Approximation Algorithms for the Maximum Multi-Commodity Service DAG Flow

Peng Li and Jianan Zhang

***Abstract*—Emerging computation-intensive services can be decomposed into subtasks whose dependencies are modeled as directed acyclic graphs (DAGs). To support such services, the maximum multi-commodity service DAG flow (MMCF-DAG) problem aims to compute routing paths and processing locations to maximize the total flow under fixed network communication and computation resources. We develop a $(1-\epsilon)$-approximation algorithm UVMP-MWU with slice-wise polynomial (XP) time complexity, based on an exact algorithm for the unconstrained valid mapping problem (UVMP) and multiplicative weight updates. Moreover, we apply column generation and develop UVMP-CG, which achieves the same performance guarantee and executes much faster in practice. To further reduce runtime, we apply heuristics for UVMP as subroutines in both UVMP-MWU and UVMP-CG. Simulations show that the algorithms substantially reduce runtime compared to the linear programming approach and achieve higher throughput for AI workflows.**

I. Introduction

The proliferation of artificial intelligence (AI), particularly large language models (LLMs), is reshaping network demands. Emerging services, including AI training and inference, as well as embodied AI, impose intensive communication and computation resource requirements. The end-to-end performance of these services relies on the joint availability of link bandwidth and node processing capacity. Consequently, operators need to determine the maximum service demand that can be served given fixed communication and computation resources.

Accurately determining this capacity limit is challenging, as emerging workloads exhibit increasingly complex structures. Although service function chains are well-studied, they fail to capture the complex fork–join patterns [1], [2] inherent in modern pipelines. Prominent examples include Mixture-of-Experts (MoE) architectures [3], AI workflows [4], [5], and deep learning computation graphs [6]. In these applications, data flows can be split into parallel branches for processing (e.g., token dispatching to experts in MoE architectures) and subsequently aggregated (e.g., token combining after the expert computation). Capturing these complex structures necessitates a model capable of representing fork–join dependencies.

The authors are with the State Key Laboratory of Photonics and Communications, School of Electronics, Peking University, Beijing 100871, China. *(Corresponding author: Jianan Zhang)*

This work was supported in part by the National Natural Science Foundation of China under Grant 62301011, in part by the Beijing Natural Science Foundation under Grant L257016, and in part by the Fundamental Research Funds for the Central Universities, Peking University.

ISBN 978-3-903176-82-9 © 2026 IFIP

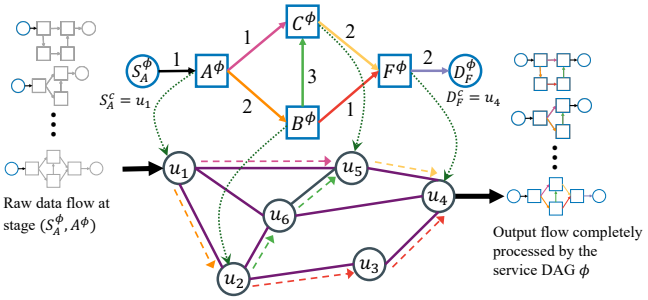

Fig. 1. Illustration of a service DAG and its execution over a distributed computing network. The top depicts DAG dependencies, and the bottom depicts a joint function placement and flow routing in the network.

To this end, we adopt a *service directed acyclic graph (service DAG)* model, wherein nodes represent service functions and edges represent execution dependencies. The service DAG model generalizes the service function chain model, enabling accurate characterization of fork–join services. As illustrated in Fig. 1, serving a service DAG requires jointly placing functions and routing flows to satisfy both resource constraints and DAG-structured dependencies.

Building on the service DAG model, we study the *maximum multi-commodity service DAG flow (MMCF-DAG)* problem, which maximizes total admitted demand by jointly optimizing function placement and flow routing under fixed network communication and computation resources. Analogous to the classical maximum flow in communication networks, MMCF-DAG quantifies the maximum throughput of service DAGs via fractional flow splitting and resource allocation.

A. Related Work

1) Maximum flow for computing networks: Several works have studied the maximum flow problem under the service function chain model. Charikar *et al.* [7] proposed a $(1-\epsilon)$-approximation algorithm for the maximum service function chain flow problem. Kuo *et al.* [8] utilized randomized rounding and heuristics to obtain polynomial-time near-optimal solutions for the maximum service function chain flow problem. When function placement is fixed, Sallam *et al.* [9] proposed a Ford–Fulkerson-based approach that reduces the problem to repeated shortest path computations under service function chain constraints.

Beyond service function chains, Lin *et al.* [10] studied network edge deployment of services with DAG dependencies, aiming to maximize the throughput. They formulated an in-

teger linear program and developed approximation algorithms based on randomized rounding.

2) Virtual network embedding (VNE): The VNE problem maps virtual graphs onto physical infrastructures [11], [12]. Standard formulations typically associate each request with a fixed demand and aim to maximize the acceptance ratio or the profit of accepted requests subject to these demands. Under these formulations, VNE is NP-complete and inapproximable [13].

3) Device placement for deep learning computation graphs: Device placement maps a deep learning computation graph (a DAG) onto heterogeneous devices to minimize execution time or energy [6], [14]–[16]. Most prior work uses reinforcement learning or heuristics to optimize the placement of a single DAG instance.

Despite the extensive literature, a critical gap remains. First, prior studies on device placement typically optimize a single DAG instance and thus ignore contention among concurrent flows. Second, in multi-commodity settings, existing works exhibit distinct limitations. Approaches such as NEST [10] restrict solutions to discrete embeddings and treat each request as indivisible. By contrast, although VNE formulations may allow fractional flow splitting, they assume fixed demands and aim to maximize the acceptance ratio or profit subject to these demands [12]. Both paradigms preclude characterizing the network capacity region. To the best of our knowledge, no prior work studies the maximum multi-commodity service DAG flow problem in which flow volumes are continuous decision variables constrained only by physical capacities.

B. Contribution

Solving the MMCF-DAG problem is nontrivial due to the inherent complexity of service DAGs. First, DAG dependencies make the decision space grow exponentially in the sizes of the service DAG and the physical network, since each function and dependency may admit multiple processing and routing realizations. Second, multi-input and multi-output functions require flow splitting and merging, so the classical flow conservation law at each node does not apply. Consequently, a new methodology is needed to model service DAG processing and characterize the network capacity region.

To address these challenges, we formulate a linear program (LP) that captures function processing, flow routing, and traffic scaling of service DAGs. To avoid exhaustive enumeration in the exponential decision space, we develop two $(1-\epsilon)$-approximation algorithms: UVMP-MWU and UVMP-CG. The main contributions of this paper are summarized as follows:

- We formulate MMCF-DAG as an LP in which fractional flows are jointly routed and processed. We identify the unconstrained valid mapping problem (UVMP) as the core subproblem and prove its NP-completeness via a reduction from the Multiway Cut problem.
- We develop two approximation algorithms: UVMP-MWU, which leverages multiplicative weight updates to penalize congested resources, and UVMP-CG, which uses column generation to iteratively identify effective service DAG computations.
- We prove that both algorithms achieve $(1-\epsilon)$-approximation to the optimum when utilizing an exact algorithm for UVMP. UVMP-MWU runs in slice-wise polynomial (XP) time parameterized by the service-DAG treewidth, while UVMP-CG converges more efficiently than UVMP-MWU. Simulations on synthetic instances show that both algorithms achieve near-optimal throughput substantially faster than exact LP solvers; for large AI computational DAGs, the heuristic variants are compared empirically with representative domain-specific baselines.

The rest of this paper is organized as follows. Section II introduces the system model and problem definition. Section III formulates MMCF-DAG as an LP for an exact solution. Section IV develops approximation algorithms for MMCF-DAG. Section V presents numerical results. Section VI concludes this paper.

II. System Model

In this section, we introduce the system model and formally define the MMCF-DAG problem.

A. Service DAG Model

Consider a service request with execution dependencies modeled by a DAG (V^ϕ, E^ϕ), where V^ϕ denotes the set of service functions and E^ϕ denotes the set of dependency edges.

To capture the data input and output, we designate the functions handling raw input as *source functions* V_s^ϕ and the functions producing final output as *sink functions* V_d^ϕ. We construct an augmented service DAG by introducing a set of auxiliary *service sources* $S^\phi = \{S_i^\phi \mid i \in V_s^\phi\}$ to feed input data to source functions, and *service sinks* $D^\phi = \{D_i^\phi \mid i \in V_d^\phi\}$ to accept output data from sink functions. For instance, in Fig. 1, A^ϕ and F^ϕ are designated as the source and sink functions, corresponding to the service source S_A^ϕ and service sink D_F^ϕ, respectively. These auxiliary nodes are connected to the core DAG via ingress edges $E_s^\phi = \{(S_i^\phi, i) \mid i \in V_s^\phi\}$ and egress edges $E_d^\phi = \{(i, D_i^\phi) \mid i \in V_d^\phi\}$. Consequently, the complete service DAG is denoted by $\phi = (\bar{V}^\phi, \bar{E}^\phi)$, which includes the original DAG as well as the input and output nodes.

We characterize the flow dynamics using *scaling factors*. Let ξ^{ij} denote the traffic scaling factor for each edge $(i,j) \in \bar{E}^\phi$, including ingress and egress edges, and let r^i denote the computation requirement for function $i \in V^\phi$. The processing logic is defined relative to a *unit of service flow*: to process one unit of flow, function i aggregates input data of volume ξ^{ki} from each immediate predecessor k with $(k,i) \in \bar{E}^\phi$, consumes r^i units of computation resources, and subsequently generates output traffic of volume ξ^{iq} on each outgoing edge $(i,q) \in \bar{E}^\phi$. This mechanism is illustrated in Fig. 1, where function C^ϕ aggregates one unit of flow from A^ϕ and three units from B^ϕ, yielding an output of two units on edge (C^ϕ, F^ϕ). Following this processing logic, the external input and output data volumes per unit of service flow are

determined by $\{\xi^{(S_i^\phi,i)} : i \in V_s^\phi\}$ and $\{\xi^{(i,D_i^\phi)} : i \in V_d^\phi\}$, respectively.

B. Computing Network Model

A computing network is modeled as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. Each node $u \in V$ can forward packets to its neighbors (acting as a router) or host service functions (acting as a computing server). Each link $(u, v) \in E$ represents a connection from node u to node v. Let μ_u denote the computation resources available at node u, and let μ_{uv} denote the communication resources available on link (u, v).

C. Traffic Model

A commodity (c, ϕ) is specified by a set of physical *source nodes* $S^c \subseteq V$, a set of physical *destination nodes* $D^c \subseteq V$, and a service DAG ϕ. There is a one-to-one mapping between the service DAG's auxiliary service source/sink nodes and the commodity's source/destination nodes: each service source $S_A^\phi \in S^\phi$ is anchored to a specific source node $S_A^c \in S^c$, representing that node S_A^c feeds data into the input of source function A^ϕ. Similarly, each service sink $D_A^\phi \in D^\phi$ is anchored to a destination node $D_A^c \in D^c$, representing that node D_A^c receives the output of sink function A^ϕ. For example, in Fig. 1, the assignment $S_A^c = u_1$ indicates that physical node u_1 injects data into function A^ϕ, whereas $D_F^c = u_4$ indicates that node u_4 collects the output of function F^ϕ. It is important to clarify that S^c and D^c are distinct from S^ϕ and D^ϕ. The sets S^c and D^c are source and destination nodes in the computing network, while S^ϕ and D^ϕ refer to service sources and sinks of the service DAG, respectively.

Let $(\mathcal{C}, \Phi)$ denote the set of all commodities. Building on these definitions, we formally define the problem.

Definition 1 (Maximum multi-commodity service DAG flow problem)**.** *Given a computing network G with fixed communication and computation resources $(\{\mu_{uv}, \mu_u\})$, and a set of commodities $(\mathcal{C}, \Phi)$, the maximum multi-commodity service DAG flow (MMCF-DAG) problem determines the routing paths and processing locations for each commodity $(c, \phi) \in (\mathcal{C}, \Phi)$ to maximize the total delivered service flow across G. The solution must ensure that the flow of every commodity is fully processed according to its service DAG.*

Next, we address the MMCF-DAG problem through exact formulation and approximation algorithms in Sections III and IV, respectively.

III. LP-Based Solution for MMCF-DAG

In this section, we introduce the unconstrained valid mapping problem as a core building block and develop an LP formulation for MMCF-DAG.

A. Unconstrained Valid Mapping Problem (UVMP)

The traditional maximum flow problem has two types of LP formulations: flow-based and path-based. The flow-based LP is constructed by enforcing flow conservation at each node, while the path-based LP is constructed by enumerating all paths in the network. The standard flow-based formulation may yield solutions that cannot be decomposed into valid service DAG computations (as detailed in Remark 1). Consequently, we formulate the MMCF-DAG problem in a path-based fashion. To this end, we introduce the concept of an *unconstrained valid mapping*, which generalizes the notion of a "path" for service DAGs.

Definition 2 (Unconstrained valid mapping)**.** *For a commodity (c, ϕ), an unconstrained valid mapping m assigns each service DAG node $i \in \bar{V}^\phi$ to a physical node $m(i) \in V$ and each edge $(i, j) \in \bar{E}^\phi$ to a routing path in G that originates at $m(i)$ and terminates at $m(j)$.*

Each service source or sink in the service DAG is fixed to its corresponding source or destination node, i.e., $m(S_i^\phi) = S_i^c, \forall i \in V_s^\phi$ and $m(D_i^\phi) = D_i^c, \forall i \in V_d^\phi$, according to the traffic model in Section II-C. Additionally, if two adjacent service DAG nodes i and j are mapped to the same node, their dependency edge is mapped to an empty path: $m(i, j) = \emptyset$. An empty dependency mapping indicates that the data transfer from function i to function j is local to a computation node and does not use any communication links. Given a set of resource prices, the UVMP computes the mapping with minimum resource cost.

Definition 3 (Unconstrained valid mapping problem)**.** *Let c_u denote the unit processing cost at node u, and let c_{uv} denote the unit transmission cost over link (u, v). The cost of an unconstrained valid mapping m for commodity (c, ϕ) is defined as* $\text{cost}(m) = \sum_{i \in V^\phi} r^i c_{m(i)} + \sum_{(i,j) \in \bar{E}^\phi} \xi^{ij} \sum_{(u,v) \in m(i,j)} c_{uv}$. *The unconstrained valid mapping problem (UVMP) determines a mapping $\hat{m}$ to minimize the cost* $\text{cost}(\hat{m})$ *for a given commodity (c, ϕ).*

The term "unconstrained" indicates that any function $i \in V^\phi$ can be mapped to any computation node, and data between any adjacent functions $(i, j) \in \bar{E}^\phi$ can be routed through any feasible path. The related valid mapping problem (VMP), studied in [12] and [13], imposes additional mapping constraints (e.g., restricting which computation nodes can host specific functions or which communication links can realize particular function dependencies). This paper studies the maximum flow problem, focusing specifically on the UVMP rather than the constrained VMP. Although relaxing these placement constraints may appear to reduce the problem to a shortest path computation, we prove that UVMP remains NP-complete.

Definition 4 (Decision version of UVMP)**.** *Given a commodity (c, ϕ), a computing network $G = (V, E)$, nonnegative resource costs $\{c_u\}_{u \in V}$ and $\{c_{uv}\}_{(u,v) \in E}$, and a cost threshold W, the decision version of UVMP (UVMP-Dec) asks whether there exists an unconstrained valid mapping whose total cost is at most W.*

Theorem 1. *If $|V| \geq 3$, UVMP-Dec is NP-complete.*

Although the NP-completeness of VMP was established

in [13], that proof does not apply to our unconstrained variant. We therefore prove NP-completeness via a reduction from Multiway Cut. The complete proof is provided in Appendix A.

B. Formulation of MMCF-DAG

Let $\mathcal{M}$ denote the global set of unconstrained valid mappings across all commodities. For each mapping $m_k \in \mathcal{M}$, let (c_k, ϕ_k) identify the specific commodity associated with m_k. Accordingly, let θ_k represent the amount of commodity-(c_k, ϕ_k) flow that is routed and processed according to mapping m_k.

We define functions $N(k,(i,j),(u,v))$ and $P(k,i,u)$ to quantify the resource requirements for each mapping. For a given mapping $m_k \in \mathcal{M}$, $N(k,(i,j),(u,v))$ specifies the communication resources on physical link (u,v) required to support one unit of commodity-(c_k,ϕ_k) flow associated with dependency (i,j), and $P(k,i,u)$ specifies the computation resources at node u required to process one unit of commodity-(c_k,ϕ_k) flow for function i.

$$N\big(k,(i,j),(u,v)\big) = \begin{cases} \xi^{ij}, & \text{if } (u,v) \in m_k(i,j) \\ 0, & \text{otherwise.} \end{cases},$$

$$P\left(k,i,u\right) = \begin{cases} r^i & \text{if } m_k(i) = u \\ 0 & \text{otherwise} \end{cases}.$$

Accordingly, we formulate the MMCF-DAG problem as the linear program below. Throughout the paper, $\sum_k$ denotes summation over all mappings $m_k \in \mathcal{M}$. For a fixed mapping m_k, $\sum_{(i,j)}$ and $\sum_i$ denote summations over all dependency edges $(i,j) \in \bar{E}^{\phi_k}$ and all service functions $i \in V^{\phi_k}$, respectively.

$$\begin{aligned} &\text{(MMCF-DAG)} \quad \max_{\boldsymbol{\theta}} \sum_k \theta_k \\ \text{s.t.} \quad &\sum_k \sum_{(i,j)} \theta_k N\left(k,(i,j),(u,v)\right) \le \mu_{uv}, \forall (u,v) \in E, \\ &\sum_k \sum_i \theta_k P\left(k,i,u\right) \le \mu_u, \forall u \in V, \\ &\theta_k \ge 0, \forall k. \end{aligned}$$

By enumerating unconstrained valid mappings, the formulation generalizes classical network flow to service DAG flows. It captures resource consumption on communication links and computation nodes, and explicitly characterizes the decision space of joint routing and processing, thereby maximizing total service DAG throughput.

However, the number of unconstrained valid mappings grows exponentially in the sizes of the service DAG and the physical network. Moreover, Theorem 1 demonstrates that the underlying UVMP—a generalization of the shortest path problem—is NP-complete. These complexities make exhaustive enumeration computationally prohibitive. Accordingly, the next section develops efficient approximation algorithms that avoid enumerating unconstrained valid mappings.

Remark 1. *In [17], it is proved that solutions to a flow-based LP relaxation of the VNE problem may not be decomposed into valid embeddings. Similarly, we can construct counterexamples (even with acyclic service graphs) where a solution to a flow-based LP cannot be decomposed into a set of service DAG flows. Therefore, we focus on the LP formulation that explicitly enumerates all unconstrained valid mappings to compute the MMCF-DAG flow.*

IV. Approximation Algorithms for MMCF-DAG

In this section, we develop two $(1-\epsilon)$-approximation algorithms for the MMCF-DAG problem: UVMP-MWU and UVMP-CG. The theoretical $(1-\epsilon)$ guarantees rely on an exact algorithm for UVMP (DynVMP [12]), which runs in XP time parameterized by the service-DAG treewidth [12].

A. The UVMP-MWU Algorithm

The UVMP-MWU algorithm builds on the multiplicative weight update (MWU) method [18], a powerful approximation framework for fractional packing and covering linear programs. Existing MWU-based approximations for multi-commodity flow with in-network processing typically assume linear service function chains with a single function [7] and therefore do not capture the fork–join dependencies inherent in service DAGs.

To solve MMCF-DAG, we associate one *expert* with each physical resource to enforce its capacity constraint: for each link (u,v) we define a link expert with weight w_{uv}, and for each node u we define a node expert with weight w_u. Let $\boldsymbol{\theta}$ be a feasible solution to MMCF-DAG. For each expert associated with link (u,v) and node u, define the gain $H_{uv}(\boldsymbol{\theta})$ and $H_u(\boldsymbol{\theta})$ as the corresponding fractional resource consumption under $\boldsymbol{\theta}$.

$$H_{uv}(\boldsymbol{\theta}) = \big(\sum_k \sum_{(i,j)} \theta_k N(k,(i,j),(u,v))\big)/\mu_{uv},$$
$$H_u(\boldsymbol{\theta}) = \big(\sum_k \sum_i \theta_k P(k,i,u)\big)/\mu_u.$$

To mitigate congestion, MWU penalizes heavily utilized resources via multiplicative weight updates: $w_{uv} \leftarrow w_{uv}(1+\epsilon)^{H_{uv}(\boldsymbol{\theta})}, w_u \leftarrow w_u(1+\epsilon)^{H_u(\boldsymbol{\theta})}$. By repeatedly computing a minimum weight solution, i.e., the least congested service DAG realization under the current weights, the algorithm routes flow toward lightly loaded resources. This mechanism balances utilization across links and computation nodes and improves the aggregate service DAG flow.

To identify the minimum-weight solution in each iteration, we employ the DynVMP algorithm [12], which computes the minimum-cost unconstrained valid mapping via tree decomposition and dynamic programming. At iteration t, we set the UVMP unit resource prices to $c_{uv}^{(t)} = w_{uv}^{(t)}/\mu_{uv}$ for each link $(u,v) \in E$ and $c_u^{(t)} = w_u^{(t)}/\mu_u$ for each node $u \in V$. Under the UVMP objective in Definition 3, this yields a link contribution $\xi^{ij} w_{uv}^{(t)}/\mu_{uv}$ for routing dependency (i,j) over link (u,v), and a node contribution $r^i w_u^{(t)}/\mu_u$ for processing function i at node u. DynVMP minimizes the total cost with a parameterized complexity of $O(|V^\phi|^3 |V|^{2\mathrm{tw}(\mathcal{T}_\phi)+3})$, where $\mathrm{tw}(\mathcal{T}_\phi)$ denotes the width of the tree decomposition $\mathcal{T}_\phi$ [12].

Algorithm 1 summarizes the UVMP-MWU algorithm. It initializes all link and node expert weights to a small positive value $\delta \in (0,1)$, chosen as a function of the network size and the target accuracy $0 < \epsilon < 1$ to ensure convergence, as specified below.

$$\delta = e^{\left(\frac{\ln(1+\epsilon)\,\ln(|V|+|E|)+\ln^2(1+\epsilon)}{\epsilon-\epsilon^2-\ln(1+\epsilon)}+\ln(1+\epsilon)\right)}.$$

It then performs three steps at each iteration t: (i) Mapping Selection: DynVMP is invoked once for each commodity $(c,\phi) \in (\mathcal{C},\Phi)$ under unit prices $c_{uv}^{(t)} = w_{uv}^{(t)}/\mu_{uv}$ and $c_u^{(t)} = w_u^{(t)}/\mu_u$. It returns a local minimum-cost mapping $\tilde{m}_{(c,\phi)}$, and the minimum-cost candidate is selected as $\hat{m}^{(t)}$. (ii) Flow Assignment: For $\hat{m}^{(t)}$, the per-unit loads on link (u,v) and node u are $\ell_{uv}(\hat{m}^{(t)}) \triangleq \sum_{(i,j)} \sum_{(u',v')\in\hat{m}^{(t)}(i,j)} \xi^{ij}\mathbb{I}\{(u,v) = (u',v')\}$ and $\ell_u(\hat{m}^{(t)}) \triangleq \sum_i r^i \mathbb{I}\{u = \hat{m}^{(t)}(i)\}$, where the sums are over the selected commodity's dependency edges and service functions. The algorithm sets α_t to the largest admissible flow on $\hat{m}^{(t)}$, limited by the bottleneck resource as in (1), and forms $\mathbf{g}^{(t)}$ by assigning α_t to $\hat{m}^{(t)}$ and zero to all other mappings. (iii) Weight Update: Expert weights are updated multiplicatively according to the resource consumption induced by $\mathbf{g}^{(t)}$. The algorithm stops when any expert weight reaches or exceeds 1; the accumulated flow is then divided by $\log_{1+\epsilon}((1+\epsilon)/\delta)$ to satisfy all capacity constraints.

$$\alpha_t = \min\left\{\min_{(u,v)\in E:\ell_{uv}(\hat{m}^{(t)})>0} \frac{\mu_{uv}}{\ell_{uv}(\hat{m}^{(t)})}, \min_{u\in V:\ell_u(\hat{m}^{(t)})>0} \frac{\mu_u}{\ell_u(\hat{m}^{(t)})}\right\}. \tag{1}$$

Let $f(\boldsymbol{\theta})$ denote the total flow value of a solution $\boldsymbol{\theta}$, i.e., $f(\boldsymbol{\theta}) = \sum_k \theta_k$. Let $\boldsymbol{\theta}^*$ be an optimal solution to MMCF-DAG.

Theorem 2. *Using an exact algorithm for UVMP (e.g., DynVMP) in each iteration, UVMP-MWU guarantees a flow value of at least* $(1-\epsilon)f(\boldsymbol{\theta}^*)$.

We provide a proof sketch here; the complete proof is omitted due to page limitations. Let $\mathcal{R} \triangleq E \cup V$ denote the set of link and node constraints (experts), with weights $\{w_r^{(t)}, \forall r \in \mathcal{R}\}$, and let $\Psi^{(t)} \triangleq \sum_{r\in\mathcal{R}} w_r^{(t)}$. Define the MWU distribution $\mathcal{D}^{(t)}$ by $\Pr[r] = w_r^{(t)}/\Psi^{(t)}$, and define the expected gain as $H(\mathcal{D}^{(t)},\boldsymbol{\theta}) = \sum_{r\in\mathcal{R}} \Pr[r]\, H_r(\boldsymbol{\theta})$. Let $\Delta^{(t)}$ be the minimum mapping cost returned by DynVMP under costs $c_{uv} = w_{uv}^{(t)}/\mu_{uv}$ and $c_u = w_u^{(t)}/\mu_u$. Then $H(\mathcal{D}^{(t)},\mathbf{g}^{(t)}) = \alpha_t\Delta^{(t)}/\Psi^{(t)}$. Since DynVMP returns a minimum-cost mapping, an optimal solution $\boldsymbol{\theta}^*$ satisfies $H(\mathcal{D}^{(t)},\boldsymbol{\theta}^*) \geq (f(\boldsymbol{\theta}^*)\,\Delta^{(t)})/\Psi^{(t)}$, and hence $H(\mathcal{D}^{(t)},\mathbf{g}^{(t)}) \leq \alpha_t/f(\boldsymbol{\theta}^*)$. Applying the standard MWU packing bound [7], [18] together with the stopping condition $\max_{r\in\mathcal{R}} w_r^{(t)} \geq 1$ (equivalently, $\max_{r\in\mathcal{R}} H_r(\boldsymbol{\theta}) \geq \log_{1+\epsilon}(1/\delta)$) yields $f(\boldsymbol{\theta}) \geq (1-\epsilon)\, f(\boldsymbol{\theta}^*)\, \log_{1+\epsilon}\left(\frac{1+\epsilon}{\delta}\right)$. Scaling down by $\log_{1+\epsilon}\left(\frac{1+\epsilon}{\delta}\right)$ enforces feasibility, completing the proof. □

Complexity analysis. In each iteration, the choice of α_t makes at least one used resource a bottleneck, i.e., for some $r \in \mathcal{R}$, $H_r(\mathbf{g}^{(t)}) = 1$, while $0 \leq H_{r'}(\mathbf{g}^{(t)}) \leq 1$ for all resources r'. Before termination, every weight is smaller than 1; hence no resource can serve as a bottleneck

Algorithm 1 UVMP-MWU Algorithm

Set $t = 1$. Initialize global solution $\boldsymbol{\theta} \leftarrow \mathbf{0}$.
Initialize weights $w_{uv}^{(1)} \leftarrow \delta$ for all $(u,v) \in E$ and $w_u^{(1)} \leftarrow \delta$ for all $u \in V$.
while True **do**
  if $\exists (u,v)$ s.t. $w_{uv}^{(t)} \geq 1$ **or** $\exists u$ s.t. $w_u^{(t)} \geq 1$ **then**
    Return scaled solution $\frac{\boldsymbol{\theta}}{\log_{1+\epsilon}((1+\epsilon)/\delta)}$.
  Mapping Selection:
  for each commodity $(c,\phi) \in (\mathcal{C},\Phi)$ **do**
    Compute optimal mapping $\tilde{m}_{(c,\phi)}$ via DynVMP.
  Select $\hat{m}^{(t)} \leftarrow \tilde{m}_{(c^*,\phi^*)}$ where $(c^*,\phi^*) = \arg\min_{(c,\phi)} \mathrm{cost}(\tilde{m}_{(c,\phi)})$.
  Flow Assignment: Compute the maximum admissible flow value α_t on $\hat{m}^{(t)}$ via (1).
  Construct increment vector $\mathbf{g}^{(t)}$ by setting the entry for $\hat{m}^{(t)}$ to α_t and others to 0.
  Update global solution: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{g}^{(t)}$.
  Weight Update:
  for each link $(u,v) \in E$ **do**
    Calculate utilization $H_{uv}(\mathbf{g}^{(t)})$ induced by the increment.
    $w_{uv}^{(t+1)} \leftarrow w_{uv}^{(t)}(1+\epsilon)^{H_{uv}(\mathbf{g}^{(t)})}$.
  for each node $u \in V$ **do**
    Calculate utilization $H_u(\mathbf{g}^{(t)})$ induced by the increment.
    $w_u^{(t+1)} \leftarrow w_u^{(t)}(1+\epsilon)^{H_u(\mathbf{g}^{(t)})}$.
  $t \leftarrow t+1$.

in more than $\ln(1/\delta)/\ln(1+\epsilon)$ iterations. Since there are $|E|+|V|$ resources, UVMP-MWU terminates after at most $(|E|+|V|)\ln(1/\delta)/\ln(1+\epsilon)$ iterations. In each iteration, the algorithm invokes DynVMP for every commodity $(c,\phi) \in (\mathcal{C},\Phi)$ to identify the globally minimum-cost mapping. Thus, the overall complexity is

$$O\left(\frac{(|E|+|V|)\log(|E|+|V|)}{\epsilon^2} \sum_{(c,\phi)\in(\mathcal{C},\Phi)} |V^\phi|^3\, |V|^{2\mathrm{tw}(\mathcal{T}_\phi)+3}\right).$$

Hence, the algorithm runs in XP time: for any fixed treewidth $\mathrm{tw}(\mathcal{T}_\phi)$, the runtime is polynomial in the input size.

B. The UVMP-CG Algorithm

UVMP-CG addresses the exponential dimensionality of MMCF-DAG via column generation. Unlike prior VNE approaches that pursue exact solutions [12], UVMP-CG uses the value of the pricing subproblem to upper-bound the optimality gap, thereby certifying $(1-\epsilon)$-approximation. This approximation enables early termination, reducing the number of iterations in practice while preserving throughput guarantees.

Restricted master problem. UVMP-CG maintains a restricted master problem and a pricing subproblem. The restricted master problem considers only a subset of mappings $\mathcal{M}' \subseteq \mathcal{M}$. By introducing slack variables s_{uv} and s_u for link and node capacity constraints, respectively, the restricted master problem can be written in standard form:

$$\max_{\boldsymbol{x}} \boldsymbol{\beta}^\top \boldsymbol{x} \quad \text{s.t.} \quad \mathbf{A}_\mathrm{r}\boldsymbol{x} = \boldsymbol{\mu}, \quad \boldsymbol{x} \geq 0.$$

where $\boldsymbol{\mu} \in \mathbb{R}^{|E|+|V|}$ denotes the vector of link and node capacities, $\boldsymbol{x}$ contains the mapping variables for $\mathcal{M}'$ and the slack variables, $\mathbf{A}_\mathrm{r}$ is the associated constraint matrix, and $\boldsymbol{\beta}$ is the coefficient vector, with entries equal to 1 for mapping variables and 0 for slack variables. Only the block of mapping variables in $\boldsymbol{x}$ constitutes the MMCF-DAG flow solution; the slack variables are auxiliary variables used to express the capacity constraints in standard form. In each iteration, we solve the restricted master problem to optimality via the simplex method and obtain the current optimal value z. Let $\mathbf{B}$ denote the current basis matrix, i.e., a square submatrix of $\mathbf{A}_\mathrm{r}$. The dual vector associated with the current optimal basis is $\mathbf{p}^\top = \boldsymbol{\beta}_B^\top \mathbf{B}^{-1}$, where $\boldsymbol{\beta}_B$ denotes the coefficient vector of the variables in the current basis. The components p_{uv} and p_u serve as shadow prices for the capacity constraints of link (u, v) and node u, respectively.

Pricing subproblem via DynVMP. The key step of UVMP-CG is to select a new mapping column with the maximum positive reduced cost. For a mapping m_k, let $\mathbf{A}_k \in \mathbb{R}^{|E|+|V|}$ denote its resource consumption column in MMCF-DAG (with entries induced by $N(\cdot)$ and $P(\cdot)$); the reduced cost is $1 - \mathbf{p}^\top \mathbf{A}_k$. Maximizing this reduced cost is equivalent to finding mapping m_k that minimizes $\mathbf{p}^\top \mathbf{A}_k$, i.e., $\min_{k=1,\ldots,|\mathcal{M}|} \sum_{(i,j)} \xi^{ij} \sum_{(u,v)\in m_k(i,j)} p_{uv} + \sum_i r^i p_{m_k(i)}$. This minimization coincides with the UVMP in Definition 3 by setting the resource costs to the current shadow prices, i.e., $c_{uv} \leftarrow p_{uv}$ and $c_u \leftarrow p_u$. We invoke the DynVMP oracle for each commodity $(c, \phi) \in (\mathcal{C}, \Phi)$ to find its local minimum-cost mapping, and then select the mapping with the lowest cost among all commodities. If this global minimum cost is less than 1, the associated column has a positive reduced cost and is added to $\mathcal{M}'$.

Optimality Gap and Termination. Consider the dual of the MMCF-DAG LP: $\min\{\boldsymbol{\mu}^\top \mathbf{y} : \mathbf{A}^\top \mathbf{y} \geq \mathbf{1},\ \mathbf{y} \geq 0\}$, where $\mathbf{A}$ denotes the constraint matrix that includes all mapping variables. The dual vector $\mathbf{p}$ is optimal for the restricted master problem (and thus dual-feasible for the columns in $\mathcal{M}'$), but it may violate the dual constraints corresponding to columns that have not yet been generated. The pricing subproblem identifies the most violated dual constraint (equivalently, the column that minimizes $\mathbf{p}^\top \mathbf{A}_k$), thereby enabling the construction of a feasible dual bound for the full problem.

Let $\gamma \triangleq \min_{k=1,\ldots,|\mathcal{M}|} \mathbf{p}^\top \mathbf{A}_k$ denote the minimum resource cost computed by the pricing oracle. If $\gamma \geq 1$, the maximum reduced cost $1 - \gamma$ is non-positive, implying that the current solution is globally optimal. Assume $0 < \epsilon < 1$. If the algorithm terminates, then $\gamma \geq 1 - \epsilon > 0$. For any iteration with $0 < \gamma < 1$, the current dual vector $\mathbf{p}$ may violate dual feasibility for ungenerated columns. However, we can construct a feasible dual vector for the global problem by scaling: $\mathbf{p}_d \triangleq \mathbf{p}/\gamma$. Since $\mathbf{p}^\top \mathbf{A}_k \geq \gamma$ for all k, it follows that $\mathbf{p}_d^\top \mathbf{A}_k \geq 1$, satisfying all dual constraints. By weak duality, the optimal primal flow $f(\boldsymbol{\theta}^*)$ is upper-bounded by the dual objective value of $\mathbf{p}_d$, i.e., $f(\boldsymbol{\theta}^*) \leq \mathbf{p}_d^\top \boldsymbol{\mu} = \frac{\mathbf{p}^\top \boldsymbol{\mu}}{\gamma} = \frac{z}{\gamma}$, where the last equality holds by strong duality of the restricted master problem ($z = \mathbf{p}^\top \boldsymbol{\mu}$). Therefore, $z \geq \gamma f(\boldsymbol{\theta}^*)$. The algorithm terminates when $\gamma \geq 1-\epsilon$, which guarantees $z \geq (1-\epsilon) f(\boldsymbol{\theta}^*)$.

Theorem 3. *Using an exact algorithm for the pricing subproblem (UVMP) (e.g., DynVMP), UVMP-CG guarantees a flow value of at least* $(1-\epsilon) f(\boldsymbol{\theta}^*)$.

Algorithm 2 summarizes the overall procedure. It initializes with slack variables and iteratively solves the restricted master problem, invokes DynVMP to select the column with maximum reduced cost, adds the generated column to the restricted master problem, and checks the duality gap for termination.

Algorithm 2 UVMP-CG Algorithm

1: **Initialization:** Initialize $\mathbf{A}_\mathrm{r} \leftarrow \mathbf{I}$, $\mathbf{B} \leftarrow \mathbf{I}$, $\boldsymbol{x} \leftarrow \boldsymbol{\mu}$, and coefficients $\boldsymbol{\beta} \leftarrow \mathbf{0}$.
2: **while** True **do**
3: **Solve restricted master problem:** Solve restricted master problem to optimality. Obtain primal value z and dual vector $\mathbf{p}$.
4: **Pricing:**
5: **for** each commodity $(c, \phi) \in (\mathcal{C}, \Phi)$ **do**
6: Compute an optimal mapping $\tilde{m}_{(c,\phi)}$ via DynVMP under shadow prices $c_{uv} = p_{uv}$ and $c_u = p_u$.
7: Select $\hat{m} \leftarrow \tilde{m}_{(c^*,\phi^*)}$, where $(c^*, \phi^*) = \arg\min_{(c,\phi)\in(\mathcal{C},\Phi)} \mathbf{p}^\top \mathbf{A}_{\tilde{m}_{(c,\phi)}}$.
8: Set $\gamma \leftarrow \mathbf{p}^\top \mathbf{A}_{\hat{m}}$.
9: **if** $\gamma \geq 1 - \epsilon$ **then**
10: **Return** the mapping variables in the current RMP solution.
11: **Update:** Add the generated column $\mathbf{A}_{\hat{m}}$ (with objective coefficient 1) to the restricted master problem.

Convergence and complexity analysis. Under an anti-cycling pivot rule (e.g., Bland's rule), UVMP-CG terminates after at most $T_c \leq |\mathcal{M}|$ iterations, since each iteration adds a new mapping and no more than $|\mathcal{M}|$ mappings exist. The complexity of the pricing subproblem is $T_{\text{price}} = \sum_{(c,\phi)} O\big(|V^\phi|^3\, |V|^{2\mathrm{tw}(\mathcal{T}_\phi)+3}\big)$ [12]. Let $\mathcal{T}_{\text{LP}}(m, n)$ denote the time complexity to solve a linear program with m constraints and n variables. At iteration t, the restricted master problem has $|E|+|V|$ constraints and $|E|+|V|+t$ nonnegative variables (slack variables and t generated mapping variables). Thus, the per-iteration cost is XP in the treewidth and is bounded by $O(\mathcal{T}_{\text{LP}}(|E|+|V|, |E|+|V|+t) + T_{\text{price}})$. Although the number of iterations T_c has no worst-case polynomial bound, empirical results in Section V demonstrate that UVMP-CG converges in far fewer iterations than UVMP-MWU.

Remark 2. *The DynVMP algorithm is the most computationally intensive step in both UVMP-MWU and UVMP-CG. In practice, one can use approximation or heuristic algorithms for DynVMP to reduce the runtime. In Section V, we replace DynVMP with a heuristic. While this heuristic variant does not retain the theoretical* $(1-\epsilon)$ *guarantee, simulations show that*

it still empirically outperforms representative domain-specific baselines under the same throughput metric.

V. Simulation Results

In this section, we evaluate the algorithms using the exact algorithm for UVMP on synthetic instances and the heuristic variants on large AI workloads through simulations[1].

A. Experimental Setup

We evaluate three physical topologies: a 2×3 mesh, the Abilene backbone (11 nodes, 14 links) [19], and the Milan network (30 nodes, 35 links) [20]. All link and node capacities are set to 1 unless stated otherwise. We consider two workload classes: (i) Synthetic DAGs: each contains 3–5 functions with scaling factors sampled uniformly from $\{1, \ldots, 5\}$; and (ii) AI computational DAGs, including computation graphs (ResNet-50, Inception-V3, BERT) and MoE models (Mixtral $8 \times 7B$, Llama-4-Maverick-17B-128E, and Qwen3-235B-A22B).

We implement UVMP-MWU and UVMP-CG, replacing DynVMP with the RW-MaxMatch heuristic [21] for large AI computational DAGs. Baselines include the LP optimum computed by Gurobi [22] and domain-specific algorithms: HSDAG [6], Lazarus [23], and FlexMoE [24].

B. Performance Analysis of Approximation Algorithms

This subsection evaluates the approximation guarantee, convergence behavior, and runtime efficiency of UVMP-MWU and UVMP-CG for the MMCF-DAG problem.

1) Approximation guarantee: We verify the $(1-\epsilon)$-approximation guarantee of UVMP-MWU and UVMP-CG on a 2×3 mesh network with three commodities, each associated with a three-function service DAG. We test $\epsilon \in \{0.05, 0.10, 0.20\}$ and report results averaged over 10 trials. Table I compares UVMP-MWU and UVMP-CG with the LP optimum computed by Gurobi.

TABLE I
Performance guarantee of UVMP-MWU and UVMP-CG.

LP Obj. value	ϵ	UVMP-MWU		UVMP-CG	
		Obj. value	Approx. ratio	Obj. value	Approx. ratio
0.72	0.05	0.72	**1**	0.72	**1**
	0.10	0.72	**1**	0.72	**1**
	0.20	0.71	**0.99**	0.70	**0.97**

As shown in Table I, both algorithms consistently achieve the promised approximation ratios across all settings. For tighter tolerances ($\epsilon = 0.05$ and $\epsilon = 0.10$), UVMP-MWU and UVMP-CG attain approximation ratios close to 1. Even for $\epsilon = 0.20$, the observed ratios (about 0.99 for UVMP-MWU and 0.97 for UVMP-CG) remain well above the theoretical lower bound $(1-\epsilon) = 0.80$. Overall, these results validate Theorems 2 and 3: the proposed algorithms guarantee the $(1-\epsilon)$-approximation and, in practice, often deliver near-optimal solutions far better than the worst-case bound.

[1]All simulations are executed on a computer equipped with 16 GB RAM and a 12th Gen Intel Core i5-12600KF CPU running at 3.7 GHz.

2) Convergence analysis: Using the same simulation setup, we study the convergence of UVMP-MWU and UVMP-CG using $\epsilon \in \{0.4, 0.3, 0.2, 0.1, 0.05, 0.01\}$. Table II reports the average number of iterations under different ϵ values.

TABLE II
Comparison of average iterations for UVMP-MWU and UVMP-CG under different ϵ.

ϵ	UVMP-MWU	UVMP-CG
0.4	154	20
0.3	277	23
0.2	647	31
0.1	2,751	35
0.05	11,622	44
0.01	331,561	62

As observed in Table II, the two algorithms exhibit distinct convergence behaviors. UVMP-CG converges in substantially fewer iterations than UVMP-MWU. This is because UVMP-CG, via the simplex method on the restricted master problem, selects mappings (columns) with the maximum reduced cost at each step. In contrast, UVMP-MWU performs multiplicative weight updates, which operate as a first-order method and typically require many smaller updates to converge. Consequently, the reduction in iteration count suggests that UVMP-CG is more computationally efficient. We next evaluate on the Abilene and Milan topologies to demonstrate that this convergence advantage translates into shorter wall-clock runtime.

3) Runtime comparison: The LP of MMCF-DAG contains an exponential number of mapping variables, which makes a direct solution intractable on large instances. In our simulations, Gurobi fails to solve the LP within a time limit of 100,000 seconds on the Milan topology. We therefore compare only UVMP-MWU and UVMP-CG on the Abilene and Milan networks under two settings: (i) five commodities with four-function DAGs, and (ii) two commodities with five-function DAGs. Table III reports the objective values and wall-clock runtimes under these scenarios, averaged over 10 trials.

As indicated in Table III, both proposed algorithms can handle large-scale instances for which the exact LP solver fails. In terms of efficiency, UVMP-CG substantially outperforms UVMP-MWU: across all tested topologies and commodity settings, UVMP-CG reduces wall-clock runtime by 91%–98% relative to UVMP-MWU. Importantly, this speedup does not come at the expense of solution quality, as UVMP-CG achieves objective values close to UVMP-MWU. Overall, these results identify UVMP-CG as a scalable and efficient approach for the MMCF-DAG problem.

C. Heuristic Variants for Large AI Workloads

We evaluate heuristic variants on AI computation tasks. Given the large service DAGs, exact DynVMP pricing becomes computationally prohibitive. We replace the exact algorithm for UVMP with RW-MaxMatch [21] in the mapping-selection or pricing step, yielding UVMP-MWU-H and UVMP-CG-H. In this heuristic, the resource prices are set to $c_{uv} = w_{uv}/\mu_{uv}$ and $c_u = w_u/\mu_u$ for UVMP-MWU-H, and $c_{uv} = p_{uv}$ and $c_u = p_u$ for UVMP-CG-H. RW-

TABLE III
WALL-CLOCK RUNTIME COMPARISON OF UVMP-MWU AND UVMP-CG ON ABILENE AND MILAN TOPOLOGIES.

Network	Commodities	ϵ	UVMP-MWU		UVMP-CG		Runtime reduction
			Obj. value	Runtime (s)	Obj. value	Runtime (s)	
Abilene	Five (4-func)	0.05	1.15	27510.45	1.14	418.25	**98.48%**
		0.10	1.14	17250.80	1.14	390.50	**97.74%**
		0.20	1.13	6650.15	1.13	368.10	**94.46%**
	Two (5-func)	0.05	0.75	35890.20	0.74	660.15	**98.16%**
		0.10	0.74	21105.50	0.74	565.40	**97.32%**
		0.20	0.71	7680.90	0.70	475.80	**93.80%**
Milan	Five (4-func)	0.05	1.25	33450.10	1.24	1050.30	**96.86%**
		0.10	1.22	22980.60	1.18	855.20	**96.28%**
		0.20	1.21	9650.40	1.15	785.90	**91.86%**
	Two (5-func)	0.05	0.85	36980.50	0.84	1210.15	**96.73%**
		0.10	0.83	26850.30	0.83	960.50	**96.42%**
		0.20	0.79	12510.80	0.79	815.40	**93.48%**

MaxMatch generates a feasible placement of service functions using topology-aware ranking and maximum matching. Each dependency (i, j) is routed along the shortest path from $m(i)$ to $m(j)$ under link length $\xi^{ij} c_{uv}$; if $m(i) = m(j)$, the dependency is mapped to $\emptyset$. The resulting placement forms a valid UVMP column. Since this mapping is not guaranteed to be minimum-cost, the -H variants do not retain the theoretical $(1-\epsilon)$ guarantee. Nevertheless, the simulations below demonstrate that they achieve higher throughput than existing domain-specific baselines.

1) Computation Graphs: A computation graph can be modeled as a DAG, where nodes represent operators and edges capture their dependencies. We follow the benchmark in [6] and consider ResNet-50, Inception-V3, and BERT. The communication requirements between operators and the computation requirements of each operator are computed as in [6], and sketches of these computation graphs are provided in the appendix of [6]. The objective is to maximize throughput: images/s for vision models and tokens/s for language models. We consider two workstations, each with one CPU (2 TFLOP/s) and two GPUs (20 TFLOP/s each). Intra-workstation links use NVLink (600 GB/s) and inter-workstation links use Ethernet (50 GB/s).

As shown in Table IV, under this throughput evaluation, UVMP-MWU-H and UVMP-CG-H achieve higher throughput than HSDAG across all models. This gap arises from the difference in objectives: HSDAG optimizes device placement for a single computation graph to minimize execution time, whereas our formulation explicitly targets network throughput, making resource sharing and flow splitting directly aligned with the throughput objective across heterogeneous devices and communication links.

TABLE IV
EVALUATION OF UVMP-MWU-H AND UVMP-CG-H ON COMPUTATION GRAPHS.

Model	UVMP-MWU-H	UVMP-CG-H	HSDAG
ResNet-50 (images/s)	15310	12522	8206
Inception-V3 (images/s)	5908	5577	2851
BERT-base (tokens/s)	1.46×10^6	1.60×10^6	1.13×10^6

2) MoE models: For MoE models, each Transformer layer comprises multi-head self-attention, expert networks, and auxiliary modules (e.g., residual connections, gating, communication, and aggregation). Since multi-head self-attention and the expert networks dominate computation, we model only these two modules and ignore the remaining components. We evaluate three MoE models assuming FP32 precision and 8K-token sequences. Mixtral 8×7B (32 layers, 8 experts, top-2 gating) incurs computational loads of 2.2 TFLOPs for attention and 1.45 TFLOPs for experts, with 64 MB communication overhead per edge. Llama-4-Maverick-17B-128E (48 layers alternating between dense and MoE; one shared and one routed expert per token) requires 2.4 TFLOPs for attention, 1.4 TFLOPs for dense networks, and 0.7 TFLOPs for experts; communication is 128 MB for dense/routed components and 64 MB for shared experts. Qwen3-235B-A22B (94 layers, 128 experts, top-8 gating) consumes 1.9 TFLOPs for attention and 0.2 TFLOPs for experts, with a uniform 64 MB communication cost. In LLM serving systems, multiple replicas of expert networks may be deployed on different nodes, enabling diverse routing and processing strategies [23], [24]. We compare against Lazarus [23] and FlexMoE [24]. Suppose that the computing network consists of six workstations, each with four GPUs (82 TFLOP/s per GPU). We assume that intra-workstation GPU links use NVLink with a bandwidth of 600 GB/s, while inter-workstation links use Ethernet with a bandwidth of 50 GB/s. Fig. 2 reports the maximum supported token throughput.

Fig. 2 indicates that the heuristic variants achieve higher token throughput than the representative MoE placement baselines (Lazarus and FlexMoE) under the simplified model. This gain stems from the service DAG model's ability to capture the fork–join traffic patterns in MoE architectures and to account for the communication overhead. Unlike Lazarus and FlexMoE, which employ heuristic expert placement and may suffer from load imbalance or link congestion, the proposed joint routing and placement heuristic explicitly accounts for both communication and computation loads, helping alleviate such bottlenecks under the simplified model.

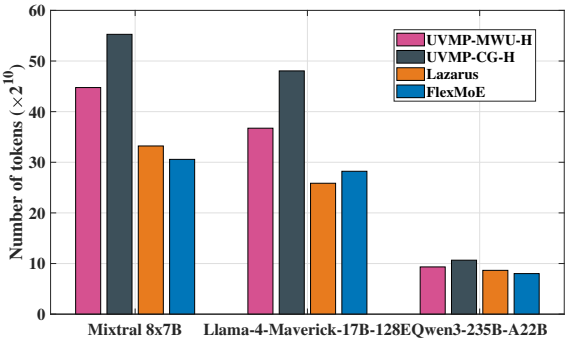

Fig. 2. Evaluation of UVMP-MWU-H and UVMP-CG-H for MoE models.

VI. Conclusion

In this paper, we formulate the MMCF-DAG problem to compute the maximum throughput of computing networks that host services with complex DAG dependencies. We derive a path-based LP formulation and establish the NP-completeness of UVMP. To address computational challenges, we develop two approximation algorithms, UVMP-MWU and UVMP-CG, and prove that both achieve the $(1-\epsilon)$-approximation. Simulations show that UVMP-CG converges substantially faster than UVMP-MWU and that both algorithms are significantly faster than exact LP solvers. Moreover, the heuristic variants scale to large AI workloads and achieve higher throughput than domain-specific baselines.

Appendix A
Proof of Theorem 1

Proof: A UVMP mapping can be verified in polynomial time by evaluating $\mathrm{cost}(m)$ in Definition 3; hence UVMP-Dec is in NP.

We reduce from Multiway Cut. Consider an instance $\langle G_m = (V_m, E_m), \omega, V_T, W_m \rangle$ with $|V_T| \geq 3$. Construct a computing network with one physical node u_t for each terminal $t \in V_T$ and bidirectional links with unit cost between every pair of terminal nodes; all node costs are zero. Construct a service DAG with one function u^ϕ for each $u \in V_m$, and orient each edge $(u,v) \in E_m$ according to an arbitrary ordering, so the constructed service graph is acyclic. For each terminal $t \in V_T$, add an auxiliary service source S_t^ϕ and an auxiliary service sink D_t^ϕ, both anchored to the physical node u_t, and add the external edges (S_t^ϕ, t^ϕ) and (t^ϕ, D_t^ϕ). Set $r^{u^\phi} = 0$ for every $u \in V_m$, set $\xi^{(u^\phi, v^\phi)} = \omega_{uv}$ for every original edge, and set scaling factors of all external edges to $M = W_m + \sum_{(u,v)\in E_m} \omega_{uv}$. Let the UVMP threshold be $W = W_m$.

If there exists a multiway cut E'_m with $\sum_{(u,v)\in E'_m} \omega_{uv} \leq W_m$, remove it from G_m. Each resulting component contains at most one terminal; map all functions in a terminal-containing component to the corresponding terminal node, and map functions in terminal-free components arbitrarily. Edges within a component are mapped to empty paths, while edges crossing components are mapped to single-hop links between terminal nodes. The total cost is at most $\sum_{(u,v)\in E'_m} \omega_{uv} \leq W_m$, yielding a feasible UVMP solution.

Conversely, suppose there is a UVMP mapping m with $\mathrm{cost}(m) \leq W$. Since $M > W$ and every nonempty physical path has cost at least one, every external edge must be mapped to an empty path. Hence, for each terminal $t \in V_T$, both external edges incident to t^ϕ force $m(t^\phi) = u_t$. Define

$$E'_m = \{(u,v) \in E_m \mid m(u^\phi) \neq m(v^\phi)\}.$$

Then $\sum_{(u,v)\in E'_m} \omega_{uv} \leq \mathrm{cost}(m) \leq W_m$. If two terminals remained connected in $G_m \setminus E'_m$, all functions along the path would have the same physical image, contradicting their distinct anchored terminal nodes. Hence E'_m is a multiway cut of weight at most W_m. ∎

References

[1] F. Baccelli, W. A. Massey, and D. Towsley, "Acyclic fork-join queuing networks," *J. ACM*, vol. 36, no. 3, pp. 615–642, 1989.

[2] Y. Dallery, Z. Liu, and D. Towsley, "Properties of fork/join queueing networks with blocking under various operating mechanisms," *IEEE Trans. Robot. Autom.*, vol. 13, no. 4, pp. 503–518, 2002.

[3] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, "A survey on mixture of experts in large language models," *IEEE Trans. Knowl. Data Eng.*, vol. 37, no. 7, pp. 3896–3915, 2025.

[4] Apache Software Foundation, "Apache airflow: A platform to programmatically author, schedule, and monitor workflows," Online, 2026. [Online]. Available: https://airflow.apache.org/

[5] Kubeflow Community, "Kubeflow: Machine learning toolkit for kubernetes," Online, 2026. [Online]. Available: https://www.kubeflow.org/

[6] S. Duan *et al.*, "A structure-aware framework for learning device placements on computation graphs," *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, pp. 81 748–81 772, 2024.

[7] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," in *Int. Symp. Algorithm. Asp. Cloud Comput.*, 2018, pp. 73–101.

[8] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2017, pp. 1–9.

[9] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 2132–2140.

[10] X. Lin, C. Liu, L. Luo, D. Guo, and M. Xu, "Nest: Optimal deploying DAG-SFCs to maximize the flows wholly served in the network edge," *Comput. Netw.*, vol. 235, p. 109995, 2023.

[11] A. Satpathy *et al.*, "Virtual network embedding: Literature assessment, recent advancements, opportunities, and challenges," *IEEE Commun. Surv. Tutor.*, vol. 27, no. 6, pp. 3861–3914, 2025.

[12] M. Rost, E. Döhne, and S. Schmid, "Parametrized complexity of virtual network embeddings: Dynamic & linear programming approximations," *Comput. Commun. Rev.*, vol. 49, no. 1, pp. 3–10, 2019.

[13] M. Rost and S. Schmid, "On the hardness and inapproximability of virtual network embeddings," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 791–803, 2020.

[14] H. Lan, L. Chen, and B. Li, "Accelerated device placement optimization with contrastive learning," in *Proc. Int. Conf. Parallel Process. (ICPP)*, 2021, pp. 1–10.

[15] R. Addanki, S. B. Venkatakrishnan, S. Gupta, H. Mao, and M. Alizadeh, "Placeto: Learning generalizable device placement algorithms for distributed machine learning," *arXiv preprint arXiv:1906.08879*, 2019.

[16] Y. Hu *et al.*, "Giph: Generalizable placement learning for adaptive heterogeneous computing," *Proc. Mach. Learn. Syst.*, vol. 5, pp. 164–185, 2023.

[17] M. Rost and S. Schmid, "Virtual network embedding approximations: Leveraging randomized rounding," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 2071–2084, 2019.

[18] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: a meta-algorithm and applications," *Theory Comput.*, vol. 8, no. 1, pp. 121–164, 2012.

[19] Internet2, "Internet2 Abilene backbone network upgrade passes transcontinental milestone," Internet2 News Release, Feb. 2003. [Online]. Available: https://lists.internet2.edu/sympa/arc/i2-news/2003-02/msg00001.html

[20] B. Xiang, J. Elias, F. Martignon, and E. Di Nitto, "A dataset for mobile edge computing network topologies," *Data in Brief*, vol. 39, p. 107557, 2021.

[21] X. Cheng *et al.*, "Virtual network embedding through topology-aware node ranking," *Comput. Commun. Rev.*, vol. 41, no. 2, pp. 38–47, 2011.

[22] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," Online documentation, 2025. [Online]. Available: https://www.gurobi.com

[23] Y. Wu *et al.*, "Lazarus: Resilient and elastic training of mixture-of-experts models with adaptive expert placement," *arXiv preprint arXiv:2407.04656*, 2024.

[24] X. Nie *et al.*, "Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement," *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 1–19, 2023.