current device because it is likely to continue sending I/O requests. The higher the number of active threads, the greater the potential for congestion.

D. Model Selection

Given that LatVision requires a binary classification task and rapid decision-making, we refrain from selecting overly complex models. We have experimented with the following mature classification models:1D Convolutional Neural Network (1D CNN) [23], Deep Neural Network (DNN) [24], Recurrent Neural Network (RNN) [25], Residual Network (ResNet) [26], EfficientNet [27], MobileNet [28]. These models each have their own advantages. After testing and comparing them, we choose the model that is more suitable for predicting SSD performance.

Given that LatVision requires a binary classification task and rapid decision-making, we refrain from selecting overly complex models. We have experimented with the following mature classification models:

1D Convolutional Neural Network (1D CNN) [23]: 1D CNNs excel in processing sequential data, making them suitable for feature extraction and classification of time-series data.

Deep Neural Network (DNN) [24]: DNNs are classical deep learning models suitable for various types of data, including images, text, and numerical data, with strong representation capabilities and flexibility.

Recurrent Neural Network (RNN) [25]: RNNs are a class of neural network models specialized in handling sequential data, capable of capturing temporal dependencies between data, applicable in areas such as natural language processing and time series prediction.

Residual Network (ResNet) [26]: ResNet is a type of deep residual network structure that addresses the problem of vanishing gradients in deep neural network training by introducing residual blocks, exhibiting good training stability and generalization performance.

EfficientNet [27]: EfficientNet is a type of neural network structure designed based on automated network scaling methods, capable of achieving high performance while maintaining relatively low model complexity.

MobileNet [28]: MobileNet is a lightweight convolutional neural network structure designed specifically for efficient image recognition and classification tasks on mobile devices with limited computational resources.

E. Optimization

Accurate inference entails the system correctly predicting when a device is operating swiftly and refraining from taking corresponding actions (true negatives), or predicting when a device is operating sluggishly and implementing mitigating measures (true positives).

Conversely, inaccuracies can manifest as (a) false negatives: where the model perceives the device as fast and consequently proceeds with normal I/O operations; (b) false positives: where the model predicts the device to be slow, yet the device is capable of fast service.

When utilizing predictive tools under low-latency device conditions and undertaking actions for slower devices, such as retracting I/O and resubmitting it to redundant devices, the associated overhead becomes non-negligible. Therefore, the impact of false positive judgments far exceeds that of false negative judgments. To mitigate false positive occurrences as much as possible, we incorporate heuristic algorithms following the model's predictions. Upon predicting a device to be slow, an additional heuristic judgment is invoked.

Within this heuristic judgment, the latency of the most recent Rh (where Rh is less than R, default Rh=R/4) I/O operations and the length of the waiting queue upon completion are evaluated over time. If a certain proportion of the Rh I/O operations marked as fast points and there is a decreasing trend in the length of the waiting queue, the model's prediction is modified to fast. In other cases, it will not change the predicted results. Because when the queue length decreases and the waiting time for each I/O becomes shorter, it is likely that the SSD has recovered from a slow state to a normal state (such as garbage collection ending), and I/O can continue to be sent to this SSD without being affected.

V. Evaluation

In this section, we first describe our evaluation setup (Section V-A) and then present the results that answer the following important questions:

- Model accuracy (Section V-B): Which model is more suitable for solving this classification task?
- Metrics (Section V-C): Discussion on Metric selection
- Overhead (Section V-D): Has LatVision introduced minor additional overhead?
- False Positive (Section V-E): How many false positives are present in LatVision?

A. Setup

1) Workloads.: Our ultimate goal is to evaluate whether LatVision can accurately predict the slow state of devices. In daily scenarios, the probability of SSDs being in a slow-running state is very low, possibly less than 1%, so we need to ensure that the device's usage space exceeds 90% before collecting the training dataset. When the remaining space on the device is minimal, it is more likely to enter a slow-running state. In summary, we need to collect data from devices that are in a busy state for training; otherwise, it may result in a scarcity of slow samples and low training efficiency. Our workload is constructed using Fio and Filebench, and tested with the default configuration of file system read/write loads.

2) Device and Environments.: In our experiment, we employed three widely adopted solid-state drives (SSDs) for testing: the Samsung 980Pro SSD, Samsung 970 EVO, and Western Digital SN770. Unless otherwise specified, Samsung 980Pro SSD will be used for testing in the experiment by default. These selections were made to encompass diverse hardware configurations and performance profiles, aiming to comprehensively assess the generality and efficacy of LatVision. Prior to this evaluation, all devices underwent several months of utilization to ensure their performance closely mimicked real-world conditions.

Concurrently, our experiments were conducted on a machine running Ubuntu 22.04 operating system. This machine featured a 2.6GHz 18-core (36-thread) Intel i9-7980XE processor with 64GB of DRAM, ensuring ample computational resources to support our experimental requirements. Apart from the variables mentioned in the comparative evaluation experiments, all other environmental configurations were maintained at their default settings to ensure the comparability and accuracy of the experimental outcomes.

B. Model Accuracy

In order to select the appropriate model among various options for completing the classification task, we chose to experiment with classification models that are currently considered mature (introduced in Section IV-D). Additionally, we conducted two experiments. On one hand, we made a simple extension to the prediction model of LinnOS in the kernel, transitioning it from predicting individual slow I/Os to predicting consecutive slow I/Os. On the other hand, drawing inspiration from the implementation pattern of neural networks in LinnOS, we independently developed the simplest form of a neural network, which we named Light Neural Network (LNN). Based on LinnOS, we expanded the input end of the neural network. To predict the continuity of slow I/Os, our tool requires more input information.

During the collection of training data, we collected data from three SSDs respectively and automatically labeled them, then unified the input format for training all neural networks. In model training, we trained

TABLE I
PREDICTION ACCURACY OF EACH MODEL.

	Samsung 980 Pro	Samsung 970 EVO	Western Digital SN770
LinnOS	49.65% ± 1.03%	46.30% ± 0.10%	47.97% ± 0.08%
Light Neural Network	85.28% ± 0.68%	86.90% ± 0.66%	84.01% ± 0.80%
RNN	86.22% ± 1.12%	86.97% ± 0.59%	87.35% ± 0.82%
1D CNN	77.81% ± 0.07%	75.09% ± 0.38%	75.36% ± 0.60%
DNN	89.69% ± 1.02%	91.44% ± 1.19%	87.10% ± 1.87%
ResNet	94.10% ± 0.66%	93.87% ± 1.69%	93.27% ± 0.98%
MobileNet	93.05% ± 0.40%	93.74% ± 0.31%	93.39% ± 0.40%
EfficientNet	**96.89%** ± 0.59%	**95.30%** ± 0.94%	**94.30%** ± 0.61%

TABLE II
THE IMPACT OF INCREASING KEY METRICS ON PREDICTION ACCURACY.

Metrics	Recent I/Os Information (R=20)	Recent I/Os Information (R=40)	+Remaining disk space	+Number of active related threads
Light Neural Network	76.31%	79.97%	83.01%	85.28%
RNN	73.98%	75.51%	82.03%	86.22%
1D CNN	65.66%	70.23%	75.50%	77.81%
DNN	85.21%	85.97%	88.76%	89.69%
ResNet	83.10%	83.32%	89.19%	94.10%
MobileNet	86.12%	89.30%	92.69%	93.05%
EfficientNet	**92.86%**	**93.65%**	**95.06%**	**96.89%**

separately for different SSDs. During testing, we also conducted tests on three disks respectively, using Fio and Filebench as the testing loads. Additionally, we recorded the latency of each I/O for validation purposes. We spend 60 minutes each time testing on each model. We conducted 3 repeated tests on each model, calculating the accuracy of the prediction models.

The accuracy results of testing each model are shown in Table I. From the table, we can observe that in terms of accuracy, EfficientNet exhibits significant advantages compared to other models. Moreover, the Light Neural Network (LNN) can surpass 1D CNN in accuracy, and since LNN has a simpler structure, it is also a preferable choice for predicting under resource-constrained scenarios. On the other hand, it is evident that the accuracy of model predictions is not strongly correlated with the actual disk performance. To avoid redundant results, all subsequent findings are based on experimental data from the Samsung 980Pro SSD.

C. Metric

In order to accurately predict consecutive slow I/Os in devices, selecting appropriate input metrics is crucial. As mentioned in Section IV-C, the metrics we discussed have the potential to improve the accuracy of prediction models. The reason for gradually increasing the input metrics during testing is that each additional metric introduces additional overhead for data transmission during real-time prediction. If a metric does not significantly improve accuracy and its transmission incurs more resource overhead, it will be excluded from our model. However, if a metric can significantly improve prediction accuracy, its selection becomes a trade-off. We need to balance between performance overhead and accuracy. After each new metric is introduced, we conduct a complete process of data collection, training, and testing.

The results of gradually increasing input metrics and conducting model tests are shown in Table II. The recent I/O information includes the following components: the number of currently pending I/Os, the latency of the R most recently completed I/Os, and the number of pending I/Os at the time when each of the R completed I/Os arrived. These three parameters intricately describe the current state of the device and its recent I/O operations.

We observed that when R=20, the EfficientNet model achieved a prediction accuracy of 92.86%. However, increasing R to 40 resulted in nearly a 2x increase in eBPF transmission resource overhead, with no significant improvement in model prediction accuracy. Therefore, simply increasing the value of R is not an optimal choice for expanding metrics.

However, we attempted to incorporate metrics from

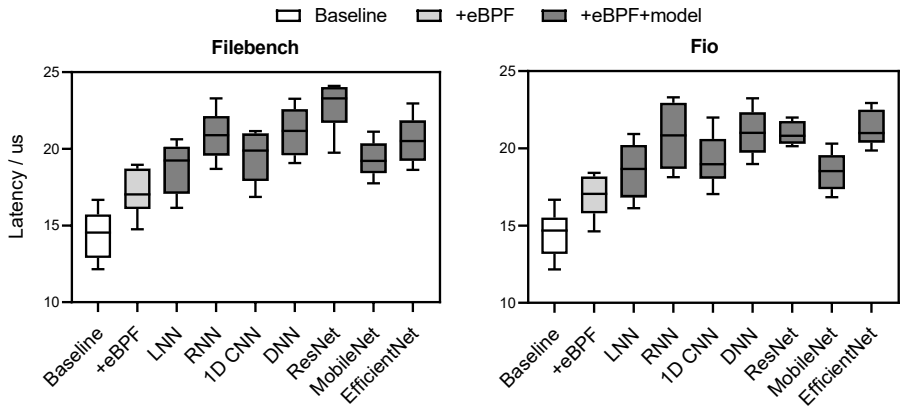

Fig. 4. Additional overhead generated by each prediction model (tested by Filebench and Fio workloads).

a macroscopic perspective for prediction, including the remaining disk space and the number of currently active relevant processes. When incorporating remaining disk space as a metric, the model's prediction accuracy significantly improved, reaching a maximum of 95.6%. Furthermore, when continuing to include the number of currently active relevant processes, the accuracy increased to 96.89%. Both of these metrics can reflect the current state of disk and system operation, closely related to I/O execution latency, thus justifying the improvement in prediction accuracy.

D. Overhead

When predicting slow I/O for low-latency devices, attention must be paid to the overhead of the prediction tool itself. Because if the overhead of the tool exceeds the impact of slow I/O, or if the operation of the tool affects the normal operation of the device, then such a prediction tool becomes meaningless.

Our goal is to ensure high accuracy while minimizing overhead as much as possible. To evaluate the additional overhead of LatVision, we designed the following experimental setup. The additional overhead introduced by LatVision is divided into two parts: one part is the eBPF data transmission module. Although the execution of eBPF hook functions does not affect the normal operation of I/O, it still affects the environment load to some extent. The other part is the prediction module of the model, which introduces additional performance overhead during inference computation. Generally, the more complex the model structure, the greater the overhead. The training of the model is independent of LatVision's operation and is not included in the additional overhead.

During the testing process, we used Filebench and Fio as the workload, with the performance results without running LatVision as the baseline. Then, we added the eBPF data collection module for testing. eBPF data collection is independent of the model. Finally, we tested each prediction model. It should be noted that here "latency" refers to the latency in the Fio results, which is the performance latency across the entire I/O storage stack. Although this is somewhat different from the slow I/O described in our device, it does not affect our observation of the proportion of performance degradation after adding latency.

We test overhead by both Filebench and Fio workloads, the results are shown in Fig.V-C. It demonstrates that LNN exhibits the least additional overhead among all models. This is attributed to its utilization of the simplest neural network architecture, thereby showcasing relatively higher efficiency in performance and resource utilization. However, despite EfficientNet's commendable accuracy performance, its introduction of a 37% additional overhead relative to the baseline model is considered a notable drawback. In contrast, MobileNet, as a lightweight model, while experiencing some loss in accuracy, significantly outperforms EfficientNet in terms of performance. After a thorough comparison of the two components, we have decided to employ MobileNet as the model for our prediction tasks. This decision is based on MobileNet's ability to achieve high predictive accuracy while maintaining relatively low computational overhead.

E. False Positive

When predicting performance for low-latency devices, a single incorrect prediction can have significant

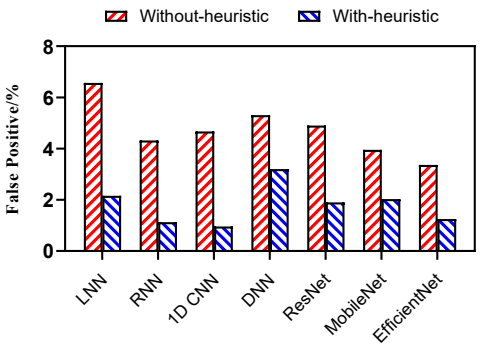

Fig. 5. Model prediction result false positive (optimized by heuristic algorithm).

consequences. Errors in prediction models can be categorized into two types: (a) false negatives, where the model mistakenly believes the device is operating at high speed when it is actually running slowly, and (b) false positives, where the model predicts the device is running slowly but it can actually provide fast service.

As we analyzed in Section IV-E, false positive predictions can potentially introduce greater additional overhead. Therefore, we applied heuristic optimization on top of the prediction models. We tested both the without-heuristic and with-heuristic models in these two rounds of testing.

Fig.5 illustrates the comparative results of these models in the two testing rounds. We can observe that the heuristic algorithm has addressed most of the false positive issues. Although the occurrence probability of false positives in prediction models is not high, we can still mitigate potential risks caused by false positives using simple heuristic algorithms.

VI. Conclusion and Discussion

We have presented an eBPF-base SSD performance prediction tool-LatVision. We have conducted a detailed discussion and comparison of the models and metrics used in the tool, and brought in-depth analysis in terms of accuracy and overhead. We have shown the feasibility of using a neural network for making frequent, black-box live inferences for SSD performance. Our work successfully brings predictability on persisting tail latency in SSD. We also believe that LatVision's success leads to exciting discussions and questions that can spur future work:

A. On Further Process

The predictive outcomes of LatVision hold significant potential for practical applications. Achieving real-time, high-accuracy predictions of whether an SSD is likely to enter a slow-running state enables several key tasks:

Improved I/O allocation: In performance-sensitive workloads where stringent requirements exist for I/O latency, reallocating I/O operations becomes feasible. By redirecting I/O originally destined for a slow-speed disk to redundant disks, performance degradation can be mitigated.

Enhanced disk management: On a macroscopic level, prolonged periods of slow operation in a disk indicate underlying performance-related issues. This insight allows for targeted interventions and resolutions based on the specific circumstances. Furthermore, in the context of disk arrays, it presents an opportunity for effective load balancing strategies.

Service Level Agreement (SLA) Management: For cloud service providers with a large number of SSDs, SSD I/O performance prediction can be used to ensure service quality and dynamically adjust resources to meet the performance needs of different customers

B. On Other Integrations and Extensions

Our research raises an intriguing question: how can the block layer effectively learn the latency behavior of SSDs, given their complex idiosyncrasies, with only a few observable features? This understanding holds the potential to inform the design and implementation of various higher layers in storage systems, including RAID configurations, direct device access (such as SPDK [29]), user or device-level filesystems, and distributed storage solutions. Furthermore, it opens avenues for future exploration, suggesting the possibility of integrating latency inference capabilities directly into SSDs at lower layers.

While one might argue that SSDs already possess comprehensive internal knowledge and therefore do not require black-box predictions, an alternative perspective emerges. SSD vendors could potentially leverage machine learning techniques consistently across diverse internal architectures, obviating the need to continually redevelop inference logic with each hardware or policy modification. This approach offers a streamlined method for adapting to changes in internal hardware, logic, and policies. Additionally, SSD vendors might consider employing a "gray-box" learning approach, blending external observations with internal knowledge to enhance prediction accuracy and robustness.

REFERENCES

[1] "Solid-state drive," https://www.ibm.com/topics/solid-state-drives.

[2] M. Mesnier, F. Chen, T. Luo, and J. B. Akers, "Differentiated storage services," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 57–70. [Online]. Available: https://doi.org/10.1145/2043556.2043563

[3] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013. [Online]. Available: http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext

[4] R. Lu, E. Xu, Y. Zhang, Z. Zhu, M. Wang, Z. Zhu, G. Xue, M. Li, and J. Wu, "NVMe SSD failures in the field: the Fail-Stop and the Fail-Slow," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 1005–1020. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/lu

[5] R. Lu, E. Xu, Y. Zhang, F. Zhu, Z. Zhu, M. Wang, Z. Zhu, G. Xue, J. Shu, M. Li, and J. Wu, "Perseus: A fail-slow detection framework for cloud storage systems," in *USENIX Conference on File and Storage Technologies*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257285445

[6] J. Hao, Y. Li, X. Chen, and T. Zhang, "Mitigate HDD fail-slow by pro-actively utilizing system-level data redundancy with enhanced HDD controllability and observability," in *35th Symposium on Mass Storage Systems and Technologies, MSST 2019, Santa Clara, CA, USA, May 20-24, 2019*. IEEE, 2019, pp. 205–216. [Online]. Available: https://doi.org/10.1109/MSST.2019.000-2

[7] S. Wu, C. Du, H. Li, H. Jiang, Z. Shen, and B. Mao, "CAGC: A content-aware garbage collection scheme for ultra-low latency flash-based ssds," in *35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021, Portland, OR, USA, May 17-21, 2021*. IEEE, 2021, pp. 162–171. [Online]. Available: https://doi.org/10.1109/IPDPS49936.2021.00025

[8] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The unwritten contract of solid state drives," in *Proceedings of the Twelfth European Conference on Computer Systems, EuroSys 2017, Belgrade, Serbia, April 23-26, 2017*, G. Alonso, R. Bianchini, and M. Vukolic, Eds. ACM, 2017, pp. 127–144. [Online]. Available: https://doi.org/10.1145/3064176.3064187

[9] J. Wu, J. Li, Z. Sha, Z. Cai, and J. Liao, "Adaptive switch on wear leveling for enhancing I/O latency and lifetime of high-density ssds," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4040–4051, 2022. [Online]. Available: https://doi.org/10.1109/TCAD.2022.3197340

[10] H. Li, M. Putra, R. Shi, X. Lin, N. Gregory, R. Ganger, H. Gunawi, and G. Ganger, "Ioda: A host/device co-design for strong predictability contract on modern flash storage."

[11] A. Tai, I. Smolyar, M. Wei, and D. Tsafrir, "Optimizing storage performance with calibrated interrupts," *ACM Transactions on Storage*, p. 1–32, Feb 2022. [Online]. Available: http://dx.doi.org/10.1145/3505139

[12] M. Hao, L. Toksoz, N. Li, E. E. Halim, H. Hoffmann, and H. S. Gunawi, "LinnOS: Predictability on unpredictable flash storage with a light neural network," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 173–190. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/hao

[13] "A thorough introduction to ebpf," https://lwn.net/Articles/740157/.

[14] "Extending extended bpf," https://lwn.net/Articles/603983/.

[15] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance implications of packet filtering with linux ebpf," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 01, 2018, pp. 209–217.

[16] "Non-volatile memory express," https://www.ibm.com/topics/nvme.

[17] "3d xpoint: A breakthrough in non-volatile memory technology," https://www.intel.com/content/www/us/en/architecture-and-technology/intel-micron-3d-xpoint-webcast.html.

[18] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, D. Srinivasan, B. Panda, A. Baptist, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, "Fail-slow at scale: Evidence of hardware performance faults in large production systems," *ACM Trans. Storage*, vol. 14, no. 3, p. Article 23, 2018.

[19] D. G. Andersen and S. Swanson, "Rethinking flash in the data center," *IEEE Micro*, vol. 30, no. 4, pp. 52–54, 2010.

[20] K. Wu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Towards an unwritten contract of intel optane SSD," in *11th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2019, Renton, WA, USA, July 8-9, 2019*, D. Peek and G. Yadgar, Eds. USENIX Association, 2019. [Online]. Available: https://www.usenix.org/conference/hotstorage19/presentation/wu-kan

[21] "Filebench," https://github.com/filebench/filebench.

[22] "Flexible i/o tester," https://github.com/axboe/fio.

[23] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, no. 2, 2012.

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:205001834

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE*, 2016.

[27] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *ArXiv*, vol. abs/1905.11946, 2019.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[29] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul, "Spdk: A development kit to build high performance storage applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 154–161.