FedMT: Multi-Task Federated Learning with Competitive GPU Resource Sharing

Yongbo Yu, Fuxun Yu, Zirui Xu, Di Wang, Minjia Zhang, *Member, IEEE*, Ang Li, *Member, IEEE*, Chenchen Liu, *Member, IEEE*, Zhi Tian, *Fellow, IEEE*, and Xiang Chen, *Member, IEEE*

***Abstract*—Federated learning (FL) nowadays involves heterogeneous compound learning tasks as cognitive applications' complexity increases. For example, a self-driving system hosts multiple tasks simultaneously (e.g., detection, classification, segmentation, etc.) and expects FL to retain life-long intelligence involvement. However, our analysis demonstrates that, when deploying compound FL models for multiple training tasks on a GPU, certain issues arise: As different tasks' skewed data distributions and corresponding models cause highly imbalanced learning workloads, current GPU scheduling methods lack effective resource allocations; Therefore, existing FL schemes, only focusing on heterogeneous data distribution but runtime computing, cannot practically achieve optimally synchronized federation. To address these issues, we propose a full-stack FL optimization scheme to tackle both intra-device GPU scheduling and inter-device FL coordination for multi-task training. Specifically, our works illustrate two key insights in this research domain: Competitive resource sharing is beneficial for parallel model executions, and the proposed concept of "virtual resource" could effectively characterize and guide the practical per-task resource utilization and allocation; Additionally, architectural-level coordination improves FL performance by aligning task workloads with GPU utilization. Our experiments demonstrate that the FL performance could be significantly escalated. Specifically, we observed a 2.16×–2.38× increase in intra-device GPU training throughput and a 2.53×–2.80× boost in inter-device FL coordination efficiency across diverse multi-task scenarios.**

***Index Terms*—Federated Learning; Multi-Task Learning; GPU Resource Allocation; Resource Sharing**

I. Introduction

FEDERATED learning (FL) is a distributed training methodology allowing multiple computing devices to jointly train cognitive tasks without sharing private data [1], [2], [3]. This approach not only mitigates privacy concerns but also significantly reduces the bandwidth and latency requirements typically associated with centralized data aggregation, making FL especially suitable for edge computing scenarios where devices are resource-constrained and communication is often unreliable. As the deployment of intelligent applications is widely spreading in ever more complex settings, FL becomes an essential technique for leveraging distributed resources and enabling timely model improvement. Given the increasing complexity of these intelligent applications, it becomes imperative to explore more efficient ways to coordinate participating devices, as each one may have vastly different computational capabilities and varying data quality [4]. Meanwhile, the nature of these applications is expanding in complexity: As illustrated in Fig. 1, each device (e.g., a self-driving system[5]) can host multiple FL tasks (e.g., detection, classification, segmentation), each with its own neural network model or sub-model structure, together forming a compound multi-task learning scenario[6]. Similarly, a hospital may participate in training several federated models at once (for different diseases)[7], [8], all on one machine. These scenarios demand efficient GPU sharing to fully utilize hardware and speed up learning.

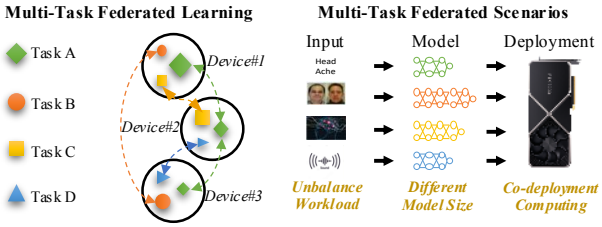


Fig. 1: Federated Learning with Multi-Tasks

However, a series of challenges arise when we put FL frameworks into practice under multi-task settings: **(1)** From the FL algorithm perspective, data and task heterogeneity are still the leading cause of difficult convergence issues [9], [10]. As shown in Fig. 1, significant heterogeneity generally exists across clients regarding learning tasks and accessible data volumes, causing potential synchronization complexity and degraded model accuracy if not properly handled. **(2)** From the local node computing perspective, each client typically needs a fine-grained multi-task computing approach, because each node might train multiple models (or multiple sub-branches of a larger model) in parallel [11], [12], [13]. This leads to the need for carefully managing concurrent task executions on GPUs, where suboptimal scheduling would result in resource contention or underutilization. **(3)** From the system federation perspective, multi-task FL still needs to ensure effective intra-device and inter-device coordination. On the one hand, the local device must maximize GPU usage to handle concurrent tasks; on the other hand, these tasks also require global synchronization cycles and model aggregation across different devices. If the local scheduling or workload balance is inadequate, the training process may suffer from uneven progress across the federation, slowing the training time or requiring more communication overhead[14].

Y. Yu and Z. Tian: George Mason University, USA ({yyu25, ztian1}@gmu.edu).
F. Yu, Z. Xu, and D. Wang: Microsoft ({fuxunyu, v-ziruixu, wangdi}@microsoft.com).
M. Zhang: University of Illinois Urbana-Champaign (minjiaz@illinois.edu).
A. Li: University of Maryland at College Park,USA (angliece@umd.edu).
C. Liu: University of Maryland Baltimore County, USA (ccliu@umbc.edu).
X. Chen (corresponding author): Peking University, China (xiang.chen@pku.edu.cn).

In the state-of-the-art, much attention has been paid to improving the convergence and generalization performance of FL under statistical heterogeneity scenarios (e.g., heterogeneous FL [15], asynchronous FL [16], clustered FL [17]), often through algorithmic innovations like model pruning [18] or gradient compression [19]. However, these methods are generally designed under the traditional single-task setup, leaving the emerging multi-task context—which is closer to real-world FL deployments—underexplored. In this paper, we focus on a complementary problem—how to efficiently schedule multiple co-running models under constrained GPU resources in multi-task FL scenarios, regardless of data distribution. Each federated task in this work is a supervised learning task for evaluation purposes; however, the method is agnostic to the training modality and would equally benefit unsupervised or self-supervised tasks running in parallel. In complex multi-task scenarios, multiple DNNs must be trained concurrently, often vying for limited GPU resources and requiring sophisticated scheduling beyond naive parallelism. Conventional parallelism techniques such as NVIDIA Multi-Process Service MPS [20] can partition GPU resources for co-located models, but they fail to account for significant differences in unbalance workload, DNN size, and runtime workload fluctuations. This can lead to performance bottlenecks where certain tasks may dominate resources while others remain idle or progress slowly.

Hence, in contrast to prior FL works that primarily focus on Non-IID learning optimization [9], [10] or single-task optimization, this paper takes a deeper look at the GPU architectural level scheduling problem in multi-task FL deployment. Concretely, three major obstacles must be tackled: **(1)** With heterogeneous model structures and workloads across tasks, how to adaptively allocate resources among different co-running DNN models to fully exploit the GPU? **(2)** When running multiple tasks concurrently, how to find an effective parallel strategy that leverages inter-task resource complementarities while avoiding severe contention? **(3)** How to extend such resource scheduling insights to a broader FL system, ensuring that multi-device coordination can benefit from the improved local throughput and thus accelerate global model training time to reach convergence?

By targeting these questions, we propose a full-stack optimization approach, named *FedMT*, which bridges intra-device GPU scheduling and inter-device FL coordination:

- We analyze *competitive resource sharing* that arises when multiple DNNs co-locate on a single GPU. Our study identifies severe contention and underutilization issues in naive allocations, motivating a novel *"virtual resource"* concept to systematically capture and optimize GPU concurrency.
- We develop a data-driven throughput model that accurately predicts multi-task performance under various batch sizes and resource budgets. This allows us to efficiently allocate virtual resource to maximum throughput.
- We propose a batch size adaptation mechanism to balance workloads between tasks at the inter-device level. By aligning each task's batch size with its data volume, we ensure no single task dominates or remains underutilized during local synchronization cycles, improving fairness and utilization.
- We introduce a communication overlap strategy that hides parameter upload/download latencies behind parallel computations. Through flexible scheduling priorities that consider both compute and communication times, we minimize idle waiting and accelerate global progress.
- We present an extensive experimental evaluation across diverse DNN architectures, datasets, and hardware platforms. Our results consistently show over $2\times$ speedup compared to standard baselines. In particular, by assigning appropriate GPU resource budgets and flexibly sharing these resources among heterogeneous tasks, we not only avoid underutilization but also effectively exploit concurrency. This acceleration of local training translates into faster system-wide FL convergence in a wide range of multi-task scenarios.

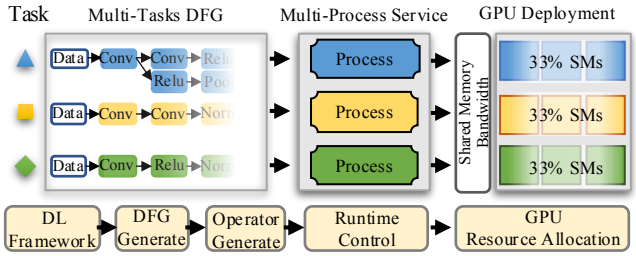


Fig. 2: Multi-Task Deep Learning with GPUs

II. Background and Motivation

A. Multi-Task Federated Learning on GPU

Multi-Task FL: Instead of a single-model setup, multi-task FL handles multiple learning objectives simultaneously. Each task's training data remains local to the client and is strictly used by that task's model training process only. Multi-task FL does not allow any direct data exchange between co-located tasks, and tasks only communicate their model updates to the central server. Therefore, the privacy of each task's data is preserved just as in standard FL. Specifically, each device j can hold I tasks, each with its own dataset $D_{i,j}$ and model parameters $W_{i,j}$. The overall FL objective is to minimize the global loss across all tasks and devices, often expressed as:

$$\begin{cases} \text{Local Training:} & \min\limits_{W_{i,j}} \sum\limits_{j=1}^{J} \text{Loss}\big(D_{i,j}, W_{i,j}\big), \\ \text{Parameter Fusion:} & W_{i,j}^{(k\text{ cycle})} \leftarrow \sum\limits_{j=1}^{J} \dfrac{|D_{i,j}|}{\sum_{j=1}^{J}|D_{i,j}|} W_{i,j}^{(k-1)}, \end{cases} \tag{1}$$

where each task's local updates get fused periodically. As the number of tasks grows and their data distributions differ (Non-IID across tasks and devices), properly balancing computation and communication becomes much more complex [11], [12].

Multi-Task Federated Learning GPU Deployment: Modern FL training heavily relies on GPUs as the primary computational units. When multiple tasks are deployed on a single GPU, each task's model is compiled into a data flow graph (DFG) within the deep learning framework, comprising various operators (e.g., convolution, ReLU, batch norm) arranged

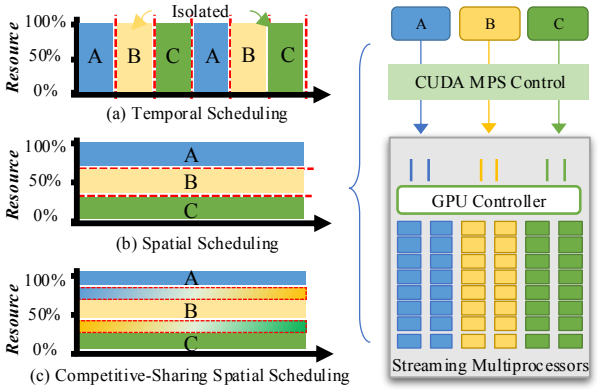


Fig. 3: Temporal and Spatial GPU Scheduling Schemes

by layer. At runtime, each DFG is typically wrapped into a separate process or stream and dispatched to the physical GPU's streaming multiprocessors (SMs) [20], as shown schematically in Fig. 2. Parallel execution occurs if different processes can run simultaneously on separate or overlapping subsets of SMs. However, naive resource partitioning often leads to either underutilization (if SMs are exclusively isolated) or severe contention (if operators from different tasks interfere heavily).

Multi-Task Resource Allocation and Scheduling on GPU: As deploying the aforementioned multi-task FL into individual devices, the major computing unit – GPU is facing a complex scheduling issue to host multiple training models. Currently, there are two major resource allocation and scheduling schemes: *Temporal scheduling* in Fig. 3 (a) isolates GPU resources into sequential time slices for individual tasks (e.g., round-robin [21]). Each task takes the whole GPU resource in its time slice without interfering with others [22]. *Spatial scheduling* in Fig. 3 (b) processes multiple GPU tasks in parallel by assigning a sub-set of SMs to individual tasks as independent processing threads [23]. Latest GPU scheduling technologies are still within these schemes, such as NVIDIA Multi-Instance GPU (MIG) [24], and Multi-Process Service (MPS) [20]. While these technologies provide basic mechanisms for resource partitioning, they often lack the flexibility to adapt to the dynamic workloads characteristic of multi-task FL, leading to under-utilization.

Although spatial scheduling is more preferred by its parallelism, there occurs *competitive resource sharing* between tasks. It is used to describe the complex interactions between concurrent tasks within a single GPU, such as the resource contention between similar computing operators from different training models, or the resource underutilization caused by insufficient resource allocation. However, it is still a newly emerging design consideration. Therefore, how to demystify it to achieve optimal resource allocation and bring it into a large-scale FL is our major research motivation.

B. *Performance and Existing Approaches in Multi-Task FL*

Multi-task FL introduces additional complexity compared to traditional single-model FL, posing unique challenges both in local training and global coordination. Below, we briefly review relevant works from three perspectives—model-oriented, resource-oriented, and coordination-oriented—highlighting their focus and limitations in tackling multi-task concurrency.

First, model-oriented studies often emphasize personalization or multi-task model structures. Approaches such as MOCHA [11] and variational multi-task FL [25] specifically focus on situations where each device may handle multiple related tasks simultaneously. They typically improve convergence by learning shared representations or latent factors across tasks while preserving task-specific parameters. Other works [18], [19], [26], [27] leverage model compression or pruning to reduce communication overhead. Although valuable, these methods generally assume each device's internal scheduling is a black box, leaving operator-level concurrency on the GPU under-explored.

Next, resource-oriented studies focus on hardware and runtime optimizations, such as GPU sharing primitives (MPS, MIG) or dynamic scheduling frameworks [28], [29]. While they enable parallel executions of multiple DNN training jobs, they often target either large-scale data-center workloads or single-task edge devices. FL-centric research on device selection or adaptive local epochs [30], [31] also seeks to mitigate straggling, but seldom deals with multi-task resource contention at the operator or sub-model level.

Finally, coordination-oriented studies address how FL aggregates or synchronizes training among distributed devices. Classical synchronous FL (FedAvg [32]) can stall if one device's multi-task training is slower, while asynchronous FL [16] partially alleviates waiting but may degrade accuracy due to out-of-sync parameters. Hierarchical or clustered FL [17], [33] groups devices with similar data or hardware capabilities to reduce waiting overhead. Nonetheless, these coordination methods typically abstract each device as a single-task participant, overlooking the fact that multiple tasks can be interleaved locally on a single GPU.

In sum, these works provide valuable insights into different aspects of FL (model structures, hardware resource usage, or global scheduling), yet few address the fundamental problem of running multiple tasks together on a single GPU while aligning that concurrency with the larger FL workflow. Our work aims to bridge this gap by jointly optimizing multi-task GPU scheduling and federated coordination, ensuring each device can effectively handle parallel tasks and seamlessly participate in global aggregation.

C. *Design Motivation*

We illustrate the high-level design of our framework in Fig. 4, where each sub-figure (a), (b), and (c) highlights a different facet of our multi-task FL approach:

1) ***Intra-Device Resource Allocation***: As shown in Fig. 4(a), model heterogeneity and task concurrency can cause significant GPU underutilization due to *competitive resource sharing*. To address this, we introduce a *virtual resource allocation* mechanism to precisely model operator-level resource interactions. By optimizing virtual resource budgets for each task, we achieve maximum GPU throughput, significantly improving local resource efficiency over naive spatial or temporal schemes.

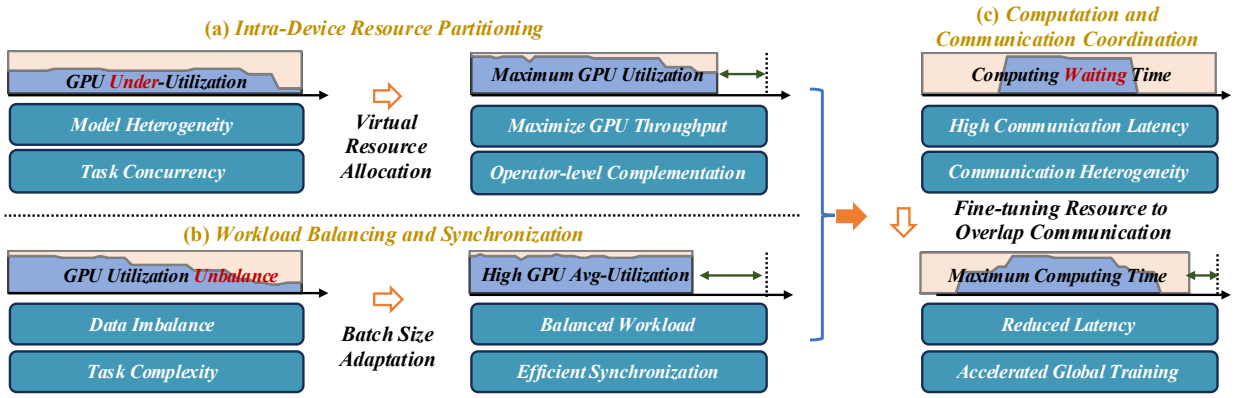


Fig. 4: **Comprehensive Optimization in Multi-Task Federated Learning (FedMT)**: This figure presents the key strategies in the FedMT framework: **(a) Intra-Device Resource Partitioning:** Virtual resource allocation improves GPU utilization and training throughput. **(b) Workload Balancing:** Balances workloads by aligning batch sizes with task data volumes. **(c) Computation and Communication Coordination:** Reduces synchronization delays by overlapping communication with computation. These components collectively enhance performance across multi-task federated learning scenarios.

2) ***Batch Size Adaptation for Workload Balancing***: As illustrated by Fig. 4(b), data imbalance and task complexity heterogeneity across tasks can lead to inefficient GPU utilization and unbalanced synchronization progress. To overcome this, we propose a *batch size adaptation* mechanism that dynamically adjusts each task's batch size according to its data volume. This ensures that tasks with larger datasets are allocated proportionally larger batches, maintaining a balanced workload and improving synchronization efficiency.
3) ***Computation and Communication Overlap***: Fig. 4(c) addresses inefficiencies caused by high communication latency during parameter uploads and downloads. Beyond optimizing local resource usage, we *overlap communication with computation* by scheduling smaller or faster models to communicate during other tasks' compute phases. This intelligent scheduling strategy eliminates idle "waiting slots," reduces latency, and accelerates global FL convergence.

Together, these three components form a cohesive solution to multi-task federated learning. First, we focus on analyzing and maximizing intra-device throughput via virtual resource modeling. Next, we detail inter-device coordination strategies—namely batch size adaptation and communication overlap to address system-wide imbalances. Finally, we validate our design with extensive experiments, demonstrating notable gains in both local efficiency and global convergence rates.

III. Competitive Resource Sharing in Multi-Task Federated Learning on GPU

In this section, we analyze the competitive resource sharing mechanism that arises when multiple DNNs are co-trained on a single GPU under a multi-task FL paradigm. Our goal is twofold: **(1)** Identify the performance bottlenecks introduced by concurrent tasks' competition for limited GPU resources; **(2)** Propose a "virtual resource" concept to effectively manage and coordinate this competitive sharing.

A. Competitive Resource Sharing Analysis

When multiple tasks run in parallel on a single GPU, we must decide how to allocate SMs and other shared resources among them. As illustrated in Fig. 3, two broad scheduling strategies exist: temporal (each task owns the entire GPU in turn) and spatial (tasks simultaneously share SMs). While spatial scheduling can boost utilization via parallelism, it also introduces potential resource contention.

Baseline Example of Spatial Scheduling: In Fig. 5 ①, we show a baseline "fully-isolated" assignment in which MPS [20] statically divides SMs among tasks (e.g., each model gets a distinct subset of SMs). Small operators often do not fully utilize their assigned SM subsets, leading to underutilization; large operators may occupy their allocated SMs for a longer time yet cannot leverage others' idle SMs.

Spatial Resource Sharing: One improvement is to overlap resource assignments, letting tasks share certain SM groups dynamically (Fig. 5②). This approach can raise GPU occupancy by matching complementary workloads—for instance, a compute-intensive operator from Task A could run concurrently with a memory-bound operator from Task B. However, the scheduling of these overlapping SMs is intricate, as different operators contend for the same resources, potentially degrading each other's performance.

Excessive Contention Overhead: As we increase the shared resources by making more overlapped resources assignments in Fig. 5 ③, task models compete for resources more fiercely, leading to resource competition and considerable contention overhead. This is the major research focus of our analysis.

Extreme Contention Kills Parallelism: The resource contention issue is not only causing overhead. Here, when large operators exist, it is hard to achieve complementary resource assignment and thus the scheduling mechanism is pushed back to temporal scheduling as shown in Fig. 5 ④. Therefore, in addition to complementary operators, special attention is also required for such cases, which further increases the analysis complexity of competitive resource sharing.

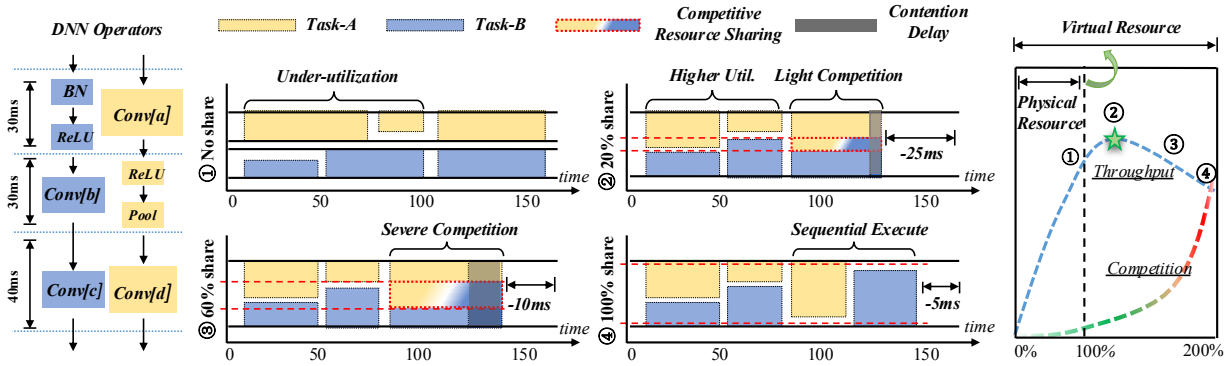


Fig. 5: Illustration of operator-level concurrency under different resource-sharing schemes: (1) fully isolated, (2) partial overlap, (3) excessive overlap leading to contention and (4) fully sharing.

B. *Competitive Resource Sharing Coordination with Virtual Resource*

Motivated by the above findings, we introduce a novel virtual resource concept to manage GPU allocations among multiple co-running tasks. Rather than naively assigning each task a fixed percentage of SMs or forcing them all to share 100% of SMs equally, we propose a scheme where the sum of assigned resources can exceed 100%—up to a limit we call virtual resource R_v.

Definition of Virtual Resource: As Fig. 5 (right side) illustrates, suppose there are I tasks sharing the GPU. The physical SM capacity is considered 100%. In a "virtual resource" framework, we allow the total assigned resource to range up to $I \times 100\%$. For instance, with $I = 2$ tasks, R_v could be as high as 200%. This does not imply we create extra SMs, but rather that each task can compete for the same pool of SMs to a certain fraction. The key is to find an optimal R_v balancing concurrency and contention.

Empirical Analysis of Virtual Resource: To showcase how R_v affects performance, we co-run three neural network models (DensNet121, VGG16, MobileNetV3) under different assignment configurations. We vary R_v from 100% (fully-isolated) to 300% (fully-shared across 3 tasks). As different models can have different stand-alone training speeds, we use two metrics to evaluate the performance: raw throughput P and fairness throughput P_f. The latter is computed as:

$$P_f = \sum_{i=1}^{n} \frac{P_i}{P_{b\_i}}, \tag{2}$$

where n is the number of co-running DNN models, P_i is the i^{th} model's co-running throughput, and P_{b_i} is the stand-alone throughput of the same model when running alone. By such normalization, we treat each model's speedup or slowdown with fair importance.

As shown in Fig. 6, the best performance emerges around $R_v \approx 200\%$, indicating a beneficial degree of overlap that yields high utilization without intolerable contention. By contrast, near 300% we see excessive slowdown due to operator collisions, while at 100% the GPU is underutilized by smaller operators. Hence, moderate sharing—i.e., **competitive resource sharing**—consistently outperforms both extremes.

Coordination via Virtual Resource: In practice, to exploit this idea, we assign each task i a "virtual resource budget" R_i such that $\sum_i R_i = R_v$. When $R_v > 100\%$, tasks overlap on some SM sets, but each still has a target fraction. We can systematically tune R_i based on task FLOPs, batch size, and memory intensity, aiming to maintain a sweet spot of concurrency. This provides a flexible knob to handle multi-task GPU concurrency: 1) If tasks are complementary (e.g., one heavily compute-bound, another memory-bound), a larger R_v can yield higher overall utilization. 2) If tasks are uniformly compute-intensive, we keep R_v closer to left side to avoid severe contention. *FedMT* avoids excessive contention by limiting the total virtual GPU allocation. While tasks are allowed to share GPU resources competitively, the algorithm detects when contention causes diminishing returns. In such cases, *FedMT* moderates the concurrency level (e.g., reducing a task's resource share) so that the GPU scheduler does not revert to pure time-slicing.

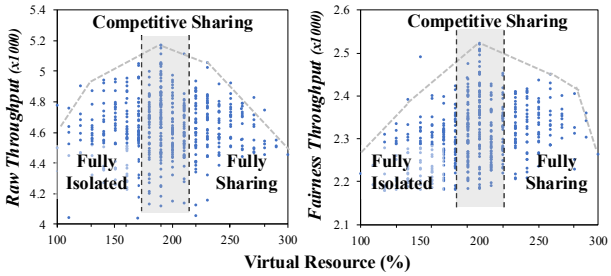


Fig. 6: Throughput analysis with different virtual resource values (R_v) from 100% (fully-isolated) to 300% (fully-shared). Left: raw throughput P. Right: fairness throughput P_f (Eq. 2).

By evaluating different levels of resource overlap, we demonstrate that there is an optimal competitive sharing region where throughput is maximized. This virtual resource perspective allows fine-grained control over concurrency, surpassing naive partitioning strategies. In the following sections, we will leverage this concept to ensure that both intra-device scheduling and inter-device coordination can benefit from enhanced parallelism.

IV. Intra-Device GPU Scheduling with Competitive Resource Sharing

In the previous section, we established that virtual resource (R_v) allocation can effectively enhance concurrency among

multiple DNN tasks on a single GPU. However, in practice, one must determine **(1)** how to assign each task's virtual resource fraction R_i, and **(2)** how to choose the per-task batch size B_i to accommodate potentially heterogeneous data scales and model complexities. This section proposes a modeling-based approach that predicts the multi-task throughput under varied $\{B_i, R_i\}$ settings and thus guides us toward near-optimal resource scheduling.

Problem Definition: Suppose we have n tasks $\{1, \ldots, n\}$ to be executed concurrently on one single GPU. Each task i is described by:

$$(f_i, m_i), \quad \text{and variables } (B_i, R_i).$$

1) f_i, m_i are static features, representing the FLOPs and memory intensity of task i. They can be derived from the DNN model architecture (e.g., ResNet vs. MobileNet).
2) B_i is the batch size, which we can adjust to alter the workload per iteration.
3) R_i is the allocated "virtual resource fraction" for task i, subject to $\sum_{i=1}^{n} R_i = R_v$. Typically, $R_v \leq n \times 100\%$.

We denote $\mathcal{O}_i$ as the workload for task i:

$$\mathcal{O}_i \;=\; \alpha\, f_i\, B_i \;+\; \beta\, m_i\, B_i, \tag{3}$$

where α, β are coefficients reflecting how compute-bound or memory-bound we expect tasks to be. Instead of manually fixing α, β, we propose to learn them from profiling data (detailed in the modeling-fitting phase). Our ultimate goal is to assign $\{B_i\}$ and $\{R_i\}$ so as to maximize the aggregated throughput:

$$\max \;\; \sum_{i=1}^{n} P_i \quad \text{s.t.} \;\; \sum_{i=1}^{n} R_i = R_v, \;\; B_i \in \mathcal{B}_i^{\text{valid}},$$

where P_i denotes the actual runtime throughput (images/sec) of task i under the given (B_i, R_i).

Multi-Input Multi-Output Predictive Model: A key challenge is that P_i depends not only on $\mathcal{O}_i$ and $\{B_i, R_i\}$ itself, but also on *interactions* with other tasks' resource usage. To capture this, we adopt a multi-input multi-output function:

$$(P_1, \ldots, P_n) \;=\; \mathrm{Pre}\Big((f_1, m_1, B_1, R_1), \ldots, (f_n, m_n, B_n, R_n)\Big), \tag{4}$$

where all tasks' (f_i, m_i, B_i, R_i) are fed into a unified predictor $\mathrm{Pre}(\cdot)$. The output is the per-task throughput $\{P_1, \ldots, P_n\}$ when these n tasks compete on the same GPU. This captures sublinear scaling, memory contention, and other concurrency effects.

Data-Driven Model Fitting: To build Pre, we conduct offline profiling on representative DNN models (e.g., VGG16, ResNet50, MobileNetV3, etc.):

1) Random/Grid Sampling of $\{B_i, R_i\}$: We sample diverse configurations $\{(B_i, R_i)\}$ for each task, combining them into multi-task scenarios with up to n concurrent tasks.
2) Collect Throughput: In each scenario, we measure $\{P_1, \ldots, P_n\}$ on a real GPU, thus recording concurrency behavior for varied resource shares.
3) Regression to Learn Pre: We form a dataset $\big\{\big((f_i, m_i, B_i, R_i)_{\text{all tasks}}, (P_1, \ldots, P_n)\big)\big\}$ and train a multi-output regressor. One may use a neural net, polynomial, or log-based model.
4) Fitting α, β in Eq. (3): Alongside throughput regression, we also handle α, β by fitting them to partial concurrency or single-task measurements. We express

$$\mathrm{time}_i \approx \gamma_0 + \gamma_1 \cdot (\alpha\, f_i\, B_i + \beta\, m_i\, B_i),$$

then solve via least squares for α, β. This yields a data-driven ratio of "FLOPs cost" vs. "memory cost" in the workload expression.

After training, Pre can quickly predict each task's throughput under any new combination $\{B_i, R_i\}$. This eliminates the need to brute-force on real hardware for every scenario.

Solving for Optimal Batch Size and Virtual Resource: Given the learned model Pre, we then tackle:

$$\max_{\{B_i, R_i\}} \; \sum_{i=1}^{n} P_i\Big((f_1, m_1, B_1, R_1), \ldots, (f_n, m_n, B_n, R_n)\Big), \tag{5}$$

subject to $\sum_i R_i = R_v$ and feasible $B_i \in \mathcal{B}_i^{\text{valid}}$.

There are two types of maximum throughput:

1) Theoretical maximum: If $\{B_i, R_i\}$ can be any allowed values (e.g., batch size up to 512, resource fraction up to $n \times 100\%$), we find an upper bound that may ignore synchronization constraints.
2) Max throughput under given batch setting: If we fix or partially constrain $\{B_i\}$ (e.g., all tasks must use the same batch, or certain tasks cannot exceed batch 64), then we only optimize $\{R_i\}$ for concurrency within that batch arrangement.

For each discrete batch-size combo and resource fraction increments, we evaluate $\sum_i P_i$ via Pre. We pick the best solution as the optimal results. This approach is much faster than real-hardware trial-and-error, as Pre infers concurrency outcomes offline.

Example: Profiling-based Schedules: In Fig. 7, we illustrate how, across eight different multi-task settings (each with certain $\{f_i, m_i\}$ and candidate batch sizes), our method searches around 125 random or grid-sampled $\{B_i, R_i\}$ schedules. We then highlight a Pareto front of top solutions. Notably, our final modeling-based solution lands in the top 3–15% among all candidates, demonstrating near-optimal resource assignments. Meanwhile, the "best single solution" may correspond to an extreme batch-size choice that is infeasible in a later FL synchronization step, so we can easily re-run the search under a restricted batch range to respect those constraints.

Overall, this modeling-based approach clarifies both the "theoretical maximum throughput" *(unconstrained $\{B_i\}$) and the "maximum throughput under certain batch constraints". By systematically searching over $\{B_i\}$ and $\{R_i\}$ with a learned multi-task concurrency predictor* Pre*, we obtain near-optimal concurrency decisions for multi-task FL. In the subsequent sections, we will integrate these solutions with higher-level FL synchronization and workload-balancing policies.*

V. Inter-Device Multi-Task Coordination

Having optimized local GPU concurrency in the previous section, we now turn to the broader challenge of inter-

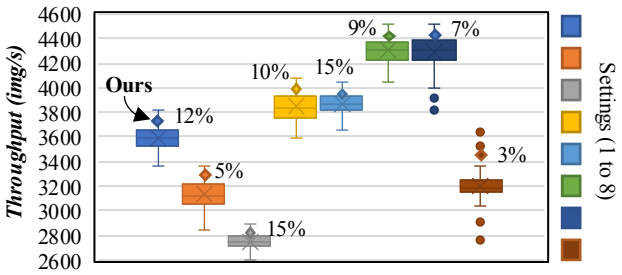


Fig. 7: Our modeling-based solutions across eight different settings yield top 3-15% training throughput among all randomly searched schedules (125 schedules for each setting).

device coordination in multi-task FL. Even if each device runs multiple tasks efficiently, the global training loop can still experience slowdowns when devices are out of sync or tasks finish their local phases at vastly different times. To address these issues, our scheme integrates: **(1)** A batch size adaptation method that refines each device's per-task batch sizes to better balance workloads across multiple tasks. **(2)** A communication-aware scheduling extension that overlaps compute and communication phases, thereby reducing idle periods and improving overall efficiency.

A. *Multi-Task Coordination by Batch Size Adaptation*

Motivation and Basic Idea: Although each device can achieve locally optimal throughput for a particular batch configuration, multiple tasks on the same device may still be imbalanced in terms of data volume or model size. A task with a relatively large dataset but a small batch size risks underutilizing the GPU in crucial moments; conversely, a task with minimal data but an overly large batch could monopolize resources unnecessarily.

Such imbalances lead to idle time or suboptimal progress, especially during the global synchronization phases, as shown in Fig. 8(a). To mitigate them, we define a simple mismatch metric to keep tasks' batch sizes proportional to their data volumes, as shown in Fig. 8(b). For instance, if Task A has double the data of Task B, we target $B_A \approx 2 \times B_B$ to balance their local progress.

Problem Formulation: Let D_i be the data volume for task i, and B_i be its batch size. To formalize these two objectives—mismatch minimization and local throughput maximization—we adopt the following formulation:

$$\begin{cases} \textbf{Objective 1:} \ \min \sum_a \sum_b \left(\frac{|D_a|}{|D_b|} - \frac{B_a}{B_b} \right), \\ \textbf{Objective 2:} \ \max \sum_i P_i, \end{cases} \tag{6}$$

where the first objective measures how well batch sizes match the ratio of data volumes, and the second objective evaluates the overall local throughput $\sum_i P_i$ once batch sizes are decided.

Specifically, for each candidate $\{B_i\}$, we compute the device's maximum achievable throughput by applying our modeling-based resource allocation approach from Section IV (i.e., solving Eq. 5 for $\{R_i\}$). By balancing batch sizes among co-located tasks, we ensure no single task finishes drastically earlier or later, thus preventing synchronization delays at the global level.

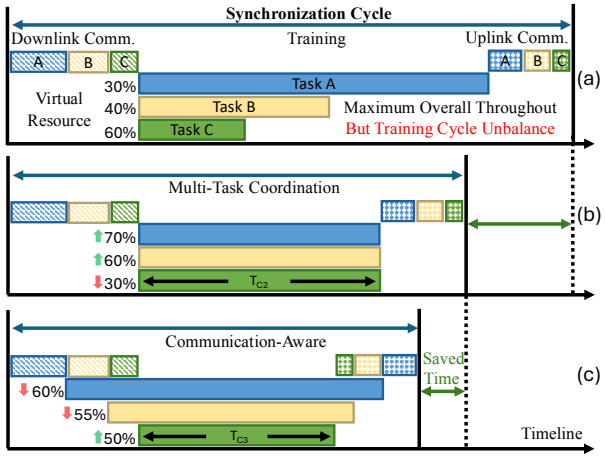


Fig. 8: Aligning tasks' batch sizes with their respective data volumes and overlapping the communication time help reduce idle GPU time and shortens global synchronization.

Algorithmic Sketch: We use the following algorithm to solve the optimization of the Eq. 6.

1) Search Over Batch Sizes: Enumerate feasible B_i for each task. Calculate the mismatch measure (*Objective 1*) to filter out poorly balanced configurations.
2) Local Scheduling: For each promising batch setting, run the intra-device scheduling (Eq. 5) to obtain the maximum local throughput (*Objective 2*).
3) Best Combination: Choose the batch-resource configuration that yields minimal mismatch and highest local throughput.

By aligning each task's batch size with its data volume, we avoid major imbalances that would otherwise slow global synchronization or waste GPU cycles. For highly variable task loads, FedMT could incorporate a damping mechanism for batch size adjustment. Instead of instantaneously responding to a spike or drop in one task's workload, the scheduler updates batch sizes gradually, using a smoothed estimate of each task's load. This prevents oscillations and ensures fairness even if task loads fluctuate violently from round to round.

B. *Communication-Aware Multi-Task Coordination by Overlap Communication Time*

While the batch-size adaptation in Eq. 6 helps balance local computation, it does not explicitly account for the communication phase. In realistic multi-task FL scenarios, each task i must upload (model/gradient) and download (updated parameters) after local training. Efficiently overlapping computation and communication can further improve device-level utilization during full synchronization cycles, as shown in Fig. 8(b) and (c).

For each task i, let

$$T_i^{\text{compute}} = \frac{D_i}{P_i} \quad \text{and} \quad T_i^{\text{comm}} = T_i^{\text{up}} + T_i^{\text{down}},$$

where D_i is the local data volume, P_i is the per-task throughput (determined by batch size and resource allocation), and $T_i^{\text{up}}, T_i^{\text{down}}$ are uploading/downloading times. A task can only start its upload after finishing its compute; similarly, the

download may happen in parallel with other tasks' compute if resources allow.

Basic Overlap Idea: Using Workload-based Priority: A straightforward strategy is to rank tasks by their local training workload (i.e., O_i as discussed in Section IV), then let the largest workload start first. Suppose we have three tasks A, B, C with $O_A \geq O_B \geq O_C$. Concretely: (1) Task A begins downloading its weights and then proceeds with local compute. (2) Task B starts second. (3) Task C, having the smallest workload, starts last.

Because C has minimal compute, it often finishes early and can upload first, overlapping its communication T_C^{comm} with A or B's remaining compute time. Meanwhile, A—the heaviest compute—uploads last, but that does not necessarily stall other smaller tasks, as shown in Fig. 8 (c).

This simple heuristic often demonstrates why communication overlap helps: smaller tasks' communication naturally "hides" behind the larger tasks' compute windows. However, It only considers compute volume. Tasks with large model size but small local data could still become suboptimal if forced to start first (thus incurring a big download immediately), or if they finish compute too quickly but require extensive communication afterward.

Extending to Flexible Priority for Broader Overlap: In real deployments, workload-based priority is not always optimal, as in the following two cases, a task may have:

1) Small compute but large communication (e.g., big model weights but few local samples), making an early compute start suboptimal if it forces a long idle while the task uploads.
2) Both large compute and large communication, where placing it first might cause a heavy download that blocks other tasks from computing in the very beginning.

Hence, beyond simple "largest-workload-first," we propose a flexible scheduling priority that considers both compute time T_i^{compute} and communication T_i^{comm}.

Formulation and Constraints: Let $T_i^{\text{compute}} = \frac{D_i}{P_i}$ represent local computation time (given batch size B_i, resource R_i), and $T_i^{\text{comm}} = T_i^{\text{up}} + T_i^{\text{down}}$ represent upload/download overhead. We fix a permutation π of tasks $\{1, \ldots, N\}$ to define the start order. T_i^{compute} starts after T_i^{down}, and T_i^{up} starts after T_i^{compute}.

Algorithmic: Algorithm 1 illustrates a combined approach:

1) Initial Batch Sizes: Start with the batch configuration $\{B_i^*\}$ that minimizes mismatch and maximizes throughput (Section V-A).
2) Enumerate Priority Orders: Generate permutations π for tasks $\{1, \ldots, N\}$.
3) Local Search around $\{B_i^*\}$: For each candidate order π, we try slight adjustments to each B_i (e.g., $\pm 20\%$) to see if partial scaling helps overlap communication further.
4) Simulate Overlap: Compute each task's T_i^{compute} and T_i^{comm}, then place tasks in order π, letting them share GPU resources.
5) Compute Makespan: Arrange according to the order π and the limitations between computation and communication. We record the final "makespan."
6) Select Best: Pick π and $\{B_i\}$ with minimal total finishing time.

Algorithm 1 Flexible Priority Scheduling.

Input:
- N tasks; local data volumes $\{D_i\}$;
- Initial batch sizes $\{B_i^*\}$ from Section V-A;
- Communication times $(T_i^{\text{up}}, T_i^{\text{down}})$ per task;
- A small search factor δ (e.g., 20%).

Output: Optimal $(\pi, \{B_i\})$ for minimal total finishing time.
$\Pi \leftarrow$ *All permutations of* $\{1, \ldots, N\}$.
bestTime $\leftarrow \infty$; bestConfig $\leftarrow \varnothing$
for $\pi \in \Pi$ **do**
  $\mathcal{B}_i \leftarrow \{(1-\delta)B_i^*,\ B_i^*,\ (1+\delta)B_i^*\}$ for each i
  for each combination $\{b_i\}$ in $\prod_i \mathcal{B}_i$ **do**
    Compute $T_i^{\text{compute}} = \frac{D_i}{P_i(b_i)}$; $T_i^{\text{comm}} = T_i^{\text{up}} + T_i^{\text{down}}$
    SimulateOverlap$(\pi, \{T_i^{\text{compute}}, T_i^{\text{comm}}\}) \rightarrow$ *makespan*
    if *makespan* $<$ bestTime **then**
      bestTime $\leftarrow$ *makespan*
      bestConfig $\leftarrow (\pi, \{b_i\})$
return bestConfig

In summary, the flexible-priority approach explicitly accounts for tasks that might have imbalanced compute vs. communication patterns. This helps ensure long communication windows are "hidden" behind another task's compute, thereby reducing idle periods. Through this method, *FedMT* overlaps each task's communication with other tasks' computation without cutting short any task's local training. If one task finishes its epoch earlier than others, it immediately begins transmitting its update while the remaining tasks continue GPU computation. Moreover, *FedMT's* adaptive batch sizing brings task completion times closer together in subsequent rounds, preventing prolonged idle waits and ensuring that overlap never results in suboptimal scheduling.

By jointly considering both compute and communication overhead, we improve multi-task FL's inter-device synchronization. In the following experiments, we show that this overlap-based scheduling can further speed up convergence, especially when tasks differ markedly in model size or local data volumes.

VI. Experimental Evaluation

A. Experimental Setup

Training Setup: To thoroughly assess our proposed framework, we construct several multi-task FL scenarios involving a diverse set of DNN architectures: AlexNet (Alex), VGG16 (V16), ResNet18 (R18), ResNet50 (R50), ResNet101 (R101), MobileNetV3 (M3), ShuffleNetV2 (S2), and DensNet121 (D121). These models cover a wide spectrum of computational and memory demands.

We evaluate each multi-task setting on the CIFAR10 and CIFAR100 datasets (image resolution $32 \times 32 \times 3$) and run experiments on three different GPUs (NVIDIA Titan V, 3080, and 3090), ensuring our conclusions hold across varying hardware generations. For the sake of evaluating the scheduling improvements in a controlled manner, we assumed an IID data partition across clients for each task. This allowed us to focus on runtime performance metrics without the additional variable of accuracy differences due to non-IID data.

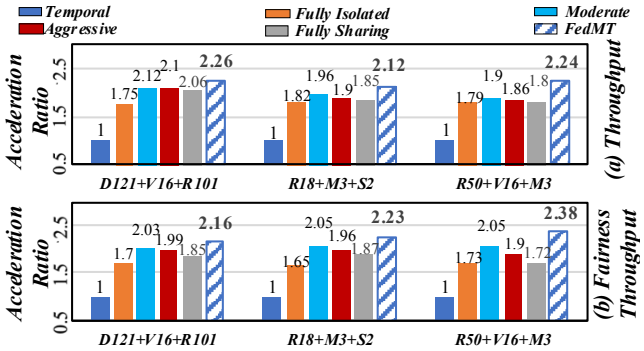


Fig. 9: Intra-Device Multi-Task GPU Throughput (Titan V)

Baseline Methods: To contextualize our results, we compare *FedMT* against resource scheduling baselines:

- *Temporal (T):* The default CUDA scheduling, training tasks sequentially on the full GPU (one task at a time). Many real-world deployments without advanced schedulers effectively behave temporally, making this a practical baseline for sequential execution.
- *Fully-Isolated (FI):* Each task is statically allocated a dedicated subset of GPU resources (e.g., via NVIDIA MPS partitioning) throughout training. This represents commercially available spatial scheduling with strong isolation and no operator-level overlap.
- *Fully-Sharing (FS):* All tasks share the GPU concurrently using MPS with no enforced isolation. This enables fine-grained parallelism but may cause resource contention due to uncoordinated execution.

Metrics: We mainly report:

- *Raw Throughput* (P): Per-model training speeds (images/second), indicating overall system throughput.
- *Fairness Throughput* (P_f): Each model's throughput normalized by its standalone speed, then summed across tasks. This treats speed on each model equally.
- *GPU Utilization*: SM occupancy to highlight how effectively each scheme uses GPU resources.

Unless otherwise noted, we normalize all throughput values against the temporal baseline to highlight relative speedups.

B. Intra-Device Speed-Up Evaluation

In this section, we demonstrate how our intra-device resource partitioning approach accelerates multi-task training on a single GPU. Specifically, we compare our *FedMT* method against three baselines (Temporal, Fully-Isolated, Fully-Sharing), measuring both total throughput and fairness throughput.

Experimental Scenarios: Fig. 9 presents results from three contrasting multi-task settings: (1) D121 + V16 + R101: A heavy scenario with large computational volumes and memory footprints. (2) R18 + M3 + S2: A lightweight scenario representative of mobile/edge tasks. (3) R50 + V16 + M3: A hybrid scenario with mixed operator latencies and resource demands. All tasks use a fixed batch size of 128 for simplicity. In addition, we add two baseline scenarios: 1) Moderate static sharing–Each task is statically assigned 60% of GPU resources, resulting in a moderate overlap (20%) between tasks; 2) Aggressive static sharing–Each task is statically assigned 80% of GPU resources, leading to higher overlap (60%) and potentially significant resource contention.

Overall Speed-up: Across all three settings, our reallocation strategy achieves a consistent **2.16× to 2.38×** speed-up compared to the temporal (sequential) training baseline. Compared to the static spatial baseline (Fully-Isolated and Fully-Sharing), *FedMT* achieves higher GPU utilization; if one task finishes early or has a lighter workload, *FedMT* reallocates its unused GPU time to other tasks, whereas in static partitioning that portion of the GPU would remain idle. As a result, *FedMT* improves training throughput by 20–40% over the best static partition setting in our experiments, in addition to vastly outperforming the temporal schedule. The outcomes also confirm observations consistent with those depicted in Fig. 6: Moderate exhibits reduced GPU idleness compared to fully isolated scenarios but still suffers from inefficiencies when tasks vary in computational demands over time. Aggressive, on the other hand, often leads to excessive resource contention, diminishing throughput.

Speed-up in Fairness Throughput: Our method also provides higher fairness throughput—reflecting that smaller models (e.g., M3, S2) receive adequate SM resources rather than being bottlenecked by larger models (e.g., R50, R101). Fully-Sharing, by contrast, can allow large models to monopolize SM usage, leaving small models starving. Hence, by enabling complementary resource sharing and informed concurrency (recall Section IV), *FedMT* improves both the total throughput and the per-model balance of progress.

C. Multi-Task Coordination by Batch Size Adaptation

In addition to optimizing concurrency within a single GPU (Section IV), our method also coordinates multiple tasks at the inter-device level by refining each task's batch size to match its data volume and capacity (detailed in Section V-A). This ensures that the global training loop remains efficient across devices, preventing bottlenecks or prolonged idle times.

Experimental Setup: We simulate an FL system with multiple devices, each co-training exactly three DNN tasks (e.g., VGG16, R18, M3), but in imbalanced data-volume proportions of 1:2:3. For all baseline approaches, we fix a uniform batch size of 128 to highlight the effects of naive scheduling. By contrast, our scheme jointly tunes batch sizes in proportion to each task's data scale, then applies the modeling-based resource allocation from Section IV to finalize $\{R_i\}$.

Results: Figure 10 shows the average throughput (top) and fairness throughput (bottom) across all devices within a single FL synchronization cycle. We observe that our approach yields a significant **2.53×–2.80×** acceleration relative to the baselines, outperforming both simpler resource partitions and single-batch-size strategies.

These results confirm that aligning batch sizes to each task's local data volume (Section V-A) is crucial when co-training multiple heterogeneous DNNs in an FL system. Not only does this strategy maintain higher local GPU utilization, but it also prevents global synchronization delays by ensuring tasks

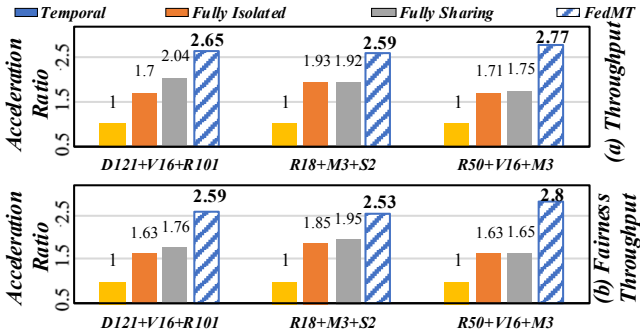


Fig. 10: Average throughput across multiple devices within one FL synchronization cycle (Titan V)

progress more evenly. Overall, we see a clear synergy between batch-size adaptation and competitive resource sharing, leading to enhanced performance across multiple devices.

D. *Verifying the Effect of Communication Overlap*

While our proposed FedMT incorporates multiple components, here we focus on isolating the communication overlap strategy (discussed in Section V-B) to evaluate its standalone impact. Specifically, we compare scenarios with and without overlapping upload/download phases under otherwise identical conditions, highlighting how concurrency in communication can reduce idle time and improve efficiency.

Experimental Setup: We select three multi-task settings in which each device co-trains three DNN models (from the models in Training Setup) on a single GPU (3090). For each model, we assume the following parameter sizes: DensNet121: $\sim$ 30 MB, VGG16: $\sim$ 55 MB, ResNet101: $\sim$ 170 MB, ResNet50: $\sim$ 100 MB, ResNet18: $\sim$ 45 MB, MobileNetV3: $\sim$ 15 MB, ShuffleNetV2: $\sim$ 9 MB. Since each FL iteration typically involves uploading local updates and downloading aggregated parameters, we emulate a realistic edge bandwidth of 400 Mbps (uplink and downlink) for all experiments.

Regarding GPU resource management, we employ: (i) GPU Resource Partition with our modeling-based approach from Section IV to ensure fair baselines, (ii) Batch Size Adaptation from Section V-A to maintain proportional batch sizes across tasks. We only vary whether communication overlap is activated:

- Overlap-Off: Each task computes locally, but all uploads/downloads occur sequentially (i.e., one task's communication must complete before another's begins).
- Overlap-On: Our flexible scheduling ensures a smaller task's communication phase can be overlapped with a larger task's computation (see Section V-B).

Metrics and Results: We measure both the average local iteration time (seconds) and the overall throughput (images/second). As summarized in Table I, enabling overlap (Overlap-On) yields appreciable gains over no overlap (Overlap-Off):

Across all tested configurations, Overlap-On reduces local iteration time by 13–19%. Notably, the scenario with D121+V16+R101 has a relatively large communication overhead, so overlapping communication with compute yields the most pronounced speedup (+19.4%). Overall, these results validate the effectiveness of the communication overlap component, demonstrating that upload/download concurrency is especially advantageous for heavier models and multi-task environments.

TABLE I: Ablation on Communication Overlap: Local Iteration Time (s) / Throughput (img/s)

Setting	Overlap-Off	Overlap-On	Speedup
D121+V16+R101	20.40s / 3.10K	17.08s / 3.70K	+19.4%
R18+M3+S2	8.62s / 5.73K	7.61s / 6.49K	+13.3%
R50+V16+M3	14.40s / 4.21K	12.27s / 4.94K	+17.3%

TABLE II: Search Overhead for Flexible Priority with 3 Tasks

Item	Value
Permutation Count	6
Batch Variation per Task	3
Total Search Points	162
Avg. Prediction Time per Point	1.8 ms
Total Search Time	0.29 s

E. *Cost of Flexible Priority Search*

Recall that our flexible priority scheduling mechanism (Algorithm 1) systematically searches all permutations and minor batch-size variations to discover an optimal communication overlap schedule. We now quantify the overhead of this search procedure and show that it is minimal—on the order of a few hundred milliseconds—compared to the lengthier training cycles in typical multi-task FL.

Experimental Setup: For this study, we pick a 3-task scenario $\{VGG16, R18, M3\}$. The total number of possible permutations is $3! = 6$. We then allow $\{\pm 20\%\}$ batch-size scaling for each model, thus creating $3^3 = 27$ micro-variations for each permutation. In total, the space of search points is $6 \times 27 = 162$. For each configuration, we compute a predicted makespan via our learned concurrency model (see Section IV).

Results:

As summarized in Table II, we measured FedMT's scheduling overhead (under 0.3s) to be minor (approximately 2–3% of the total training time per round). This overhead is negligible in light of the 2× or higher training throughput improvements FedMT achieves, and it is often amortized by overlapping scheduling computations with GPU execution. Moreover, even for scenarios with 4–5 concurrent tasks (thereby increasing the number of permutations), the total search time remains below one second in our implementation. These findings confirm that the search procedure is both feasible and efficient, ensuring that the scheduling overhead does not offset the performance gains from flexible priority overlap.

F. *Scheduling Visualization with Nsight*

To illustrate how each model executes at the operator level under different resource-sharing strategies, we profile the combination of D121, V16, and M3 using the NVIDIA Nsight tool. The primary goal of this visualization is to offer a clearer, more intuitive, and realistic view of runtime behaviors under different resource-sharing schemes (Spatial-Isolated, Moderate Sharing—190%, Excessive Competition,

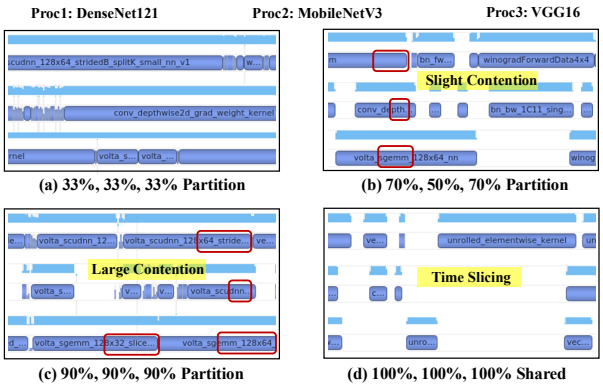


Fig. 11: Operator-level scheduling visualization via NVIDIA Nsight, showing the GPU kernel timelines of three models.

and Fully Sharing) as initially discussed conceptually in Fig.5. Unlike the quantitative metrics (e.g., throughput and fairness throughput) shown earlier, Fig.11 qualitatively illustrates GPU kernel execution timelines, clearly highlighting differences in operator-level parallelism, contention, and GPU resource utilization. This visual evidence complements and substantiates the numerical evaluations, providing deeper insight into the practical implications and runtime performance characteristics of the proposed FedMT strategy.

As shown in Fig. 11, each scheduling configuration corresponds to a distinct row, displaying the kernel execution timeline of three streams (one per model):

- Spatial-Isolated: Each model is allocated a disjoint subset of SMs. Although simpler to manage, certain operators remain underutilized, prolonging model latency.
- Moderate Sharing – 190%: By letting the sum of allocations exceed 100%, streams can overlap selectively, improving concurrency while limiting contention.
- Excessive Competition: Over-allocating virtual resources leads to increased contention. Even smaller models (e.g., M3) experience greater operator delays.
- Fully Sharing: Tasks share the entire GPU dynamically, often reverting to time-sliced usage when contention is high, thereby undermining parallelism.

From these traces, we observe that moderate or balanced resource sharing strikes a better trade-off between concurrency and contention. In contrast, naive fully shared or heavily overcommitted allocations can degrade performance for both large and small models.

G. *Resource Utilization Assessment*

To evaluate how effectively *FedMT* utilizes GPU resources, we collect both compute and memory usage during training and conduct a detailed analysis of operator-level execution patterns under parallelization. Specifically, we leverage the NVIDIA Nsight Profiler to measure the achieved SM Occupancy—an indicator of how many warps are actively running on the GPU's streaming multiprocessors (SMs).

Fig. 12 displays the SM occupancy over time (milliseconds) for four different scheduling approaches when running the multi-task combination {R101, D121, M3}. From the occupancy timeline, we make the following observations:

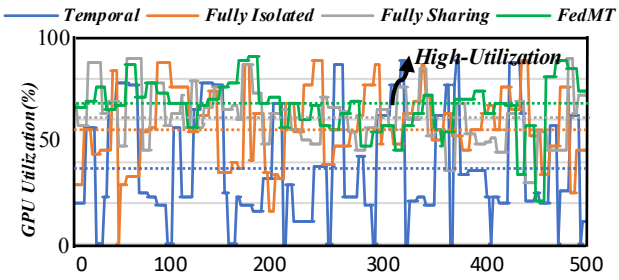


Fig. 12: GPU SM Occupancy

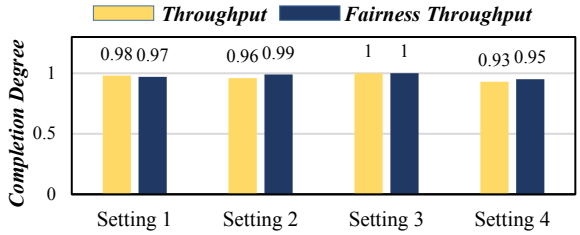


Fig. 13: Performance Comparison

TABLE III: Resource Allocation in Varied Multi-Task Settings

Setting	R50	V16	**Setting**	R50	V16	M3	D121
①	128	32	②	128	128	128	128
Best	80%	70%	Best	70%	80%	60%	70%
Ours	80%	80%	Ours	70%	90%	60%	80%
Setting	R50	M3	**Setting**	R50	V16	M3	D121
③	128	512	④	32	32	32	32
Best	80%	80%	Best	60%	80%	50%	70%
Ours	80%	80%	Ours	70%	90%	70%	70%

- Temporal: Each model sees exclusive GPU access during its slot, but idle gaps commonly appear between task switches.
- Fully-Isolated: If a large operator is scheduled alone on its subset of SMs, that subset can briefly reach high occupancy. However, such events are relatively scarce, and smaller operators underutilize those SMs for much of the time.
- Fully-Sharing: Parallel execution can boost utilization but also causes contention when big operators collide, leading to suboptimal occupancy in critical intervals.
- FedMT: By adaptively partitioning SMs according to "virtual resource" fractions, *FedMT* mitigates severe contention while still permitting beneficial overlaps, sustaining higher average occupancy over time.

Hence, these results confirm that *FedMT* outperforms naive scheduling methods in fully exploiting the GPU's parallel execution capabilities, leading to higher throughput and improved consistency across co-running tasks.

H. *Generality Performance*

We now evaluate the broader applicability of our modeling-based resource allocation beyond the specific use cases discussed earlier. In particular, we seek to verify whether the proposed method can consistently approach near-optimal solutions under a variety of model and batch-size configurations.

TABLE IV: Scalability Evaluation: Throughput (img/s) for Different Task Combinations (Titan V)

Models	Combination	Temporal	Fully Isolated	Fully Sharing	*FedMT*
2×	V16 + R34	1.75K	2.12K (1.21x)	2.28K (1.30x)	**2.80K (1.60x)**
	R18 + R34	1.77K	2.19K (1.24x)	2.25K (1.27x)	**2.75K (1.55x)**
	R34 + R50	1.40K	1.82K (1.30x)	1.83K (1.31x)	**2.33K (1.66x)**
	R50 + R101	1.20K	1.43K (1.19x)	1.43K (1.19x)	**1.73K (1.44x)**
4×	V16 + R18 + M3 + D121	1.78K	3.60K (2.02x)	3.95K (2.22x)	**4.91K (2.75x)**
	M3 + R34 + R50 + R101	1.69K	3.52K (2.08x)	3.80K (2.25x)	**4.64K (2.74x)**
5×	V16 + R18 + R34 + R50 + D121	1.73K	3.51K (2.03x)	3.78K (2.18x)	**4.93K (2.84x)**

TABLE V: GPU Generality Evaluation (img/s). T: Temporal; FI: Fully Isolated; FS: Fully Sharing.

Models	T-3080	T-3090	FI-3080	FI-3090	FS-3080	FS-3090	FedMT-3080	FedMT-3090
ALEX+V16+R18	1.94K	2.98K	3.20K (1.65x)	5.00K (1.68x)	3.30K (1.70x)	5.16K (1.73x)	4.01K (**2.07x**)	6.32K (**2.12x**)
D121+V16+R34	1.56K	2.34K	2.81K (1.80x)	4.17K (1.78x)	2.82K (1.81x)	4.17K (1.78x)	3.47K (**2.22x**)	5.43K (**2.32x**)
R50+V16+M3	1.76K	2.70K	3.15K (1.79x)	4.73K (1.75x)	3.17K (1.80x)	4.83K (1.79x)	3.97K (**2.25x**)	5.98K (**2.21x**)
R101+D121+M3	1.46K	2.25K	2.48K (1.70x)	3.80K (1.69x)	2.60K (1.78x)	4.10K (1.82x)	3.35K (**2.30x**)	5.10K (**2.27x**)
R34+R50+R101	1.42K	2.19K	2.39K (1.68x)	3.61K (1.65x)	2.56K (1.80x)	3.88K (1.77x)	2.98K (**2.10x**)	4.70K (**2.15x**)

Table III lists four representative multi-task settings involving either 2 or 4 concurrent DNN models. Each setting indicates the targeted batch sizes (e.g., 128 for R50, 32 for V16) and compares two resource allocation strategies:

- Best: The top-performing configuration found via a (potentially exhaustive) search, serving as a reference baseline.
- Ours: The allocations yielded by our modeling-based approach without manual fine-tuning.

We also define a completion degree metric, which measures the ratio of our method's throughput (T or T_f) to that of the best solution. A value close to 1.0 indicates near-optimal performance.

Fig. 13 provides a graphical summary of these results. Across all four settings, our approach attains a Completion Degree consistently above 0.9, demonstrating that it closely tracks the best-known solution. Notably, in Setting ③, our allocations match the "Best" configuration exactly (i.e., a Completion Degree of 1.0).

These findings validate the generality and robustness of our modeling-based scheduling approach. Even when facing different DNN models and batch sizes (spanning from large models like R50 to lighter ones like M3), it can systematically discover high-quality resource partitions that approach or achieve the best possible throughput. Consequently, we expect the same methodology to generalize to additional tasks or system constraints with minimal tuning effort.

I. Scalability Evaluation

A key question for multi-task FL is whether our proposed *FedMT* framework continues to scale effectively as the number of concurrently trained tasks grows. To investigate this, we evaluate *FedMT* under increasingly larger task sets on a single NVIDIA Titan V GPU, with configurations of 2×, 4×, and 5× simultaneous tasks (see Table IV for specific combinations).

From Table IV, two key observations emerge:

- Throughput and Fairness Gains: Compared to both the Temporal (sequential) and conventional MPS-based (Fully Isolated, Fully Sharing) baselines, *FedMT* consistently achieves higher overall throughput. Fairness throughput also improves, indicating a more balanced resource distribution across tasks.
- Stronger Acceleration at Larger Scales: As the number of concurrent tasks increases (e.g., from 2× to 5×), the relative speedup of *FedMT* grows accordingly. This highlights the framework's ability to exploit more opportunities for parallelism and complementary resource usage under heavier workloads.

Overall, these results demonstrate robust scalability in *FedMT*: even when multiple DNNs run together, our adaptive resource-sharing and modeling-based scheduling can effectively mitigate contention. Hence, *FedMT* remains well-suited for federated learning scenarios featuring multiple tasks with heterogeneous resource demands.

J. Generality across Different GPUs

Although the above results are primarily based on a Titan V GPU, practical deployments involve a variety of hardware platforms. To assess how *FedMT* performs across different GPUs, we repeat multi-task experiments on NVIDIA RTX 3080 and 3090 cards. Table V compares four scheduling strategies — *Temporal* (T), *Fully Isolated* (FI), *Fully Sharing* (FS), and *FedMT* — for five representative model combinations.

- FedMT consistently outperforms all baselines: On both 3080 and 3090, *FedMT* provides substantial throughput gains, often doubling naive sequential performance.
- Speedups remain stable across hardware generations: Even though the 3090 boasts higher raw compute capability, *FedMT* maintains a clear advantage in relative speedups, highlighting that our method scales well when GPU capacity increases.
- Competitive resource sharing remains crucial: Both Fully Isolated and Fully Sharing can suffer from either underutilization or contention, whereas *FedMT* splits GPU resources to capitalize on concurrency benefits.

Overall, these findings reinforce the versatility of *FedMT*. Whether the device is a mid-tier 3080 or a more powerful 3090, the proposed modeling-based competitive sharing approach consistently yields robust improvements over conventional GPU scheduling paradigms.

VII. Conclusion

As the demand for multi-task Federated Learning (FL) continues to grow, driven by its ability to accommodate heterogeneous and complex learning objectives, this work introduces a comprehensive optimization framework, *FedMT*. By bridging intra-device GPU scheduling and inter-device FL coordination, we address critical challenges in resource contention, workload balancing, and synchronization efficiency.

At the intra-device level, our proposed competitive resource-sharing mechanism utilizes a novel "virtual resource" abstraction to achieve optimal GPU utilization, balancing concurrency and contention dynamically. At the inter-device level, we integrate batch size adaptation with communication-aware scheduling to minimize idle periods and improve global synchronization. These innovations significantly enhance the system's throughput and resource efficiency.

Experimental results validate the effectiveness of *FedMT*, demonstrating substantial improvements in both intra- and inter-device performance. Specifically, compared to the standard temporal (sequential) scheduling baseline, we observed a 2.16×–2.38× increase in intra-device GPU training throughput and a 2.53×–2.80× boost in inter-device FL coordination efficiency across diverse multi-task scenarios. Further comparisons with state-of-the-art spatial scheduling approaches (such as NVIDIA MPS-based Fully-Isolated and Fully-Sharing schemes) confirm that *FedMT* consistently provides substantial gains (up to 1.4×–1.7× improvement) even against these baselines, demonstrating the practical efficiency of our proposed scheduling strategies.

Future work could naturally extend *FedMT* to multiple GPUs per client. Tasks may be mapped to different GPUs to minimize contention (e.g., several tasks per GPU when feasible), and our competitive sharing approach can manage any GPUs running multiple tasks. This would further improve training throughput, as tasks benefit from both inter-GPU parallelism and intra-GPU optimization.

References

[1] T. Li *et al.*, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, no. 3, pp. 50–60, 2020.

[2] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: challenges and applications," *International Journal of Machine Learning and Cybernetics*, vol. 14, no. 2, pp. 513–535, 2023.

[3] S. Banabilah, M. Aloqaily, E. Alsayed, N. Malik, and Y. Jararweh, "Federated learning review: Fundamentals, enabling technologies, and future applications," *Information processing & management*, vol. 59, no. 6, p. 103061, 2022.

[4] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, and A. H. Celdrán, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys & Tutorials*, 2023.

[5] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz *et al.*, "Self-driving cars: A survey," *Expert systems with applications*, vol. 165, p. 113816, 2021.

[6] J. Mills, J. Hu, and G. Min, "Multi-task federated learning for personalised deep neural networks in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 630–641, 2021.

[7] L. Che, J. Wang, Y. Zhou, and F. Ma, "Multimodal federated learning: A survey," *Sensors*, vol. 23, no. 15, p. 6986, 2023.

[8] Z.-A. Huang, Y. Hu, R. Liu, X. Xue, Z. Zhu, L. Song, and K. C. Tan, "Federated multi-task learning for joint diagnosis of multiple mental disorders on mri scans," *IEEE Transactions on Biomedical Engineering*, vol. 70, no. 4, pp. 1137–1149, 2022.

[9] J. Pan, X. Lin, J. Xu, Y. Chen, and C. Zhuo, "Lithography hotspot detection based on heterogeneous federated learning with local adaptation and feature selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[10] J. Xia, T. Liu, Z. Ling, T. Wang, X. Fu, and M. Chen, "Pervasivefl: Pervasive federated learning for heterogeneous iot systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4100–4111, 2022.

[11] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[12] F. Yu *et al.*, "Fed2: Feature-aligned federated learning," in *Proceedings of the 27th ACM KDD*, 2021, pp. 2066–2074.

[13] C.-H. Wang *et al.*, "Heterogeneous federated learning through multi-branch network," in *Proceedings of the IEEE ICME*. IEEE, 2021.

[14] Z. Tang, J. Huang, R. Yan, Y. Wang, Z. Tang, S. Shi, A. C. Zhou, and X. Chu, "Bandwidth-aware and overlap-weighted compression for communication-efficient federated learning," in *Proceedings of the 53rd International Conference on Parallel Processing*, 2024, pp. 866–875.

[15] Z. Xu *et al.*, "Helios: heterogeneity-aware federated learning with dynamically balanced collaboration," in *Proceedings of the 58th ACM/IEEE DAC*. IEEE, 2021, pp. 997–1002.

[16] Y. Chen *et al.*, "Asynchronous online federated learning for edge devices with non-iid data," in *Proceedings of the 2020 IEEE Big Data*. IEEE, 2020, pp. 15–24.

[17] M. Xie *et al.*, "Multi-center federated learning," *arXiv preprint arXiv:2108.08647*, 2021.

[18] Y. Jiang *et al.*, "Model pruning enables efficient federated learning on edge devices," *arXiv preprint arXiv:1909.12326*, 2019.

[19] Y. Lin *et al.*, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint:1712.01887*, 2017.

[20] Nvidia, "Multi-Process Service," 2021. [Online]. Available: https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[21] R. V. Rasmussen*et al.*, "Round robin scheduling–a survey," *European Journal of Operational Research*, vol. 188, no. 3, pp. 617–636, 2008.

[22] S. Kato *et al.*, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *Proceedings of the USENIX ATC*, 2011, pp. 17–30.

[23] Y. Liang *et al.*, "Efficient gpu spatial-temporal multitasking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 748–760, 2014.

[24] N. Inc., "NVIDIA Multi-Instance GPU User Guide," 2021. [Online]. Available: https://docs.nvidia.com/datacenter/tesla/mig-user-guide/

[25] L. Corinzia and J. M. Buhmann, "Variational federated multi-task learning," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[26] Z. Chen, C. Jia, M. Hu, X. Xie, A. Li, and M. Chen, "Flexfl: Heterogeneous federated learning via apoz-guided flexible pruning in uncertain scenarios," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4069–4080, 2024.

[27] D. Wu, W. Yang, H. Jin, X. Zou, W. Xia, and B. Fang, "Fedcomp: A federated learning compression framework for resource-constrained edge computing devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[28] Y. Yu, Y. Shao, Y. Chen, and Y. Chen, "Salus: Fine-grained gpu sharing primitives for deep learning applications," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–15.

[29] W. Xiao *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 595–610.

[30] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[31] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

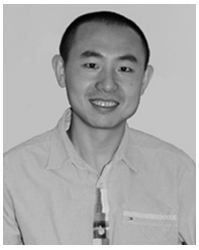
[32] B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the AISTATS*. PMLR, 2017, pp. 1273–1282.

[33] M. S. H. Abad *et al.*, "Hierarchical federated learning across heterogeneous cellular networks," in *Proceedings of the ICASSP IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2020, pp. 8866–8870.

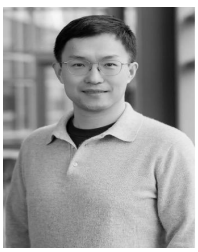
Yongbo Yu received the B.S. degree from Ocean University of China, Qingdao, China, in 2017, and M.S. degree from Ocean University of China, Qingdao, China, in 2020. He is currently pursuing his Ph.D. degree in the Department of Electrical and Computer Engineering at George Mason University under the supervision of Prof. Zhi Tian. His current research interests include Graphic Computing Optimization for Deep Learning, Multi-Tenant AI Computing on GPU, and Federated Learning.

Fuxun Yu received the B.S. degree from the Harbin Institute of Technology, Harbin, China, in 2017, and obtained the Ph.D. degree from the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA, in 2022, under the supervision of Prof. X. Chen. He is currently a Principal Research Manager with Microsoft, Redmond, WA, USA. His current research interests include high-performance deep neural network computing, full-stack DNN computing optimization, deep learning security, interpretability, and explainability of deep learning.

Zirui Xu received the B.S. and M.S. degrees from Beijing Jiaotong University, Beijing, China, in 2014 and 2017, respectively, and obtained the Ph.D. degree from the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA, in 2022, under the supervision of Prof. X. Chen. He is currently a Senior ML Research Scientist at Microsoft. His current research interests include high-performance mobile computing systems and mobile intelligent application robustness and security.



Di Wang received the B.E. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2005, the M.S. degree in computer systems engineering from the Technical University of Denmark, Kongens Lyngby, Denmark, in 2008, and the Ph.D. degree in computer science and engineering from The Pennsylvania State University, University Park, PA, USA, in 2014. He is currently a Principal Research Manager with Microsoft, Redmond, WA, USA. His research spans the areas of artificial intelligence, computer systems, computer architecture, and energy-efficient system design and management. Dr. Wang received five best paper awards and two best paper nominations.



Minjia Zhang (Member, IEEE) is an Assistant Professor at the Department of Computer Science, University of Illinois Urbana-Champaign. His research focuses on efficient machine learning systems, effective algorithms, and large-scale AI applications. Before joining the University of Illinois Urbana-Champaign, he completed his Ph.D. in the Computer Science Department at Ohio State University in May 2016, where he worked on building efficient and scalable systems with strong semantics for parallel programs.

Ang Li (Member, IEEE) received the Ph.D. degree from Duke University, Durham, NC, USA, in 2022, under the supervision of Prof. Y. Chen. He is a tenure-track Assistant Professor with the Department of Electrical and Computer Engineering, University of Maryland at College Park, College Park, MD, USA. His research interests lie in the intersection of machine learning and edge computing, with a focus on building large-scale networked and trustworthy intelligent systems to solve practical problems in a collaborative, scalable and secure.

ChenChen Liu (Member, IEEE) received the M.S. degree from Peking University, Beijing, China, in 2013, and the Ph.D. degree from the ECE Department, University of Pittsburgh, Pittsburgh, PA, USA, in 2017. She is currently an Assistant Professor with the Department of Computer Science and Electrical Engineering, University of Maryland at Baltimore County, Baltimore, MD, USA. Her current research interests include brain-inspired computing systems, machine learning, and emerging nonvolatile memory technologies.

Zhi Tian (Fellow, IEEE) is a Professor with the Electrical and Computer Engineering Department, George Mason University, since 2015. Prior to that, she was a faculty member at Michigan Technological University from 2000 to 2014. She served as a Program Director with the U.S. National Science Foundation from 2012 to 2014. Her research interest lies in the areas of statistical signal processing, wireless communications, and distributed machine learning. Her current research focuses on distributed network optimization and learning, wireless spectrum sensing, and millimeter-wave systems. She received the IEEE Communications Society TCCN Publication Award in 2018. She served as an Associate Editor for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS and the IEEE TRANSACTIONS ON SIGNAL PROCESSING. She was a General Co-Chair of the 2016 IEEE GlobalSIP Conference and the 2023 IEEE SPAWC Workshop. She was a Member-at-Large of the Board of Governors of the IEEE Signal Processing Society from 2019 to 2021. She was an IEEE Distinguished Lecturer for both the IEEE Communications Society and the IEEE Vehicular Technology Society.

Xiang Chen (Member, IEEE) received the bachelor's degree from Northeastern University, Shenyang, China, in 2010, and the M.S. and Ph.D. degrees in computer engineering under the supervision of Dr. Y. Chen from the University of Pittsburgh, Pittsburgh, PA, USA, in 2012 and 2016, respectively. In 2023, he joined Peking University, Beijing, China, where he is a tenure-track Associate Professor with the Department of Computer Science and Technology, School of Computer Science. After his Ph.D. graduation in 2016, he directly joined George Mason University, Fairfax, VA, USA, where he led seven National Science Foundation (NSF) projects. His research focuses on mobile and distributed computing systems, high-performance intelligence computing, and related edge, IoT, and CPS applications. Dr. Chen achieved the NSF CAREER Award.