# No DNN Left Behind: Improving Inference in the Cloud with Multi-Tenancy

Amit Samanta[1,2], Suhas Shrinivasan[1,2], Antoine Kaufmann[2], Jonathan Mace[2]
[1] Student Author [2] Attending Conference

Max Planck Institute for Software Systems

## MOTIVATION

With the rise of machine learning, inference on deep neural networks (DNNs) has become a core building block on the critical path for many cloud applications. Inference, in contrast to training, is a low-latency online task that makes predictions on-demand using a trained DNN. In the cloud, inference workloads bear similarity to other online applications like databases, web services, and microservices. DNNs are typically hosted separately from application logic, and accessed via RPC. We identify a few desirable properties for inference: *low latency*, *high elasticity*, and *cost-effectiveness*.

With these desirable properties in mind, we find that existing approaches to deploying DNNs for inference fall significantly short. The conventional approach, based on provisioning containers or virtual machines, suffers from over-provisioning, slow auto-scaling, high cold-start latencies, and mismatched pricing abstractions. This forces users to compromise on consistent latency, elasticity, or cost-efficiency, depending on workload characteristics.

**Multi-Tenancy**: In this work, we propose to elevate DNN inference to be a first class cloud primitive provided by a shared multi-tenant system, akin to cloud storage and databases. Multi-tenant systems are only justifiable for *core datacenter functionality*, where there is a common need for the functionality across many different tenants and workloads. We believe that DNN inference is sufficiently important to justify a specialized multi-tenant system.

With this approach, users no longer provision per-workload resources. Instead, the system operator provisions resources for the system as a whole, and runs long-lived system processes that receive and execute requests from different tenants concurrently.

## SYSTEM OVERVIEW

Our system architecture is similar to existing systems such as shared filesystems and databases. Meta-operations are handled by a logically centralized controller. DNN inference is handled by worker processess spread across many machines.

The lifecycle from a user's perspective is to (1) upload a trained DNN to the system, then (2) send inference requests. The system performs inference when requests are received, and transparently scales based on the workload demand. Internally, the system distributes models to one or more workers. Inference requests are routed to whichever workers host the model. Workers host models from many tenants simultaneously, and multiplex execution across different models.

**Key Challenges and Opportunities**:

**Security** Shared systems execute requests of different tenants within the same, shared processes. Thus, users are no longer separated by rigid OS or VM boundaries. Hence, we must ensure security between different tenants' workloads.

**Performance Isolation** The system must prevent performance interference between different tenants. However, shared systems cannot rely on OS mechanisms for isolation between tenants, instead must address isolation at application level.

**Optimization-Based Scheduling** We exploit DNN predictability to do a much better job of request scheduling, both at request admission, and at finer granularity within the system. Instead of heuristic-based best-effort scheduling, we can confidently optimize an objective across all pending requests, such as minimizing average execution latency.

**Multi-Resource Scheduling** Worker processes host many models simultaneously, and alternate service between different tenants. In the worst case, each request may require loading and executing a model that isn't currently loaded. This introduces additional resource costs, such as the need to copy a model from a remote machine or cold storage. Similarly, if we use hardware accelerators, then models need to be copied from host memory to device memory. As discussed, execution latency for DNNs is inherently predictable. However, so too is transfer latency, since memory footprint of a DNN is fixed and known a priori. This leads to a multi-resource scheduling problem, has some unique constraints: (1) each resource is individually predictable; (2) resources are consumed one-at-a-time; and (3) scheduling decisions can be interposed before each resource. This provides an opportunity for high quality, fine-grained scheduling decisions.

**Memory Management** Not all inference requests incur memory transfer overheads because of caching. The memory footprint of a DNN is in the tens/hundreds of MBs; in contrast, today's web servers often exceed 1TB of main memory, present GPUs have up to 32GB device memory, and present TPUs have 64GB device memory. It is much more efficient to swap between cached models than to reload from scratch.