# Evaluating Achievable Latency and Cost SSD Latency Predictors (MittOS Model Inference)

Olivia Weng and Andrew A. Chien
*The University of Chicago*

**Abstract**—Cutting tail latencies at the millisecond level in internet services for good response times in data-parallel applications is possible by integrating MittOS, an OS/data center interface. Typically MittOS analyzes white-box information of the internals of devices such as SSD's and decides if a given server can "fast reject" a service request. But commercial SSD's have a black-box design, so MittOS researchers have developed machine learning models to determine if requests to commercial SSD's can be rejected or not. When run on CPUs however, these models cannot predict in the time it takes an SSD to fully process a request, defeating MittOS's fast-rejecting abilities. We demonstrate that ASICs such as the Efficient Inference Engine (EIE) accelerate the prediction times of these MittOS models well within the time it takes an SSD to complete a request at minimal cost, cutting SSD tail latencies. EIE achieves 2.01 $\mu$s inference latency while incurring minimal area costs (20.4 mm$^2$) and power costs (0.29 W). We show that integrating machine learning into the critical path of operating systems becomes cost-efficient and within reason.

## 1 Introduction

Tail tolerance is a widely-acknowledged and studied problem in internet services [1]; it is considered fundamental to providing good response for applications such as search, feeds, ads, and more that use large-scale parallelism to process large data sets in service of a single service request. MittOS proposes an OS/data center interface that frames a latency threshold with each request, giving each of the parallel servers an opportunity to "fast reject" the request. If the parallel servers can make these decisions well, MittOS can effectively reduce service tail latencies at the millisecond level in disks, SSD's, and the OS cache [2].

However, most of the work to date with MittOS assumes "white-box" performance models for server latency, that is these devices are assumed to expose their internal complexities such as how full IO queues are, sources of variation in latency, etc. This makes a common IO element, commercial SSDs, a challenge to incorporate in a MittOS system. Commercial SSDs generally employ a black-box design because of the algorithms and structures within are a critical part of their competitive performance and cost. For such devices, we cannot build a fast reject responder based on knowledge of the internals, and this leaves MittOS with no way to predict whether a given SSD request's SLO deadline can be met.

Recently, MittOS researchers are exploring the use of supervised machine learning to build black-box models for closed SSDs and more generally subsystems of all kinds without explicit modelling. This has produced a series of deep neural network models that are the focus of study in this report. These models can be trained with a black-box SSD as the training model, and be used to predict long latency requests, and thus "fast reject" for the black-box SSDs [3]. Others report the accuracy of these models [3], but here we focus on the "performance as a system component" evaluation, focusing on time to prediction latency and the cost (in power, silicon) to deliver those predictions.

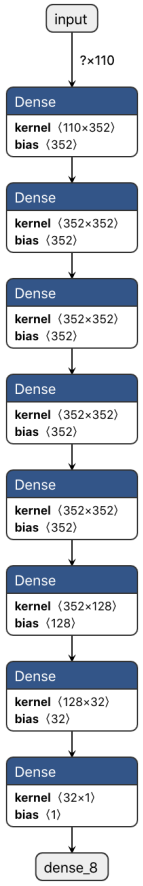


Fig. 1. The precise_linear model

Because SSDs have much lower latency (~50 microsec-

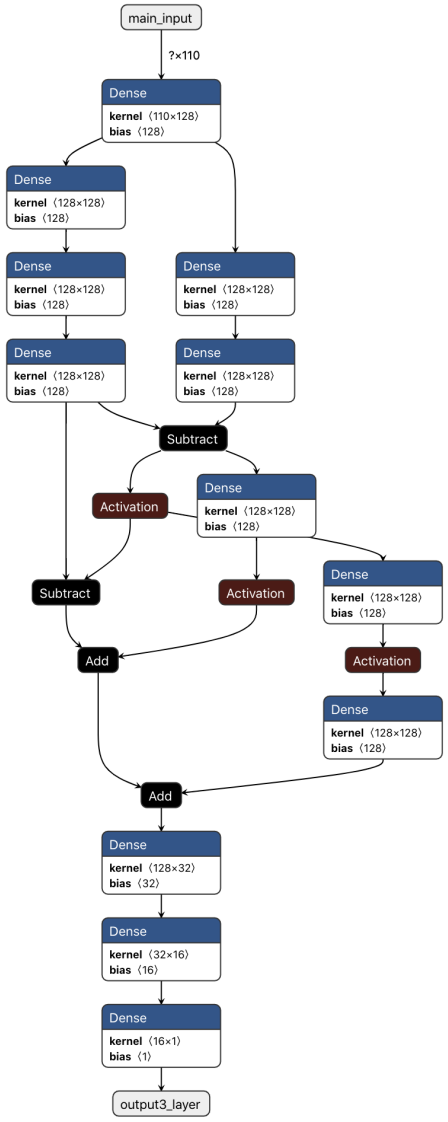


Fig. 2. The good_custom model

onds) when compared to traditional HDDs (5+ milliseconds) and even data center network latencies (100s of microseconds when loaded), predictors for systems built on SSDs must make decisions much more quickly. SSDs also support much higher IOPs rates than HDDs, so on a per-device basis, prediction (or "fast reject" decisions) must be made at a much higher rate. Together, these make prediction latency and cost critical factors. Prediction is useful for tail-tolerance only if it can be done in approximately the latency for an SSD to serve an IO request, which is on the order of $\mu$s (a 4KB read takes 100 $\mu$s).

The MittOS team has produced a number of models. The precise_linear model (Figure 1), is a simple, small neural network with eight fully-connected layers and 583k weights, and achieves 99.94% accuracy. A second model from the team is the good_custom model (Figure 2), is a smaller but more complex model, that achieves 97.94% accuracy. These accuracy levels are high enough to be useful in a scalable MittOS system.

In this report, we study the achievable latencies and costs for running precise_linear and good_custom inference models. As discussed above, the workings of tail latency mitigation for SSDs demand prediction latencies on the scale of ~50 $\mu$s or less. Moreover, the cost dynamics of the storage industry require that the power and circuit costs of implementing these models be small on a per-SSD basis. To explore these considerations, we explore well-known techniques to reduce the compute requirements of the models (compression techniques such as pruning [4], [5]). We also consider execution on an accelerator, the Efficient Inference Engine (EIE) [6], an ASIC optimized for running compressed model inference. Our results are summarized below:

1) Studies show that precise_linear and good_custom can be reduced in size and computation cost by 98% at a cost of 2% accuracy and 95% at no apparent reduction in accuracy by using well-known pruning approaches.
2) For the precise_linear, latencies of 25 $\mu$s can be achieved for the unpruned network, with latencies scaling down to around 2.5 $\mu$s for the heavily pruned network. For good_custom, a smaller model, the unpruned latencies were under 10 $\mu$s, and dropped below 2 $\mu$s for the pruned network. All of these performance estimates were based on a 32 PE EIE, and are well within the performance requirements for MittOS.
3) Considering the precise_linear model, larger so it has higher power requirements, power estimates of $\frac{1}{3}$ to $\frac{2}{3}$ of a watt are required for maximum inference rate (400,000/second), based on a 65nm CMOS process, so a more recent process could reduce this requirement well below 0.1 watts—well within the power envelope feasible in the application context.

Our overall conclusion is that the achieved performance is fast enough for SSD prediction, and the costs of such are well within those feasible on a per-SSD basis within a data center.

## 2 Reducing Model Size

### 2.1 The precise_linear model

To prune the precise_linear model, we used Tensorflow's Keras-based weight pruning API, which gradually removes unnecessary low-weight connections and then retrains the newly pruned model, repeating this process until a desired sparsity is achieved. Based on [5], the API looks for small weights because they have been shown to have little to no impact on inference when compared to larger weights. When looking at the weight distribution of precise_linear (Figure 3), we see that our model is made up of mostly such small weights, making it particularly amenable to pruning.

We pruned the model to 90%, 95%, 97%, and 98% sparsities to see how sparse we could go with respect to accuracy. We did not look into 99% sparsity because pruning that aggressively on a small model had the undesired effect of completely losing connections between layers. From the weight distributions of the pruned models in Figure 4, we can see how low-magnitude weights are gradually removed.

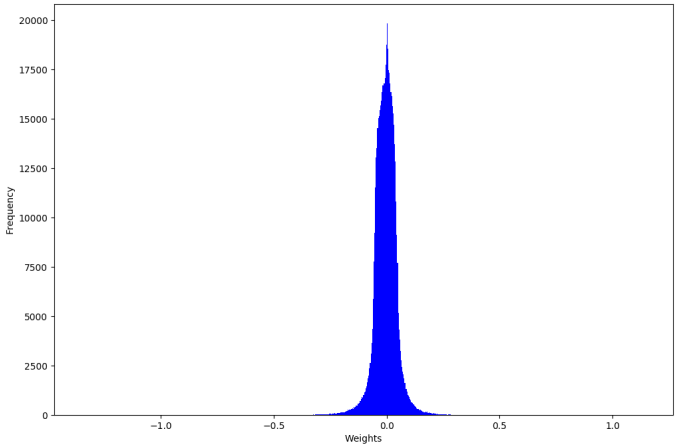


Fig. 3. Weight distribution of the precise_linear model

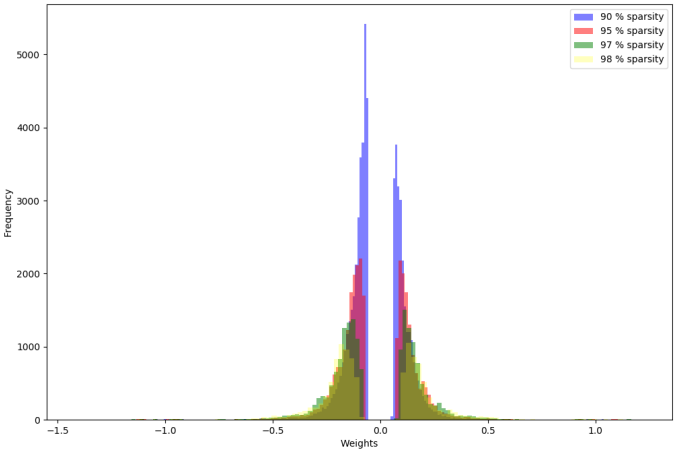


Fig. 4. Weight distribution of the pruned precise_linear models

TABLE 1
Pruned precise_linear model accuracies

| Sparsity (%) | Accuracy (%) |
|---|---|
| 0 | 99.94 |
| 90 | 99.87 (0.07 loss) |
| 95 | 99.61 (0.33 loss) |
| 97 | 98.82 (1.12 loss) |
| 98 | 98.03 (1.91 loss) |

It further follows from these weight distributions that low-magnitude weights are, in fact, low-impact because the model does not suffer much loss in accuracy. In Table 1, we can see that even the largest loss in accuracy, which came from pruning to 98% sparsity, is only 1.91%.

### 2.2 The good_custom model

We used a similar method to prune the good_custom model. However, since the good_custom model is so small ( 150k weights), we pruned good_custom to 90% and 95% sparsity, as any more aggressive pruning would prune away all of the connections between layers, destroying the model. In Figure

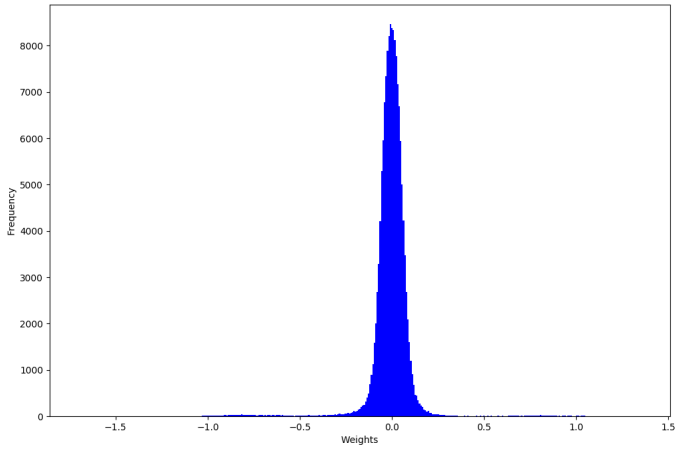


Fig. 5. Weight distribution of the good_custom model

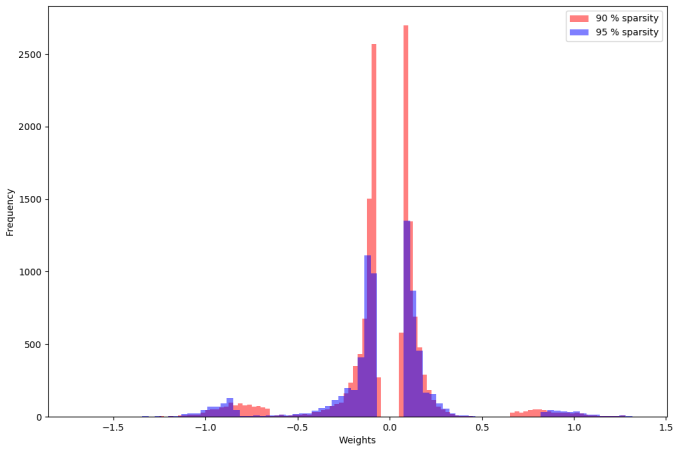


Fig. 6. Weight distribution of the pruned good_custom models

5, we can see the weight distribution before pruning, and in figure 6, we can see the weight distribution after pruning.

However, we had a significant problem with pruning. The accuracy of the pruned versions are unstable and seems to predict that every request is NON-EBUSY, meaning that an SSD is not currently busy fulfilling a request. All of the pruned versions started their retraining at 92% accuracy and stayed stuck there, likely because of unstable weights since the model we have is under-trained and not robust enough to undergo such aggressive pruning [5]. After pruning, we calculated that both 90% and 95% sparse models exhibited 92.18% accuracy, but this may not be a good representation of the model's accuracy. Of the 100,000 requests predicted in our testing, 92,178 should be and were indeed predicted to be NON-EBUSY; however, the remaining 7,822 requests which should have been predicted to be EBUSY were all inaccurately predicted to be NON-EBUSY. As such, these accuracy results do not meaningfully reflect how well the pruned good_custom models predict signals since they simply infer all requests to be NON-EBUSY.

## 3 Latency and Power

Having pruned our models, we can look into how they would perform on a CPU and the EIE. Performance on

TABLE 2
Average latency of precise_linear models ($\mu$s / inference)

| Sparsity (%) | CPU (i7-8700) | EIE (1PE) | EIE (2PEs) | EIE (4PEs) | EIE (8PEs) | EIE (16PEs) | EIE (32PEs) |
|---|---|---|---|---|---|---|---|
| 90 | 14.1 | 81.1 | 40.5 | 20.3 | 10.1 | 5.07 | 2.53 |
| 95 | 13.0 | 40.7 | 20.3 | 10.2 | 5.08 | 2.54 | 2.31 |
| 97 | 12.4 | 24.7 | 12.4 | 6.18 | 3.09 | 2.81 | 2.55 |
| 98 | 13.4 | 17.3 | 8.63 | 4.32 | 2.45 | 2.23 | 2.01 |

TABLE 3
Average latency of good_custom models ($\mu$s / inference)

| Sparsity (%) | CPU (i7-8700) | EIE (1PE) | EIE (2PEs) | EIE (4PEs) | EIE (8PEs) | EIE (16PEs) | EIE (32PEs) | EIE (64PEs) |
|---|---|---|---|---|---|---|---|---|
| 0 | 7.49 | 251 | 126 | 62.8 | 31.4 | 15.7 | 7.85 | 3.93 |
| 90 | 7.08 | 25.5 | 12.7 | 6.37 | 3.19 | 1.99 | | |
| 95 | 7.36 | 13.3 | 6.65 | 3.32 | 2.08 | 1.89 | | |

TABLE 4
Peak throughput, area, and power of EIE

| | CPU (i7-8700) | EIE (1PE) | EIE (2PEs) | EIE (4PEs) | EIE (8PEs) | EIE (16PEs) | EIE (32PEs) | EIE (64PEs) |
|---|---|---|---|---|---|---|---|---|
| Peak Throughput (GOPS) | 82.8 | 1.6 | 3.2 | 6.4 | 12.8 | 25.6 | 51.2 | 102.4 |
| Area ($mm^2$) | 14.06.25 | 0.638 | 1.28 | 2.56 | 5.10 | 10.2 | 20.4 | 40.8 |
| Power (W) | 65 | 0.0092 | 0.018 | 0.037 | 0.073 | 0.15 | 0.29 | 0.59 |

the EIE is calculated via estimating based on EIE's reported peak throughput (1.6 GOPS) and examination of the sparsity of the weight matrices to account for workload GOPs and load imbalance. EIE scales near-linearly up until it hits significant load imbalance, when on average $\leq$1 rows are allocated per PE at a given time, as marked by the yellowed cells. According to [6], at this level of load imbalance, EIE suffers from high variation in distributing the rows to the PEs, as many PEs remain idle. Area and power estimates are based on EIE's reported area and power consumption per PE and any additional components (e.g., the Leading Non-Zero Detect unit).

The weight matrices of the precise_linear model are small (352 rows max), and performance drops around 16 and 32 PEs because increasing sparsity and having less weights to distribute among the PEs incur even more load imbalance. In spite of this, we can see that we achieve $<$10 $\mu$s latency easily with the EIE. As seen in the tables below, at 98% sparsity, which has comparable accuracy as seen previously, EIE can achieve a mere 2.01 $\mu$s/inference latency with 32 PEs while only incurring an area footprint of 20.4 $mm^2$ and dissipating 0.29 W (Table 4), which is significantly low-cost compared to allocating an entire CPU for inference. Because of load imbalance, however, it would be the most cost-effective to use 8 PEs, which achieves 2.45 $\mu$s/inference for our 98% sparse model (Table 2), well within our $<$10 $\mu$s latency requirement. This way we have fewer idle PEs at a given time while further reducing area footprint to 5.10 $mm^2$ and power dissipation to 0.073 W (Table 4).

While the runtime latencies of the pruned good_custom models seem promising, their unstable accuracies give us pause. But, since the original, unpruned good_custom model is so smallits weight matrices are only 128 rows max, the EIE does not end up suffering from the load balance issues that the sparse ones do. As such, its runtime on 64 PEs is comparable to the runtime of the pruned precise_linear models (Table 3). We did not, however, calculate the runtimes for 32 and 64 PEs for the pruned, sparse good_custom models because at those points the load imbalance is so severe that the EIE becomes extremely volatile and the runtimes become unclearperhaps running the EIE simulator on these settings may reveal these runtimes.

## 4 Future Work

Not only is pruning an effective method of compression, quantization has been shown to reduce model size while maintaining accuracy [4]. Although EIE can be computed in float32 with little to no change in area and power (SRAM accesses dominate), it aims to quantize weights and compute in float16, so it would be worth seeing how robust precise_linear is in float16. It is also worth looking into how precise_linear and good_custom would run on the EIE simulator to get a clearer idea of how much the load imbalance at a higher number of PEs affects performance.

## 5 Conclusion

Since the EIE can achieve such low latency with significantly little area usage and power consumption, these MittOS neural networks become useful and effective tools in predicting SSD I/O requests, bringing us one step closer to integrating MittOS into operating systems. It becomes unreasonable to assume that we must rely on the CPU to support operating systems, as introducing machine learning via ASICs into the

critical path of operating systems becomes a reality. With 32 PEs, EIE can predict our 98% sparse precise_linear model in 2.01 $\mu$s, and with 64 PEs, EIE can predict the unpruned good_custom model in 3.93 $\mu$s. Thus, our models achieve latencies well below the $<$10 $\mu$s latency of SSD reads and writes. As such, by means of ASICs like the EIE, introducing our MittOS models into the CPU or near the SSD controller to serve SSD requests, cutting their millisecond tail latencies, becomes practicable.

## REFERENCES


[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, p. 7480, Feb 2013.

[2] M. Hao, H. Li, M. H. Tong, C. Pakha, R. O. Suminto, C. A. Stuardo, A. A. Chien, and H. S. Gunawi, "Mittos: Supporting millisecond tail tolerance with fast rejecting slo-aware os interface," *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP 17*, 2017.

[3] M. Han, A. Zhang, A. Roaffa, A. A. Chien, H. Hoffman, and H. S. Gunawi, "Machine learning for operating systems: A case of using deep neural network for os-level i/o latency prediction," in *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018, (Poster).

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[5] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," 2017.

[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.