Centre Énergie, Matériaux et Télécommunications

Deep Neural Network Inference Task Offloading and Resource Allocation for UAV-based Wireless Networks

Par

Muhammad Ismail

Mémoire présenté pour l'obtention du grade de
Mastère de Sciences, MSc.
en télécommunications

Jury d'évaluation

Président du jury et examinateur interne	Prof. Zakaria Abou El Houda *INRS-EMT*
Examinateur externe	Prof. Thi-Mai-Trang Nguyen *Sorbonne Paris Nord University*
Examinateur interne	Prof. Zakaria Abou El Houda *INRS-EMT*
Directeur de recherche	Prof. Long Bao Le *INRS-ÉMT*

Acknowledgment

I would like to express my deepest gratitude to my supervisor, Professor Long Bao Le, for his invaluable support and motivation throughout my MSc studies at INRS-EMT, University of Québec. His consistent guidance, encouragement, and unwavering patience in mentoring my academic progress have been pivotal in my journey. Professor Le's profound expertise, passion for research, and remarkable empathy toward my individual circumstances have been a constant source of inspiration.

I am sincerely thankful to Professor Zakaria Abou El Houda for his insightful feedback and for reviewing my dissertation as a member of my MSc thesis evoluation committee. I am also thankful to Professor Thi-Mai-Trang Nguyen from Sorbonne Paris Nord University, who served as an external examiner and provided constructive feedback to enhance the technical quality of my dissertation.

I wish to acknowledge my colleagues at the Networks and Cyber-Physical Systems Lab (NECPHY-Lab) at INRS-EMT, University of Québec and all my friends at Montreal. Their thoughtful recommendations and camaraderie have enriched my research experience and made my time in Canada unforgettable. Finally, I owe my heartfelt thanks to my family, including my parents, Nigar Bibi and Shah Room, my brother, Abubakar Saddique, and all other members of my family, for their unwavering support and encouragement in all aspects of my life. Their belief in me and constant presence have been the bedrock of my achievements.

Abstract

Novel integration of Deep Neural Networks (DNNs) into Unmanned Aerial Vehicles (UAVs) based systems has enabled numerous smart civilian and military applications. However, the inherent computational complexity of DNNs often results in significant inference delay, which could violate the delay requirements of practical UAV operations and applications. UAVs can be equipped with relatively strong servers so they can collaboratively perform inference for pre-trained DNNs, enabling complex recognition tasks based on onboard sensing data such as image and video. Such the collaborative inference is critical for applications where ground communications and computing infrastructure is not available, not secure or cost-efficient such as those for military, disaster recovery and rescue. Collaborative DNN inference in the UAV wireless network, is, however, challenging because one must decide how the computation load related to different layers of the DNN is distributed among UAVs and how to efficiently allocate both radio and computing resources to facilitate the underlying offloading process. This thesis aims to address these challenges where we make the following novel contributions.

First, we formulate the joint DNN layer assignment, radio and computing resource allocation problem as an optimization problem which aims to minimize the total inference latency considering constraints on the processing order of DNN layers, wireless connectivity, and limited computational resources of individual UAVs. To solve this difficult mixed integer and non-linear problem, we employ an alternating optimization technique and develop an efficient algorithm, named LARA. Numerical studies show that LARA performs very well in different studied scenarios and achieves up to 80% improvement in terms of inference latency compared to other baselines which perform DNN layer assignment and resource allocation in a heuristic manner.

Second, we propose a novel decentralized matching game theory-based algorithm (called GAME-MIND), enabling efficient allocation of DNN layers of different inference requests to the UAVs considering UAVs' processing quotas. By modeling the problem as a matching game, GAME-MIND ensures efficient resource utilization while balancing the computation load across UAVs in the network. Numerical results show that GAME-MIND achieves desirable delay performance compared to the centralized LARA counterpart. In addition, GAME-MIND greatly outperforms two different heuristic-based methods that offload DNN layers to nearest neighboring UAVs where these two heuristic baselines can achieve at least twice the inference delay due to GAME-MIND in high-load conditions. We also demonstrate the efficacy of GAME-MIND in terms of convergence and study its achieved communications, computation latency, and total inference delay under different network settings.

Contents

List of Figures

List of Abbreviations

5G	Fifth-generation Mobile Communications Technology
AdaBoost	Adaptive Boosting
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
FLOPS	Floating Point Operations per Second
IoT	Internet of Things
LSTM	Long short-term memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
RL	Reinforcement Learning
DRL	Deep Reiforcement Learning
UAV	Unmanned Aerial Vehicle
MILP	Mixed-Integer Linear Programming
MEC	Mobile Edge Computing
MINLP	Mixed-Integer Nonlinear Programming
SCA	Successive Convex Approximation

Chapter 1

Résumé Long

Ce chapitre présente le résumé en français de la thèse intitulée:

"Déchargement des tâches d'inférence de réseaux neuronaux profonds et allocation des ressources pour les réseaux sans fil basés sur des UAV"

1.1 Contexte et motivation

Avec les progrès récents de l'Internet des objets (IoT), des applications mobiles et des technologies émergentes telles que la réalité augmentée, les systèmes autonomes et l'apprentissage profond, la demande de systèmes informatiques et sans fil à faible latence et haute performance augmente. Le cloud computing traditionnel pourrait avoir du mal à répondre à ces exigences en raison de sa dépendance à des centres de données centralisés, ce qui peut entraîner une latence de communication importante. Pour relever ces défis, l'edge computing est devenu un paradigme informatique qui déploie les ressources de calcul au plus près des sources de données, telles que les terminaux ou les capteurs distants de collecte de données spécifiques à une mission. Ainsi, l'edge computing réduit considérablement les temps de réponse et améliore les performances globales du système, permettant ainsi le développement d'applications temps réel. Cependant, les capacités de calcul limitées, les contraintes énergétiques et les défis de gestion des ressources comptent parmi les principaux défis auxquels sont confrontés les systèmes d'edge computing pour gérer des tâches complexes et gourmandes en ressources. Pour surmonter ces limitations, le déchargement des tâches de calcul

a été proposé, permettant aux terminaux de déléguer les tâches de calcul lourdes à des appareils ou serveurs à proximité, améliorant ainsi l'efficacité, prolongeant l'autonomie de la batterie et optimisant les performances globales des applications temps réel.

Cependant, la conception du délestage des tâches de calcul nécessite une prise en compte conjointe de plusieurs facteurs, tels que l'état du réseau, l'efficacité énergétique, les risques de sécurité, la répartition de la charge de travail et la disponibilité des ressources de calcul. La nature dynamique de la bande passante du réseau, les exigences de charge de travail et la mobilité des terminaux complexifient encore davantage le processus de délestage. De plus, l'efficacité énergétique demeure un défi majeur, en particulier pour les dispositifs IoT et mobiles alimentés par batterie, qui doivent trouver un équilibre entre performance de calcul et consommation d'énergie.

Afin de répondre aux exigences de qualité de service des applications mobiles gourmandes en données, une gamme de solutions de MEC a été introduite. Cependant, le déploiement de serveurs Edge n'est pas toujours envisageable dans des environnements difficiles tels que les régions isolées, les terrains montagneux ou les zones touchées par des catastrophes naturelles. Dans ces cas, les UAV offrent une alternative prometteuse. Grâce à la communication en visibilité directe (LoS), les UAV peuvent faciliter le transfert de tâches et améliorer l'efficacité des téléchargements. Le MEC assisté par UAV étant un domaine de recherche encore en développement, les recherches approfondies menées jusqu'à présent ont été limitées. Par exemple, les auteurs de [1] ont mis en évidence plusieurs scénarios critiques, tels que les zones sinistrées, les zones de conflit et les zones à forte demande, où les UAV équipés de capacités MEC pourraient offrir des avantages significatifs.

Motivée par ces défis, cette recherche vise à développer des stratégies d'externalisation du calcul adaptatives et efficaces en termes de latence pour le système d'informatique en périphérie basé sur les UAV, afin de répondre efficacement aux requêtes d'inférence des DNN. Grâce à l'intégration d'algorithmes de ML et d'optimisation, les décisions d'externalisation du calcul peuvent être prises dynamiquement en fonction des conditions réseau en temps réel, des variations de charge de travail et des besoins spécifiques des applications. En concevant des techniques novatrices et efficaces d'externalisation des tâches de calcul, cette recherche cherche à améliorer les performances du système, à réduire la latence d'inférence des DNN et à permettre l'exécution fluide d'applications sensibles à la latence dans les écosystèmes d'informatique en périphérie. En définitive, cette étude ambitionne de contribuer au développement de mécanismes d'externalisation du calcul robustes,

évolutifs et performants, capables de répondre aux exigences croissantes des applications informatiques modernes, en permettant le traitement des inférences DNN dans un environnement sans fil basé sur des UAV.

1.2 Contribution originale de la recherche

L'intégration innovante des DNN dans les systèmes basés sur des UAV a permis de nombreuses applications civiles et militaires intelligentes. Cependant, la complexité de calcul inhérente aux DNN entraîne souvent un retard d'inférence important, susceptible de violer les exigences de délai des opérations et applications pratiques des UAV. Nous montrons comment le déchargement conjoint des couches DNN et l'allocation de ressources pour prendre en charge l'inférence DNN dans les systèmes sans fil basés sur des UAV peuvent être formulés comme un problème d'optimisation, visant à minimiser la latence totale d'inférence en tenant compte des contraintes sur l'ordre de traitement des couches DNN, la connectivité sans fil, les ressources de calcul limitées et les quotas de traitement de chaque UAV. Nous proposons ensuite un algorithme d'optimisation (appelé LARA) qui utilise la méthode d'optimisation alternée en abordant le problème formulé par la résolution itérative des trois sous-problèmes jusqu'à convergence.

Pour une potentielle implémentation décentralisée, un nouvel algorithme basé sur la théorie des jeux de correspondance (appelé GAME-MIND) est conçu pour permettre une allocation efficace des couches DNN des différentes requêtes d'inférence aux UAV. En modélisant le problème comme un jeu de correspondance, GAME-MIND garantit une utilisation efficace des ressources tout en répartissant la charge de calcul entre les UAV du réseau. Les résultats numériques montrent que GAME-MIND atteint des performances de délai souhaitables par rapport à son homologue LARA centralisé. De plus, GAME-MIND surpasse largement deux méthodes heuristiques différentes qui déchargent les couches DNN vers les UAV voisins les plus proches, ces deux bases heuristiques pouvant atteindre au moins deux fois le délai d'inférence dû à GAME-MIND en conditions de charge élevée. Nous présentons ci-dessous les détails techniques de notre contribution ainsi que les résultats numériques obtenus.

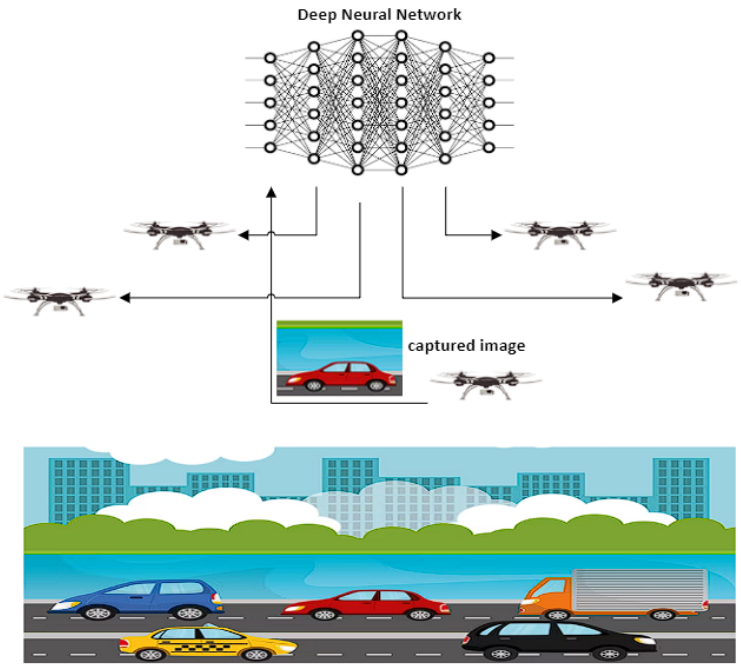

Figure 1.1: Modèle de système

1.3 Modèle de système

Nous considérons un réseau sans fil de UAV connectés, tel qu'illustré à la figure 1.1, dans lequel chaque UAV peut transmettre des données à n'importe quel autre UAV du réseau via des communications à saut unique ou multiple.[1] Nous supposons que les positions de tous les UAV sont fixes (c'est-à-dire qu'ils planent sur une zone désignée et maintiennent la connectivité avec les autres UAV) pendant l'exécution des requêtes d'inférence du DNN considérées. Nous supposons également que les UAV effectuent de manière collaborative l'inférence d'un DNN pré-entraîné à partir des données collectées par un sous-ensemble de UAV. Les tâches d'inférence pour cet ensemble de UAV correspondent à l'ensemble des requêtes de calcul. Plus précisément, le DNN est composé de plusieurs couches et les calculs requis par la tâche d'inférence correspondent au traitement couche par couche du DNN. Pour plus de commodité, nous désignons les ensembles de UAV, de demandes d'inférence et de couches DNN par $\mathcal{U} = \{1, 2, 3, ..., N\}$, $\mathcal{R} = \{1, 2, 3, ..., R\}$ et $\mathcal{L} = \{1, 2, 3, ..., L\}$, respectivement.

Chaque UAV initiant la requête d'inférence correspondante peut effectuer les calculs requis par toutes les couches DNN; cependant, le déchargement de certaines couches DNN vers des UAV

[1] Le placement des UAV pour former le réseau connecté dépasse le cadre de cet article et peut être résolu en utilisant la technique décrite dans [2].

voisins disposant de ressources de calcul importantes peut améliorer la latence d'inférence. Une telle conception de déchargement doit prendre en compte la transmission des données liées aux couches DNN déchargées et l'optimisation de l'allocation des ressources de calcul. Les détails de la conception du déchargement et de l'allocation des ressources sont décrits ci-dessous.

1.3.1 Modèle de communication

Pour capturer la connectivité entre les UAV, nous introduisons des paramètres binaires pour chaque paire de UAV, $\lambda_{i,k}, \forall i,k \in \mathcal{U}$, tels que,

$$\lambda_{i,k} = \begin{cases} 1 & \text{si UAV } i \text{ et } k \text{ sont connectés,} \\ 0 & \text{sinon.} \end{cases}$$

Par souci de concision, nous désignons le lien connecté entre le drone i et k par $\{(i,k)$, if $\lambda_{i,k} = 1, \forall i,k \in \mathcal{U}\}$. Dans notre conception, nous supposons que les UAV restent connectés pendant toute la durée de la session d'inférence.

Soit $g_{i,k,c}$ le gain de canal de la liaison UAV (i,k) sur le canal assigné et N_0 la densité de puissance du bruit gaussien au niveau du récepteur. Le débit $\delta_{i,k}$ de la liaison (i,k) peut alors être calculé comme suit:

$$\delta_{i,k} = \omega \log_2 \left(1 + \frac{p_{i,k,c} g_{i,k,c}}{\omega . N_0}\right), \tag{1.1}$$

où ω désigne la bande passante de chaque canal, $p_{i,k,c}$ représente la puissance de transmission, et nous avons supposé des attributions de canaux orthogonales pour différentes liaisons réseau.

1.3.2 Demandes d'inférence DNN

Rappelons qu'il existe R requêtes d'inférence générées par différents UAV. Chaque requête d'inférence est composée de L sous-tâches de calcul correspondant à différentes couches du DNN, allant de la

couche $l = 1$ à la couche $l = L$. Pour décharger une sous-tâche particulière correspondant à la couche l de la requête r, nous devons transmettre les données associées à la couche précédente du DNN (c'est-à-dire les caractéristiques extraites par la couche concernée). Notez que les données associées aux différentes sous-tâches (r, l) peuvent être transmises sur le même lien (i, k). Soit $\pi_{r,l,i,k}$ le débit alloué à la sous-tâche (r, l) pour transmettre les données associées sur le lien (i, k). Nous avons alors,

$$\sum_{r=1}^{R}\sum_{l=1}^{L}\pi_{r,l,i,k} \leq \delta_{i,k}.\lambda_{i,k}. \tag{1.2}$$

Pour capturer les affectations de couche DNN pour différents UAV, soit $q_{r,l,i}$ une variable binaire capturant si la couche l de la requête r est affectée au drone i :

$$q_{r,l,i} = \begin{cases} 1 & \text{si la couche } l \text{ de la requête } r \text{ est attribuée au UAV}i, \\ 0 & \text{sinon.} \end{cases}$$

Par ailleurs, supposons que $\rho_{r,l,(l+1)}$ représente la quantité de données à transmettre si la couche $l + 1$ de la requête r est déchargée et traitée par un drone différent de la couche de traitement l du drone i. Le délai de communication $\tau_{r,l,i,k}$ impliqué dans le transfert des données associées à la couche l de la requête r du drone i vers le drone k peut alors être calculé comme suit:

$$\tau_{r,l,i,k} = \frac{q_{r,l,i}.q_{r,(l+1),k}.\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}. \tag{1.3}$$

1.3.3 Modèle informatique

Supposons que la capacité de calcul du UAV $i \in \mathcal{U}$ soit S_i. Cette capacité doit alors être partagée pour traiter les différentes sous-tâches (r, l) attribuées à chaque UAV. Soit la ressource de calcul allouée au traitement de la sous-tâche (r, l) du UAV i soit $S_{r,l,i}$. On a alors

$$\sum_{r=1}^{R}\sum_{l=1}^{L}S_{r,l,i} \leq S_i. \tag{1.4}$$

Soit $\gamma_{r,l}$ la demande de calcul de la sous-tâche (r, l). Le délai de calcul de cette sous-tâche (r, l) au drone i peut alors être calculé comme suit:

$$\sigma_{r,l,i} = \frac{q_{r,l,i} \cdot \gamma_{r,l}}{S_{r,l,i}}. \tag{1.5}$$

1.3.4 Formulation du problème

Définissons les variables d'affectation de couche DNN, les variables d'allocation de ressources de calcul et les variables d'allocation de débit de données comme $\mathbf{Q} = \{q_{r,l,i}, \forall r \in \mathcal{R}, i \in \mathcal{U}, l \in \mathcal{L}\}$, $\mathbf{S} = \{S_{r,l,i}, \forall r \in \mathcal{R}, l \in \mathcal{L}, i \in \mathcal{U}\}$ et $\mathbf{R} = \{\pi_{r,l,i,k}, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, i, k \in \mathcal{U}\}$, respectivement. Ensuite, le problème d'optimisation conjointe pour minimiser la latence d'inférence totale, appelé problème **(P)**, peut être formulé comme,

$$\begin{aligned} \min_{\{\mathbf{Q},\mathbf{R},\mathbf{S}\}} & \sum_{r=1}^{R} \sum_{l=1}^{L} \sum_{i=1}^{N} \sum_{k=1}^{N} \frac{q_{r,l,i} \cdot q_{r,(l+1),k} \cdot \gamma_{r,l}}{S_{r,l,i}} \\ + & \sum_{r=1}^{R} \sum_{l=1}^{L-1} \sum_{i=1}^{N} \sum_{k=1}^{N} q_{r,l,i} \cdot q_{r,(l+1),k} \cdot \frac{\rho_{r,l,(l+1)} \cdot \lambda_{i,k}}{\pi_{r,l,i,k}} \end{aligned}$$

s.t

$$\sum_{i=1}^{N} q_{r,l,i} = 1, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \tag{1.6}$$

$$\sum_{r=1}^{R} \sum_{l=1}^{L} q_{rlk} \leq \zeta_{th,k}, \forall k \in \mathcal{U}, \tag{1.7}$$

$$\sum_{r=l}^{R} \sum_{l=l}^{L} S_{r,l,i} \leq S_i, \forall i \in \mathcal{U}, \tag{1.8}$$

$$q_{r,l,i} . q_{r,(l+1),k} \leq \lambda_{i,k}, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i, k \in \mathcal{U}, \tag{1.9}$$

$$\sum_{r=1}^{R} \sum_{l=1}^{L} \pi_{r,l,i,k} \leq \delta_{i,k} \lambda_{i,k}, \forall i, k \in \mathcal{U}, \tag{1.10}$$

$$q_{r,l,i} \in \{0, 1\}, \ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i \in \mathcal{U}, \tag{1.11}$$

$$S_{r,l,i} \geq 0, \ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i \in \mathcal{U}, \tag{1.12}$$

$$\pi_{r,l,i,k} \geq 0, \ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i, k \in \mathcal{U}, \tag{1.13}$$

où les contraintes (1.6) capturent le fait que chaque sous-tâche (r, l) est assignée et traitée par exactement un UAV, les contraintes (1.7) garantissent que le nombre total de couches DNN traitées par chaque UAV k est limité à sa limite prédéterminée $\zeta_{th,k}$ (également appelée quota de traitement) pour un traitement de calcul équilibré et l'évitement de la congestion, les contraintes (1.8) imposent que la ressource de calcul totale allouée pour prendre en charge différentes sous-tâches à chaque UAV i doit être limitée par la capacité de calcul disponible, les contraintes (1.9) impliquent qu'une sous-tâche ne peut être déchargée de l'UAV i vers l'UAV k que s'ils sont connectés, (1.10) capture la contrainte de capacité sans fil pour chaque lien (i, k). Enfin, les contraintes (1.11), (1.12), (1.13) capturent les caractéristiques binaires ou non négatives de différentes variables d'optimisation. Il s'agit d'un problème d'optimisation mixte non linéaire en nombres entiers, très difficile à résoudre.

1.4 Algorithme basé sur l'optimisation

Nous proposons d'utiliser la méthode d'optimisation alternée pour résoudre le problème d'optimisation présenté dans la section précédente. Plus précisément, nous concevons un nouveau cadre, appelé affectation de couches DNN et allocation de ressources (LARA), qui décompose le problème en plusieurs sous-problèmes, chacun comportant un ensemble de variables d'optimisation donné par

les valeurs des autres variables d'optimisation. Ces sous-problèmes sont résolus itérativement jusqu'à convergence. Nous expliquons comment les résoudre ci-après.

1.4.1 Sous-problème d'attribution de couche DNN

Pour des valeurs données de **S** et **R**, le sous-problème d'affectation de couche DNN peut être écrit comme

$$\begin{aligned}\textbf{(P1)}: \min_{\mathbf{Q}} & \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N}\frac{q_{r,l,i}\cdot q_{r,(l+1),k}\cdot\gamma_{r,l}}{S_{r,l,i}} \\ & + \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} q_{r,l,i}\cdot q_{r,(l+1),k}\cdot\frac{\rho_{r,l,(l+1)}\cdot\lambda_{i,k}}{\pi_{r,l,i,k}}\end{aligned}$$

s.t (1.6), (1.7), (1.9), and (1.11).

Le problème **(P1)** est un programme entier non linéaire en raison des termes quadratiques de la fonction objectif et des contraintes (1.9). Pour linéariser ce problème, nous introduisons les variables auxiliaires $y_{r,l,i,k}$ définies comme suit:

$$y_{r,l,i,k} = q_{r,l,i}\cdot q_{r,(l+1),k}, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i,k \in \mathcal{U}. \tag{1.14}$$

Ensuite, les contraintes suivantes doivent être imposées pour garantir que $y_{r,l,i,k}$ soit égal à $q_{r,l,i}\cdot q_{r,(l+1),k}$:

$$y_{r,l,i,k} \leq q_{r,l,i}, \tag{1.15}$$

$$y_{r,l,i,k} \leq q_{r,(l+1),k}, \tag{1.16}$$

$$y_{r,l,i,k} \leq q_{r,l,i} + q_{r,(l+1),k} - 1. \tag{1.17}$$

Par commodité, définissons $\mathbf{Y} = \{y_{r,l,i,k}.\forall r, l, i, k\}$. Le problème (**P1**) peut alors être transformé en le problème suivant:

$$\begin{aligned}\textbf{(P1.1)}: \min_{\{\mathbf{Q},\mathbf{Y}\}} & \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N}\frac{y_{r,l,i,k}.\gamma_{r,l}}{S_{r,l,i}} \\ & + \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} y_{r,l,i,k}.\frac{\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}\end{aligned}$$

soumis à des contraintes (1.6), (1.7), (1.9), (1.11), (1.15), (1.16), (1.17), et

$$y_{r,l,i,k} \in \{0,1\},\ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i,k \in \mathcal{U}. \tag{1.18}$$

Le problème **(P1.1)** est un programme entier linéaire, qui peut être résolu efficacement en utilisant l'outil Python CVXPY.

1.4.2 Sous-problème d'allocation des ressources informatiques

Pour des valeurs données de $\mathbf{Q}, \mathbf{R}$, le sous-problème d'allocation des ressources informatiques peut s'écrire comme

$$\begin{aligned}\textbf{(P2)}: \min_{\mathbf{S}} & \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N}\frac{q_{r,l,i}.q_{r,(l+1),k}.\gamma_{r,l}}{S_{r,l,i}} \\ & + \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} q_{r,l,i}.q_{r,(l+1),k}.\frac{\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}\end{aligned}$$

soumis à des contraintes (1.8) et (1.12).

Le problème (**P2**) est convexe car $1/S_{r,l,i}$ est convexe par rapport à $S_{r,l,i}$ et toutes les contraintes (1.8) et (1.12) sont linéaires. Il peut donc être résolu à l'aide de l'outil Python CVXPY.

1.4.3 Sous-problème d'allocation du débit de données

Pour des valeurs données de $\mathbf{Q}, \mathbf{S}$, le sous-problème d'allocation de taux peut être exprimé comme

$$\begin{aligned}\textbf{(P3)}: \min_{\mathbf{R}} & \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N}\frac{q_{r,l,i}\cdot q_{r,(l+1),k}\cdot\gamma_{r,l}}{S_{r,l,i}} \\ + & \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} q_{r,l,i}\cdot q_{r,(l+1),k}\cdot\frac{\rho_{r,l,(l+1)}\cdot\lambda_{i,k}}{\pi_{r,l,i,k}}\end{aligned}$$

sous les contraintes (1.10) et (1.13).

Le problème (**P3**) possède une fonction objectif convexe et des contraintes linéaires (1.10). Il peut donc être résolu à l'aide de l'outil Python CVXPY. Les détails de l'algorithme proposé sont résumés dans l'Algorithme 1.1 La CONDITION D'ARRÊT de l'algorithme itératif proposé est atteinte lorsque la différence de la latence d'inférence totale entre deux itérations consécutives devient inférieure à 1% de la latence d'inférence obtenue.

Algorithm 1.1. Algorithme d'Attribution des Couches DNN et d'Allocation des Ressources (LARA)

INPUT: N, $\mathcal{R}$, $\mathcal{L}$, $\{\lambda_{i,k}, \forall i, k \in \mathcal{U}\}$
OUTPUT: Variables optimisées **Q**, **S**, **R**, et temps total d'inférence

```
while CONDITION D'ARRÊT = FAUSSE do
    Résoudre le problème (P1.1) pour obtenir la solution d'attribution des couches DNN Q
    Résoudre le problème (P2) pour obtenir la solution d'allocation de calcul S
    Résoudre le problème (P3) pour obtenir la solution d'allocation de débit de données R
end while
Extraire les valeurs optimisées de Q, S, R
Calculer le temps total d'inférence
```

1.5 Algorithme basé sur un jeu d'appariement

Dans cette section, nous montrons comment résoudre le problème **(P)** en utilisant la théorie des jeux d'appariement. Cette technique permet de mettre en correspondance des agents de deux groupes (par exemple, travailleurs et emplois, ressources et appareils dans les systèmes de communication sans fil) de manière à répondre à des exigences prédéterminées, ce qui conduit souvent à un résultat stable et efficace [3]. La méthode de Gale-Shapley, parfois appelée algorithme d'acceptation différée,

est l'application la plus connue de la théorie des jeux d'appariement. Nous proposons notamment un algorithme décentralisé basé sur la méthode de Gale-Shapley pour minimiser le délai d'inférence DNN, appelé GAME-MIND dans ce qui suit. Le cadre GAME-MIND serait plus adapté aux systèmes dynamiques à grande échelle et plus résilient au problème du point de défaillance unique que l'approche d'optimisation centralisée.

1.5.1 Formulation de l'affectation des couches de DNN comme un jeu d'appariement

Dans un jeu d'appariement, chaque agent d'un ensemble est associé à un agent différent de l'autre ensemble. Pour un appariement donné, si aucun agent ne préfère être associé à un autre agent que son appariement actuel, l'appariement est considéré comme stable [3]. Plus précisément, dans un jeu d'appariement, chaque agent d'un ensemble a une préférence pour ses partenaires potentiels, chaque agent classant effectivement les agents de l'autre ensemble. Ces relations de préférence sont établies sur la base de fonctions d'utilité, comme cela sera précisé ultérieurement. Dans le contexte de l'inférence collaborative DNN, la théorie des jeux d'appariement peut être utilisée pour résoudre le problème de l'attribution de couches DNN aux UAV. L'objectif est d'attribuer les couches DNN de manière à minimiser le délai d'inférence total tout en tenant compte de la dépendance entre les différentes couches DNN, des capacités de calcul limitées et de la connectivité des UAV.

Un appariement μ est une fonction $\mu : \mathcal{R} \times \mathcal{L} \to \mathcal{U}$ qui attribue à un UAV particulier $k \in \mathcal{U}$ le traitement de chaque couche DNN $l \in \mathcal{L}$ de la requête d'inférence $r \in \mathcal{R}$. De cette façon, chaque couche DNN est attribuée à un seul UAV, et chaque drone peut gérer plusieurs couches DNN en fonction de sa capacité et de son quota de traitement. Étant donné que les couches DNN doivent être traitées de manière séquentielle (c'est-à-dire que la couche DNN $l+1$ de toute requête d'inférence ne peut être traitée qu'après l'exécution de la couche l), nous pouvons décomposer les attributions de couches DNN en L phases de correspondance, chacune nécessitant l'attribution d'une couche DNN correspondante de toutes les requêtes d'inférence de $\mathcal{R}$ aux UAV appropriés pour traitement. Pour plus de commodité, nous désignons la couche DNN l de la demande d'inférence r par (r, l) dans ce qui suit.

Dans la conception basée sur les jeux d'appariement, les fonctions d'utilité sont utilisées pour capturer les préférences des agents d'un ensemble envers leurs partenaires de l'autre ensemble. Pour prendre la décision d'assigner la couche DNN (r, l) à un certain UAV k, nous devons spécifier le

UAV traitant la couche DNN précédente $(r, l-1)$, sauf si nous considérons la première couche. Plus précisément, supposons que le UAV i traite la couche DNN $(r, l-1)$ si $l > 1$ ou que le UAV i est celui qui initie la requête d'inférence r si $l = 1$. Ensuite, nous notons la fonction d'utilité $\nabla_{(i)}^{(r,l)}(k)$ de la couche DNN (r, l) pour le UAV k, tandis que la fonction d'utilité du UAV k pour la couche DNN (r, l) est notée $\nabla'_{(i,k)}(r, l)$. Étant donné ces utilités, nous pouvons dire que la couche DNN considérée (r, l) préfère le drone k_1 au UAV k_2 si $\nabla_{(i)}^{(r,l)}(k_1) > \nabla_{(i)}^{(r,l)}(k_2)$ et cette préférence est notée $k_1 \succ_{(r,l)} k_2$. De même, le UAV k préfère la couche DNN (r_1, l) au UAV (r_2, l) si $\nabla'_{(i,k)}(r_1, l) > \nabla'_{(i,k)}(r_2, l)$ et cette préférence est exprimée par $(r_1, l) \succ_k (r_2, l)$. Nous définissons maintenant la fonction d'utilité $\nabla_{(i)}^{(r,l)}(k)$ comme

$$\nabla_{(i)}^{(r,l)}(k) = -\left(\frac{\rho_{r,(l-1),l}}{\alpha_{(i,k)}^{(r,l)}.\delta_{(i,k)}} + \frac{\gamma_{r,l}}{\beta_k^{(r,l)}.S_k}\right), \tag{1.19}$$

où le premier terme entre parenthèses estime le délai de transmission des données de sortie associées à la couche DNN $(r, l-1)$ sur la liaison UAV (i, k) tandis que le second terme entre parenthèses estime le délai de calcul; les facteurs de partage de débit et de calcul, notés $\alpha_{(i,k)}^{(r,l)}$ et $\beta_k^{(r,l)}$, peuvent être calculés respectivement sur la base des fractions de données ou de charge de calcul partageant le débit de liaison UAV correspondant ou la capacité de calcul du UAV comme suit:

$$\alpha_{(i,k)}^{(r,l)} = \frac{\rho_{r,(l-1),l}.\lambda_{i,k}}{\rho_{r,(l-1),l} + \sum_{r'=1}^{R}\sum_{l'=2}^{l} q_{r',(l'-1),i}.q_{r',l',k}.\rho_{r',(l'-1),l'}}, \tag{1.20}$$

$$\beta_k^{(r,l)} = \frac{\gamma_{r,l}}{\gamma_{r,l} + \sum_{r'=1}^{R}\sum_{l'=1}^{l} .q_{r',l',k}.\gamma_{r',l'}}, \tag{1.21}$$

Où (r', l') représente la couche DNN potentiellement traitée par le UAV k. En définissant la fonction d'utilité en fonction du délai total de transmission et de calcul des données, notre conception basée sur la correspondance favorise l'attribution de couches DNN avec un délai d'inférence plus court, ce qui est précisément notre objectif.

De même, la fonction d'utilité du drone k pour la couche DNN (r, l) peut être définie comme suit:

$$\nabla'_{(i,k)}(r, l) = -\frac{\gamma_{r,l}}{\beta_k^{(r,l)}.S_k}. \tag{1.22}$$

Algorithm 1.2. Jeu d'appariement pour l'attribution des couches DNN afin de minimiser le délai d'inférence (GAME-MIND)

Input: N, $\mathcal{R}$, $\mathcal{L}$, $\mathcal{U}$.
Discovery
- Chaque UAV $i \in \mathcal{U}$ identifie l'ensemble de ses voisins à un saut (appelés aides potentielles) $\mathcal{U}_i$ en se basant sur les informations de connectivité, en s'incluant lui-même dans cet ensemble.

Initialisation:
- Toutes les couches (r, l), $r \in \mathcal{R}, l \in \mathcal{L}$ sont libres (non assignées)
- Chaque UAV $k \in \mathcal{U}$ possède une liste vide de couches DNN acceptées : $\mathcal{A}_k = \emptyset, k \in \mathcal{U}$
- Soit $i_{r,l}$ l'UAV qui traite la couche DNN $(r, l-1)$ si $l > 1$, ou $i_{r,l}$ l'UAV initiant la requête d'inférence (r, l) si $l = 1$

for $l = 1 : L$ **do**
  while Il existe une couche DNN $(r, l), \forall r \in \mathcal{R}$ non acceptée et il existe des aides potentielles à un saut pouvant être sollicitées **do**
    for $r \in \mathcal{R}$ **do**
      Pour chaque couche DNN (r, l):
      - L'UAV $i_{r,l}$ établit les relations de préférence de ses aides potentielles à un saut $\mathcal{U}_{i_{r,l}}$ en se basant sur (1.19)
      - Trouver $k^* = \arg\max_{k \in \mathcal{U}_{i_{r,l}}} \nabla^{(r,l)}_{(i_{r,l})}(k)$
      - L'UAV $i_{r,l}$ envoie une requête de calcul ou d'appariement à l'UAV k^* pour exécuter la couche (r, l) en fixant $R_{(r,l)\to k^*} = 1$
    end for
    Pour chaque UAV $k \in \mathcal{U}$:
    - Mettre à jour l'ensemble des couches DNN en demande: $\mathcal{N}^{req}_k := \{(r, l) : R_{(r,l)\to k} = 1\}$
    if $|\mathcal{N}^{req}_k| + |\mathcal{A}_k| \leq \zeta_{th,k}$ **then**
      - L'UAV k accepte toutes les requêtes dans $\mathcal{N}^{req}_k$
      - Mettre à jour $\mathcal{A}_k := \mathcal{A}_k \cup \mathcal{N}^{req}_k$
    else
      - Mettre à jour l'ensemble des couches DNN en requête: $\mathcal{N}^{req}_k := \mathcal{A}_k \cup \mathcal{N}^{req}_k$
      - Choisir $\zeta_{th,k}$ couches DNN avec la plus grande utilité $\nabla'_{(i_{r,l},k)}(r, l)$ parmi celles dans $\mathcal{N}^{req}_k$ et les assigner à $\mathcal{A}_k$
      - Rejeter les couches DNN restantes et mettre à jour: $\mathcal{N}^{rej}_k := \mathcal{N}^{req}_k \backslash \mathcal{A}_k$
      - Pour chaque couche DNN $(r, l) \in \mathcal{N}^{rej}_k$, l'UAV k envoie une notification de refus à l'UAV demandeur $i_{r,l}$
      - À la réception de la notification de refus de l'UAV k, chaque UAV demandeur $i_{r,l}$ supprime l'UAV k de son ensemble d'aide potentielles: $\mathcal{U}_{i_{r,l}} := \mathcal{U}_{i_{r,l}} \backslash \{k\}$
    end if
  end while
end for
Retourner: Ensembles de couches DNN traitées par différents UAVs $\mathcal{A}_k$, $\forall k \in \mathcal{U}$.

La fonction d'utilité $\nabla'_{(i,k)}(r, l)$ fournit une mesure quantitative de l'appariement des UAV avec les couches DNN des requêtes d'inférence. Là encore, la ressource de calcul de chaque UAV est supposée être allouée proportionnellement à la demande de calcul des couches DNN attribuées, garantissant ainsi une répartition équitable des ressources. Étant donné un UAV k d'une capacité

de calcul totale S_k, la ressource de calcul attribuée à chaque couche (r, l) est supposée être basée sur sa charge de calcul $\gamma_{r,l}$ par rapport à la charge de travail totale de toutes les couches DNN attribuées au UAV k. De plus, un UAV n'acceptera des couches DNN supplémentaires que si le nombre total de couches DNN attribuées ne dépasse pas son quota de traitement prédéfini $\zeta_{th,k}$, évitant ainsi tout délai de traitement excessif. Nous fournissons maintenant une définition plus formelle de la stabilité de l'appariement.

Définition 1. Une solution d'affectation ou un appariement des couches de DNN aux UAV sera dite instable s'il existe deux couches de DNN (r, l_1) et (r, l_2) qui sont affectées respectivement aux UAV k_1 et k_2, alors que la couche de DNN (r, l_2) préfère l'UAV k_1 à l'UAV k_2, et que l'UAV k_1 préfère la couche de DNN (r, l_2) à la couche (r, l_1).

L'appariement mentionné dans la définition ci-dessus est instable, car il est possible d'améliorer la solution d'affectation en associant la couche de DNN (r, l_2) à l'UAV k_1 et en réaffectant la couche de DNN (r, l_1) à un autre UAV disposant encore d'un quota disponible. Un appariement est alors dit stable s'il n'existe aucun couple d'affectation/appartement tel que décrit dans la définition précédente. Fait intéressant, il existe une stratégie d'appariement efficace, appelée procédure d" acceptation différée' (deferred acceptance) [3], qui peut être utilisée pour concevoir un algorithme d'appariement, aboutissant à une solution stable et efficace. Cette approche est effectivement adoptée dans notre conception présentée ci-après.

L'Algorithme 1.2 décrit notre processus proposé, guidé par l'utilité, pour l'affectation des couches de DNN aux UAV, fondé sur la théorie des jeux d'appariement. Initialement, toutes les couches de DNN sont marquées comme non affectées, et chaque UAV k commence avec un ensemble vide de couches acceptées, noté $\mathcal{A}_k$. L'algorithme fonctionne de manière itérative et s'exécute sur L phases d'appariement, correspondant aux différentes couches du DNN. À chaque phase d'appariement, pour chaque couche libre (r, l), l'UAV représentant $i_{r,l}$ identifie, parmi les UAV voisins à une seule liaison (one-hop), l'UAV k qui maximise l'utilité définie dans (1.19) et lui envoie une requête d'appariement. Pour chaque UAV k, après réception de toutes les requêtes d'appariement dans une phase donnée, il vérifie si l'acceptation de toutes les nouvelles requêtes contenues dans $\mathcal{N}_k^{req}$ permet encore de respecter sa contrainte de traitement, avec un maximum de $\zeta_{th,k}$ couches de DNN acceptées. Si cette contrainte est respectée, l'UAV k accepte et ajoute toutes les couches de DNN de $\mathcal{N}_k^{req}$ à $\mathcal{A}_k$. Dans le cas contraire, l'UAV k accepte uniquement exactement $\zeta_{th,k}$ couches de DNN qu'il préfère

Table 1.1: Configuration de la simulation

Paramètres	Valeur
Zone réseau	500mx500m
Altitude des UAV	100 m
Bande passante du canal ω	15 KHz
Puissance d'émission des UAV	20 dBm
Densité de puissance du bruit N_0	-174 dBm/Hz
Nombre d'UAV N	10, 15, 20
Nombre de couches DNN L (VGG16)	16
Nombre de requêtes d'inférence R	[1:1:20]
Image d'entrée	595x326 RGB
Capacité de calcul S_i	[10:5:20] GFLOPs
Nombre de tours	10

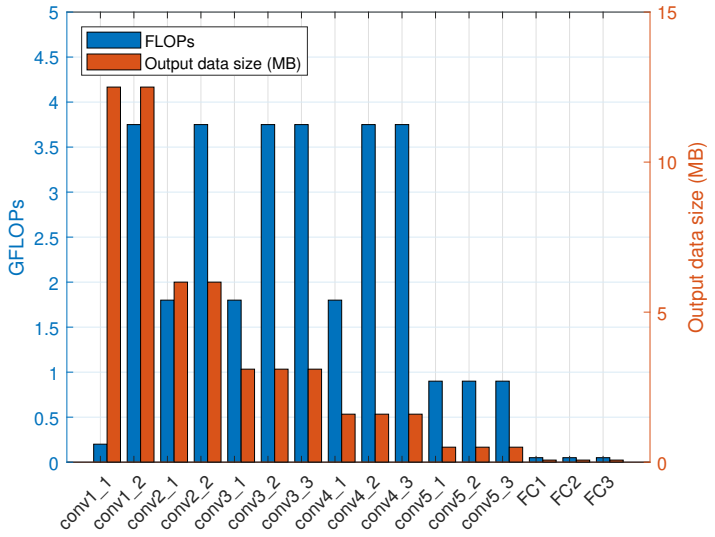

Figure 1.2: FLOPS et taille des données de sortie par couche de VGG16

le plus selon l'utilité exprimée dans (1.22), met à jour $\mathcal{A}_k$, et rejette les couches restantes. Pour chaque couche de DNN $(r, l) \in \mathcal{N}_k^{rej}$ rejetée, l'UAV k est retiré de l'ensemble des UAV potentiels accessibles en une seule liaison, qui seront pris en compte dans les futures tentatives d'appariement. Cette procédure itérative se poursuit pour toutes les couches de DNN jusqu'à ce que chaque couche soit soit acceptée par un UAV, soit rejetée par tous les UAV voisins potentiels.

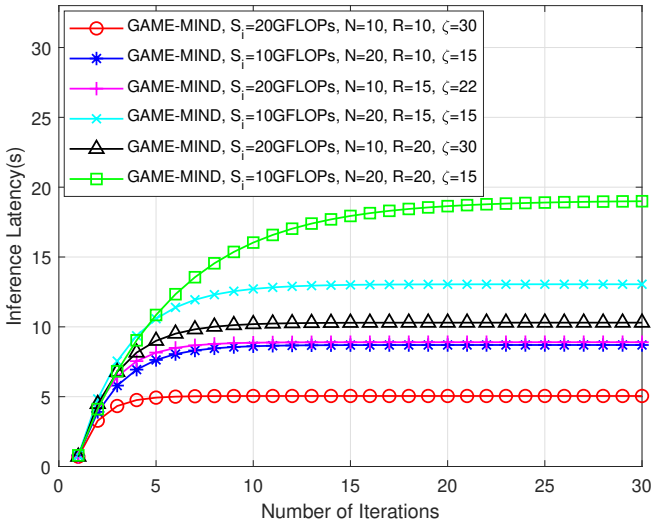

Figure 1.3: Latence d'inférence de GAME-MIND en fonction de différentes tailles de réseau, capacités de calcul des UAV et quotas des UAV selon les itérations d'appariement

1.6 Résultats numériques

Dans cette section, nous évaluons les performances des algorithmes LARA et GAME-MIND proposés. Les paramètres de simulation sont résumés dans le tableau 1.1. Nous plaçons aléatoirement les UAV sur une zone réseau de 500 m x 500 m, et deux UAV dont la distance inter-distance est inférieure à 50 m sont supposés connectés. Les gains du canal de puissance sont simulés en tenant compte de la perte de trajet, où la perte de trajet pour la liaison (i, k) est supposée être $PL_{ik} = PL_o d_{ik}^{-2}$, d_{ik} étant la distance entre les UAV i et k et PL_0 la perte de trajet à une distance de référence de 1 m. Le DNN considéré est le VGG16, dont les besoins de calcul en FLOPS et les données associées aux différentes couches sont présentés dans la figure 1.2. Des simulations ont été réalisées à plusieurs reprises avec différents facteurs aléatoires, notamment le placement aléatoire des UAV et les demandes d'inférence. Les résultats présentés dans cette section sont la moyenne de plusieurs simulations indépendantes (placements de UAV et requêtes d'inférence). Nous avons fixé le quota de traitement de chaque UAV (c'est-à-dire le nombre maximal de couches DNN acceptées) pour qu'il soit identique pour tous les UAV, et nous désignons ce paramètre commun par ζ dans la suite. Nous fixerons également la capacité de calcul maximale de chaque UAV (S_i) pour qu'elle soit identique afin d'obtenir les résultats présentés dans chaque figure.

La figure 1.3 illustre l'impact de la capacité de calcul (S_i), de la taille du réseau (N), du nombre de requêtes d'inférence (R) et du quota de traitement des UAV (ζ) sur la convergence de l'algorithme GAME-MIND. Le scénario avec ($S_i = 20$ GFLOPs, $N = 10$, $R = 10$, $\zeta = 30$) présente la latence d'inférence la plus faible à la convergence, environ 5s sur 10 itérations correspondantes, car les UAV avec une capacité de calcul plus élevée et des quotas plus importants peuvent traiter davantage de couches DNN localement et pour les UAV proches, minimisant ainsi la surcharge de déchargement et accélérant la convergence. En revanche, le paramètre avec ($S_i = 10$ GFLOPs, $N = 20$, $R = 10$, $\zeta = 15$) atteint une latence légèrement plus élevée (environ 7 s) à la convergence puisque, malgré la taille plus grande du réseau, les UAV avec une capacité de calcul plus faible et des quotas plus petits déchargent les tâches plus fréquemment, augmentant le délai de communication et le nombre de requêtes correspondantes.

Lorsque le nombre de requêtes d'inférence augmente ($R = 15, 20$), la latence d'inférence s'accroît en raison de la demande de traitement plus élevée. Pour les configurations considérées avec ($S_i = 20$ GFLOPs, $N = 10$, $R = 15$, $\zeta = 22$) et ($S_i = 20$ GFLOPs, $N = 10$, $R = 20$, $\zeta = 30$), on observe que même avec une capacité de calcul plus élevée, l'augmentation de R entraîne une hausse progressive de la latence d'inférence ainsi que du temps de convergence. Dans deux autres scénarios avec ($S_i = 10$ GFLOPs, $N = 20$, $R = 15$, $\zeta = 15$) et ($S_i = 10$ GFLOPs, $N = 20$, $R = 20$, $\zeta = 15$), où les UAV disposent d'une capacité de calcul plus faible ($S_i = 10$ GFLOPs), le système peut rencontrer des difficultés à traiter un grand nombre de requêtes d'inférence, ce qui entraîne une latence d'inférence élevée. Le quota plus faible ($\zeta = 15$) dans ces cas oblige les UAV à externaliser les tâches plus fréquemment, ce qui accroît la congestion et les délais de communication, tandis que la capacité de calcul limitée ($S_i = 10$ GFLOPs) engendre un délai de calcul important. En revanche, un quota plus élevé ($\zeta = 30$) réduit le besoin d'externalisation des calculs et améliore donc le temps de convergence. De plus, un réseau de UAV plus étendu ($N = 20$) améliore les possibilités d'externalisation des tâches, mais ne conduit pas nécessairement à une latence d'inférence plus faible lorsque les UAV disposent de ressources limitées et doivent traiter un grand nombre de requêtes ($R = 15, 20$), en particulier si leur capacité de calcul est restreinte. Ainsi, un équilibre entre la capacité de calcul, la taille du réseau et le quota de traitement est essentiel pour assurer une convergence rapide et une faible latence d'inférence avec l'approche GAME-MIND.

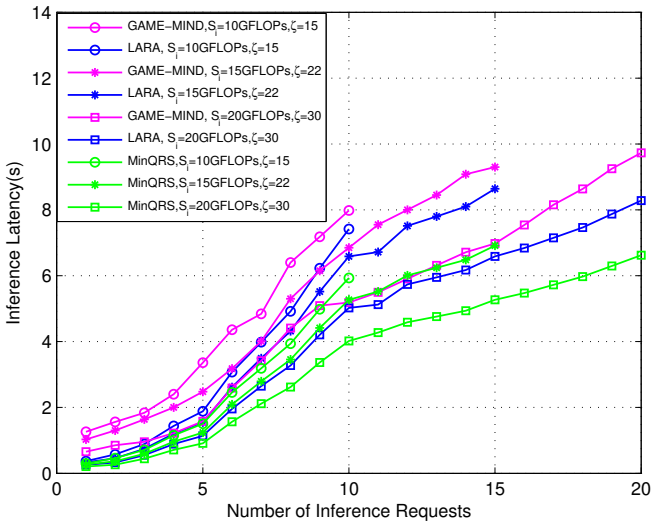

Figure 1.4: Latence d'inférence de GAME-MIND, LARA et MinQRS en fonction du nombre de requêtes d'inférence DNN pour $N = 10$

Dans la Figure 1.4, nous comparons la latence d'inférence de GAME-MIND, LARA et du schéma de référence appelé MinQRS pour un nombre variable de requêtes d'inférence, sous différents paramètres de configuration. Plus précisément, la méthode de référence MinQRS résout le problème global **P** à l'aide d'un solveur sans le décomposer en sous-problèmes comme le fait LARA. Malgré une complexité de calcul élevée, MinQRS fournit une solution quasi-optimale, pouvant servir de référence pour évaluer d'autres algorithmes à plus faible complexité. Comme on peut le constater, dans le cas où $S_i = 20$ GFLOPs, la latence d'inférence obtenue avec GAME-MIND reste très proche de celle de LARA et MinQRS, pour $S_i = 15$ GFLOPs et $S_i = 10$ GFLOPs, lorsque le nombre de requêtes d'inférence augmente de 1 à 15. Cette tendance suggère que, lorsque le nombre de requêtes d'inférence reste relativement faible, les capacités de calcul des UAV sont suffisantes pour traiter efficacement la charge de travail dans les trois algorithmes (LARA, MinQRS et GAME-MIND). Comme la demande de calcul reste limitée dans ces cas, la majorité des couches DNN sont traitées soit localement par les UAV sources, soit affectées à des UAV proches ; ainsi, le délai de communication ne contribue pas de manière significative à la latence totale d'inférence pour les trois algorithmes. Cependant, lorsque le nombre de requêtes dépasse 15, une divergence de performance en termes de latence commence à apparaître entre les trois approches. En particulier, GAME-MIND présente une latence d'inférence légèrement plus élevée par rapport à LARA et MinQRS dans ce régime. Cela s'explique par la nature décentralisée des décisions d'appariement prises par GAME-MIND pour l'affectation des couches DNN, qui peuvent être moins efficaces que

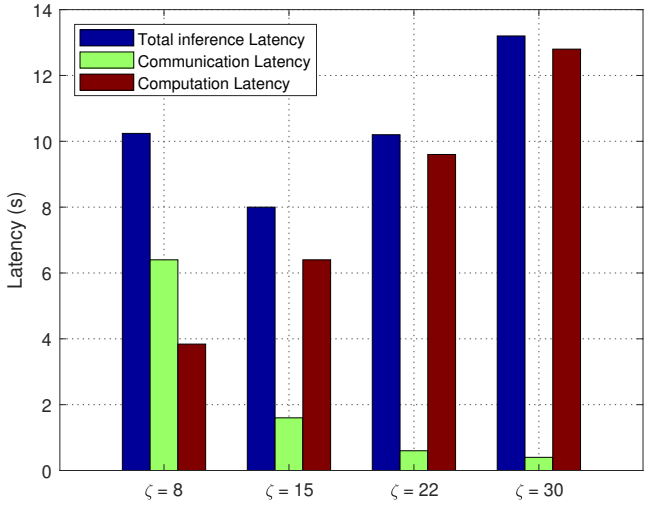

Figure 1.5: Latences d'inférence, de communication et de calcul de GAME-MIND pour différentes valeurs de quota de traitement ζ, avec $R = 10$, $N = 10$ et $S_i = 10$

l'approche d'optimisation centralisée dans des contextes de charge de calcul élevée. Lorsque les UAV disposent de capacités de calcul plus élevées ($S_i = 20$ GFLOPs), les trois algorithmes affichent une meilleure latence d'inférence, mais LARA et MinQRS surpassent encore GAME-MIND dès lors que le nombre de requêtes devient suffisamment important. Bien que LARA et MinQRS permettent une latence plus faible que GAME-MIND, leur nature centralisée soulève des enjeux de passage à l'échelle et de robustesse dans des réseaux UAV dynamiques et de grande taille. GAME-MIND, malgré une latence plus élevée, offre une approche plus évolutive et flexible, le rendant adapté aux opérations décentralisées et autonomes des UAV.

La Figure 1.5 illustre l'impact du quota de traitement ζ des UAV sur les latences de communication, de calcul, et d'inférence totale dans le cadre de l'algorithme GAME-MIND. Lorsque ζ augmente de 8 à 30, la latence de communication diminue, car avec un quota de traitement plus élevé, les UAV peuvent accepter et traiter davantage de couches de DNN localement, réduisant ainsi le besoin d'externalisation des tâches vers des UAV plus éloignés. Cependant, cela entraîne une augmentation de la latence de calcul, car un plus petit nombre d'UAV prend en charge un plus grand nombre de couches, ce qui accroît la charge de traitement et prolonge le temps d'exécution. Pour des valeurs de quota faibles (par exemple, $\zeta = 8$), les UAV atteignent rapidement leurs limites autorisées, ce qui les oblige à externaliser les calculs et à transmettre les données vers leurs UAV

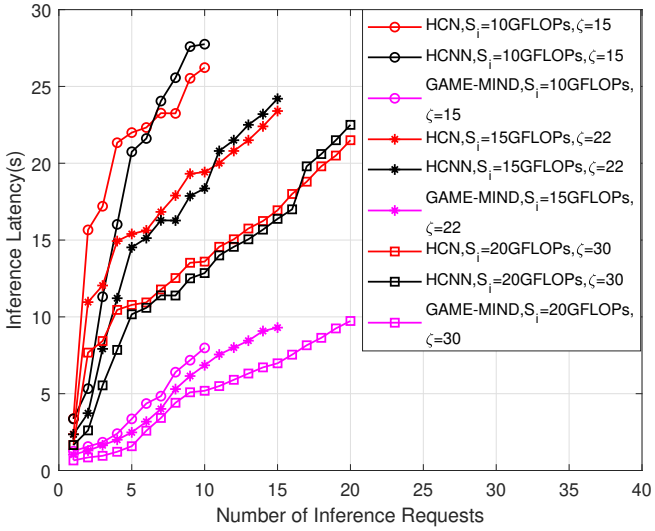

Figure 1.6: Latence d'inférence de GAME-MIND et des heuristiques HCN et HCNN en fonction du nombre de requêtes d'inférence pour $N = 10$

voisins, entraînant ainsi une latence de communication plus élevée mais une latence de calcul relativement faible. À l'inverse, avec des quotas plus élevés (par exemple, $\zeta = 30$), les UAV prennent en charge davantage de couches de DNN au lieu de les externaliser, ce qui réduit la latence de communication mais augmente considérablement la latence de calcul, menant à la latence d'inférence totale la plus élevée (environ 14 secondes). Ces résultats suggèrent que des valeurs de quota trop faibles provoquent des goulets d'étranglement au niveau des communications, tandis que des quotas trop élevés entraînent une surcharge computationnelle et donc une latence de calcul excessive. Un bon compromis entre communication et calcul peut être atteint avec $\zeta = 15$, ce qui permet d'obtenir la meilleure latence d'inférence pour les paramètres considérés.

La Figure 1.6 présente une étude comparative entre GAME-MIND et deux approches heuristiques, HCN (High Computation Neighbor) et HCNN (High Computation Nearest Neighbor). Dans HCN, chaque UAV source transmet simplement sa requête d'inférence au voisin immédiat disposant de la plus grande capacité de calcul, dans le but de réduire le délai de calcul et la latence d'inférence. Toutefois, cette approche ne prend pas en compte la distance entre l'UAV source et ses voisins, ce qui peut entraîner un délai de communication élevé pour les données externalisées, et par conséquent une latence d'inférence importante. Dans la méthode de référence HCNN, l'UAV source transmet sa requête d'inférence au voisin le plus proche ayant la plus grande capacité de calcul. On observe qu'à partir de six requêtes d'inférence, la latence d'inférence induite par HCNN devient supérieure

à celle de HCN. Cela s'explique probablement par une insuffisance des ressources de calcul dans la proximité des UAV sources. Il est important de noter que GAME-MIND surpasse systématiquement les deux heuristiques, en maintenant une latence d'inférence plus faible dans l'ensemble des scénarios étudiés. En particulier, pour des configurations à faible capacité de calcul ($S_i = 10$ GFLOPs), HCN et HCNN connaissent une augmentation significative de la latence d'inférence à mesure que le nombre de requêtes augmente, avec HCN affichant les pires performances. La forte hausse de la latence d'inférence pour HCN indique une allocation inefficace des charges de travail et une mauvaise utilisation des ressources. HCNN offre de meilleures performances que HCN grâce à un mécanisme d'externalisation plus adapté, mais reste inférieur à GAME-MIND, notamment en présence de charges élevées.

On peut observer que GAME-MIND atteint une latence nettement inférieure à celle des deux approches de référence. La latence de GAME-MIND demeure considérablement plus faible, en particulier lorsque les UAV disposent d'une capacité de calcul plus élevée ($S_i = 20$ GFLOPs). Pour $S_i = 20$ GFLOPs, GAME-MIND présente les meilleures performances, avec une augmentation progressive et maîtrisée de la latence à mesure que le nombre de requêtes d'inférence augmente, ce qui met en évidence l'efficacité de GAME-MIND dans la gestion équilibrée des ressources de calcul et de communication. Cette comparaison confirme que GAME-MIND est plus évolutif et plus robuste que les approches heuristiques, notamment dans les scénarios à forte charge.

Chapter 2

Introduction

2.1 Background and Motivation

With the recent advancement in Internet of Things (IoT), mobile applications, and emerging technologies such as augmented reality, autonomous systems, and deep learning, there is an increasing demand for low-latency, high-performance computing and wireless systems. Traditional cloud computing turned out to be struggling to meet these demands due to its dependence on centralized data centers, which can introduce significant communication latency. To address these challenges, edge computing has emerged as a computing paradigm that deploys computational resources closer to the data sources such as end devices or remote mission specific data collecting sensors. In this way, edge computing paradigm significantly reduces response time and improves the overall system performances enabling real-time applications. However, limited computational capacity, energy constraints, and resource management challenges are among the critical challenges for edge computing devices to handle complex and resource-intensive tasks. To overcome these limitations, computation task offloading has been proposed, allowing end devices to offload heavy computational tasks to nearby computing rich devices or servers, thereby improving efficiency, prolonging device battery life, and optimizing the overall performance of real-time applications.

However, computation task offloading design requires joint consideration of multiple factors, such as network conditions, energy efficiency, security risks, workload distribution, and computational resource availability. The dynamic nature of network bandwidth, workload demands, and mobility

of devices makes the offloading process further complicated. Additionally, energy efficiency remains a major challenge, particularly for battery-powered IoT and mobile devices, which must balance computational performance with energy consumption.

To support the quality-of-service demands of data-intensive mobile applications, a range of Mobile Edge Computing (MEC) solutions have been introduced. However, deploying edge servers is not always feasible in challenging environments such as remote regions, mountainous terrains, or areas affected by natural disasters. In such cases, UAVs offer a promising alternative. Provided that line-of-sight (LoS) communication is achievable, UAVs can facilitate task offloading and enhance download efficiency. Since UAV-assisted MEC is still a developing research area, there has been limited in-depth exploration so far. For instance, authors in [1] highlighted several critical scenarios—such as disaster zones, conflict areas, and high-demand locations—where UAVs equipped with MEC capabilities could offer significant advantages.

Motivated by these challenges, this research aims to develop adaptive, delay and cost-efficient computation offloading strategies for the unmanned aerial vehicle (UAV) based edge computing system, enabling to efficiently serve inference requests of deep neural networks (DNN). With the integration of machine learning (ML) and optimization algorithms, computation offloading decisions can be made dynamically based on real-time network conditions, workload variations, and application specific needs. By designing efficient and novel computation task offloading techniques, this research seeks to improve system performance, reduce DNN inference latency and enable seamless execution of latency-sensitive applications in edge computing ecosystems. Ultimately, this study aims to contribute to the development of robust, scalable, and efficient computation offloading mechanisms that can support the growing demands of modern computing applications, enabling inference processing of DNN inference in the UAV-based wireless environment. In the following, we discuss important design concepts and interactions among different key aspects of the considered system.

2.1.1 UAVs and Edge Computing

Edge computing encompasses the technologies that enable computation to take place at the network edge, handling downstream data on behalf of cloud services and upstream data for IoT and/or mobile applications. In this context, the “edge” refers to any computing and networking resource positioned

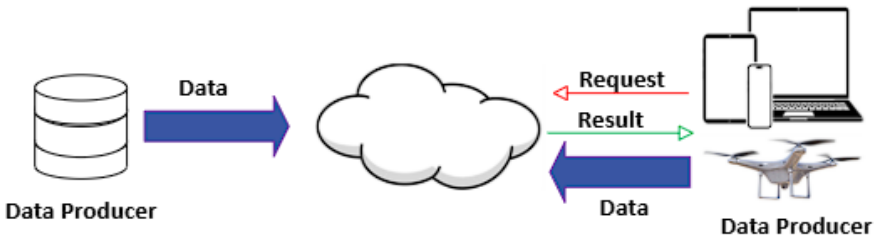

Figure 2.1: Cloud computing paradigm

along the path between data sources and cloud data centers [4]. For instance, a smartphone can act as the edge between wearable devices and the cloud [5], a smart home gateway can serve as the edge between household IoT devices and the cloud, and both micro data centers and cloudlets can function as the edge between mobile devices and the cloud. The core principle of edge computing is that data processing should occur as close as possible to its source. From our perspective, edge computing is largely synonymous with fog computing, but while edge computing is more focused on end devices, fog computing [6] emphasizes the supporting infrastructure. We believe edge computing has the potential to revolutionize society on a scale comparable to that of cloud computing.

Figure 2.1 illustrates the traditional cloud computing architecture. In this setup, data producers collect raw data and transmit it to the cloud, while data consumers send their data and computing requests to the cloud, as shown by the blue solid line. The red dotted line represents the data consumption request from consumers to the cloud, and the green dotted line indicates the response delivered by the cloud. However, this model falls short for delay-sensitive IoT applications. First, the vast volume of data generated at the edge can cause excessive use of bandwidth and computational resources. Second, ensuring data privacy can be challenging in a cloud-centric approach. Lastly, many edge/IoT devices are constrained by limited energy, and wireless communication consumes significant energy. Therefore, offloading some computational tasks to edge devices can offer improved energy efficiency and processing delay.

Figure 2.2 depicts the bidirectional flow of computation in the edge computing framework. In this model, devices at the edge act both as data producers and data consumers. These devices can not only request services and content from the cloud but also handle computing tasks delegated by the cloud. The edge can manage computation offloading, data storage, caching, and processing, in addition to facilitating the transmission of requests and service delivery between the cloud and end

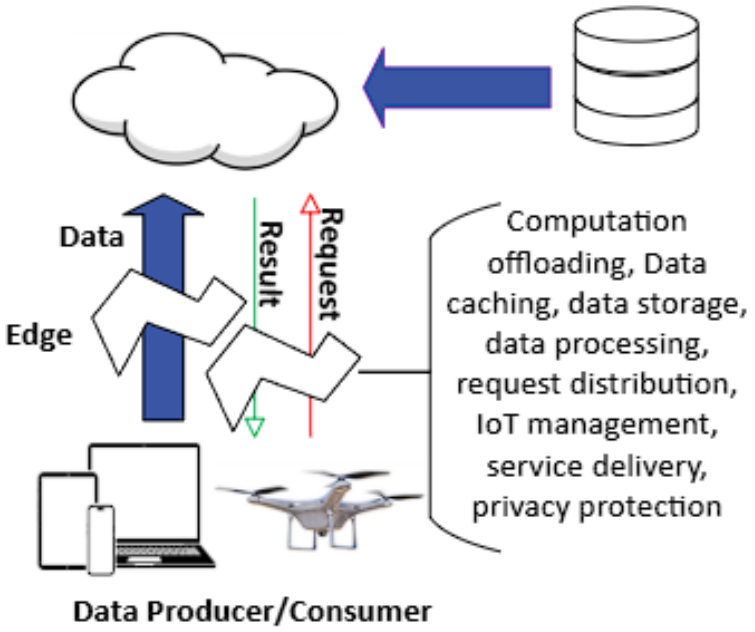

Figure 2.2: Edge computing paradigm

users. Given these responsibilities, the edge infrastructure must be thoughtfully designed to ensure efficient performance, with a strong focus on reliability, security, and privacy protection.

Offloading computing tasks to the cloud has been recognized as an efficient approach to data processing, as the cloud's computing power far surpasses that of edge and IoT devices. However, while data processing speeds have advanced rapidly, network bandwidth has remained relatively stagnant. As the volume of data generated at the edge continues to grow, data transmission latency becomes a major bottleneck in the cloud-based computing model. For instance, a Boeing 787 produces approximately 5 gigabytes of data every second [7], yet the bandwidth between the aircraft and either a satellite or a ground-based station is insufficient for low-delay data transfer. Similarly, an autonomous vehicle generates around 1 gigabyte of data per second, requiring real-time processing to ensure accurate decision-making [8]. If all this data had to be transmitted to the cloud for processing, the response time would be excessively long. Furthermore, the current network bandwidth and reliability would struggle to support a high density of vehicles in each area. To address these challenges, processing data at the edge is essential for reducing response time, improving efficiency and alleviating network congestion.

UAVs are aerial platforms that operate either autonomously or under remote control. Their adaptability, ease of deployment, and relatively low operational costs have led to their widespread

adoption in both civilian and military fields. In military contexts, UAVs are primarily used for tasks such as reconnaissance, surveillance, and precision strikes. The selection of UAV platforms is typically tailored to specific mission requirements, which include factors like operational range, target locations, mission duration, and payload capacity. For real-time video surveillance, multiple UAVs may be deployed cooperatively, increasing the complexity of mission planning and coordination [9].

Although UAV networks exhibit several distinct features, they are often broadly classified as ad hoc networks. However, this classification does not fully capture their nature. Unlike traditional ad hoc networks, UAV network architectures—particularly in multi-UAV deployments—are less thoroughly explored and can take on a variety of topological structures . UAVs may remain stationary, move slowly, or travel at high speeds, depending on the scenario. Moreover, these networks often depend significantly on infrastructure support, which differentiates them from typical ad hoc systems. UAVs can also serve various roles such as clients or servers, depending on the operational context. Another critical aspect is the need for robust link management, as the dynamic movement of multiple UAVs introduces complex connectivity challenges.

2.1.2 Deep Neural Network and Edge Computing

Artificial Intelligence (AI) is the broad field of creating intelligent machines, within which Machine Learning (ML) enables systems to learn from data, and Deep Learning (DL) is a specialized subset of ML that uses deep neural networks to model complex patterns. DNNs have become fundamental to the advancement of artificial intelligence, offering state-of-the-art performance in complex tasks such as image recognition, object detection, speech processing, and autonomous decision-making. These models, characterized by their layered architecture and deep hierarchical learning capabilities, are particularly effective at extracting high-level features from raw data.

The inference phase of DNNs demand substantial computational resources and low-latency processing, especially in time-critical applications. To address the limitations of centralized cloud processing, edge computing has emerged as a powerful solution by enabling data processing at or near the data source. Through edge computing, DNN inference can be performed closer to end devices, reducing the need to transmit data to remote servers and thereby minimizing latency, bandwidth usage, and energy consumption. This is especially valuable in scenarios where real-time analytics and immediate response are essential. One such scenario is the deployment of DNNs in UAVs,

which are increasingly used for applications such as aerial surveillance, search and rescue, precision agriculture, and autonomous navigation. Given their limited onboard computational capacity and energy constraints, UAVs often rely on edge-assisted architectures to offload DNN inference tasks to nearby edge servers or ground stations. This synergy between DNNs, edge computing, and UAV technology enables intelligent aerial systems to operate efficiently in dynamic, resource-constrained environments, ensuring timely and accurate decision-making in mission-critical operations.

2.1.3 Computation Task Offloading

In computation offloading, a mobile device transfers a portion of its tasks to the cloud. This process includes application partitioning, making offloading decisions, and executing tasks in a distributed manner. The goal of computation offloading is to improve performance in terms of processing speed, delay, energy consumption, and enhance the overall efficiency of the system by leveraging external computational capabilities.

In many modern applications, end or mobile devices generate large amounts of data that require significant processing power [10]. However, due to hardware limitations, performing complex computations locally may lead to high energy consumption, slow processing speeds, and increased response time. Computation offloading helps overcome these limitations by allowing devices to delegate tasks to more capable computing nodes. Depending on the architecture, offloading can occur at different levels:

1. Cloud Offloading – Tasks are sent to remote cloud servers with high computational power and storage capacity. This method provides significant processing benefits but can introduce latency due to network delay.

2. Edge Offloading – Tasks are offloaded to nearby edge servers or fog nodes located closer to the end devices. This approach reduces latency and bandwidth usage while improving real-time processing capabilities.

3. Hybrid Offloading – A combination of cloud and edge offloading, where tasks are dynamically allocated based on network conditions, computational resources, and task requirements.

The offloading decision depends on multiple factors, including network bandwidth, latency requirements, energy constraints, computational complexity, and security concerns. Some offloading mechanisms use AI and machine learning to optimize task distribution dynamically, ensuring efficient resource utilization. For example, in mobile gaming or augmented reality applications, real-time rendering and computation-intensive tasks can be offloaded to edge servers, improving performance and user experience without draining device battery life. Similarly, in autonomous vehicles, AI-driven decision-making processes require substantial computing power, making edge or cloud offloading essential for real-time navigation and object detection. Despite its advantages, computation task offloading poses challenges such as security risks, data privacy concerns, and dynamic network fluctuations. Efficient scheduling, adaptive offloading strategies, and resource management are critical to ensuring seamless and optimized performance in edge computing environments.

A centralized optimization approach for computation task offloading relies on a strategy where a central controller, typically located at an edge server, cloud server, or network gateway, is responsible for making global offloading decisions based on system-wide information. This controller collects real-time data from multiple devices, including their processing power, battery levels, network conditions, and task complexity, and then determines the optimal task allocation using various optimization techniques. These techniques include integer linear programming [11], deep reinforcement learning (DRL) for dynamic environment [12], and metaheuristic algorithms like genetic algorithms and particle swarm optimization for near-optimal solutions [13, 14]. The central controller assigns computational tasks to appropriate computing nodes, such as local edge servers or cloud data centers, aiming to minimize energy consumption, reduce execution delay, and balance workloads across the network. While this approach offers significant advantages, including global efficiency, improved load balancing, and optimized resource utilization, it also presents challenges such as scalability issues, high communication overhead, and vulnerability to a single point of failure. If the central controller fails or becomes a bottleneck, system performance can degrade, making real-time task offloading inefficient. In large-scale or dynamic environments, hybrid or decentralized offloading mechanisms are often integrated to overcome the drawbacks of a purely centralized approach while maintaining optimal computing performance.

A distributed approach for computation task offloading distributes the decision-making process among multiple edge nodes or devices instead of relying on a single central controller. In this approach, offloading decisions are made locally or collaboratively among edge nodes, IoT devices,

and nearby computing resources, reducing dependency on a central orchestrator and improving system scalability, fault tolerance, and adaptability to dynamic network conditions. Each device or node independently evaluates its computational capacity, energy levels, network conditions, and task requirements to determine whether to process the task locally or offload it to a nearby edge server, fog node, or peer device. Unlike centralized approaches, which require a global view of the network, decentralized offloading relies on local knowledge, distributed optimization techniques, and peer-to-peer communication to make real-time, context-aware decisions.

To achieve optimal task allocation, decentralized offloading often employs multi-agent reinforcement learning (MARL) [15], game theory-based models [16], and heuristic algorithms [17] that enable edge nodes to collaborate efficiently. One key advantage of decentralization is enhanced scalability, as the system can accommodate a growing number of devices without creating a computational bottleneck. Additionally, the removal of a single point of failure ensures greater reliability, making this approach suitable for large-scale, dynamic, and mission-critical applications such as vehicular networks, industrial IoT, and disaster response systems. However, challenges include higher coordination overhead, suboptimal decision-making due to limited local knowledge, and potential security risks, as distributed nodes must establish trust and communication reliability. Despite these challenges, decentralized offloading is increasingly favored in modern edge computing environments where real-time adaptability, resilience, and scalability are essential for efficient task execution.

2.2 Literature Review

In this section, we present a review of recent optimization approaches for computation offloading and resource allocation problems. The review is organized based on the techniques employed to address these problems.

2.2.1 Optimization based Computation Offloading

Due to the nonconvex nature of objective functions or constraints, most computation offloading and resource allocation designs involve solving nonconvex optimization problems. Several techniques are employed to convert nonconvex problems into convex ones and to solve these these difficult optimization problems within polynomial time. Common approaches include the linearization technique,

Successive Convex Approximation (SCA) method, and variable substitution methods. Once transformed into convex form, these problems can be effectively addressed using optimization algorithms such as the interior-point methods, or Lagrangian techniques.

In MEC systems, storage capacity, computational power, and available energy are all limited and are considered in the optimization formulations. Sardellitti et al. [18] addressed this by formulating an optimization problem that jointly optimizes both radio and computational resources. To reduce the total energy consumption of users, they introduced an iterative algorithm grounded in the SCA technique. Mao et al. [19] employed flow shop scheduling and convex optimization techniques to minimize latency and energy consumption in a single-user MEC system. The formulated objective function involved both integer variables, representing scheduling decisions, and continuous variables, corresponding to transmission power allocation. Consequently, the problem was categorized as a mixed-integer nonlinear programming (MINLP) problem. Given the high computational cost of an exhaustive search, the authors proposed a low-complexity suboptimal algorithm to solve the underlying problem.

Convex optimization techniques have been extensively applied to minimize energy consumption in various studies [20–22]. In particular, You et al. [20] investigated the resource allocation problem in a multi-user MEC system. They optimized the computation offloading strategy by considering both channel conditions and local computation energy under a time constraint. While the study in [20] provided a detailed modeling analysis and focused on optimizing the offloading ratio for each user, it did not address the joint optimization of radio and computational resources. The aspects not addressed in [20] were further explored by Song et al. [21] where they examined the joint allocation of radio and computational resources in a non-orthogonal MEC environment within heterogeneous networks. Their approach involved decomposing the original problem into two subproblems: one focused on computational resource allocation, which was formulated as a convex optimization problem, and the other on uplink power control, which was solved using sequential convex programming.

In [22], the authors addressed an optimization problem aimed at enhancing energy efficiency and reducing latency in cache-enabled multi-user MEC systems by optimizing caching, computation, and communication resources. To tackle this problem, they employed a combination of block coordinate descent and convex optimization techniques. Following a similar approach to that in [21], the overall

optimization was divided into two subproblems: one for determining the caching policy and the other for resource allocation. In [23, 24], convex optimization techniques were employed to address multi-objective optimization problems. Yu et al. [23] formulated the combinatorial optimization problem as a system cost minimization task, taking into account both energy consumption and task completion time. They transformed the original problem into a convex form and proposed a distributed algorithm for its solution. Additionally, Guo et al. [24] introduced a dynamic offloading and resource scheduling strategy aimed at minimizing energy consumption while meeting deadline and task-dependency constraints.

The Alternating Direction Method of Multipliers (ADMM) has emerged as a prominent technique for solving optimization problems in MEC systems [25, 26]. Both studies focused on resource allocation strategies aimed at reducing delay and enhancing overall system utility. The algorithms proposed in these works exhibited similar computational complexity. Full Duplex technology offers the potential to enhance spectrum efficiency. In [25], Tan et al. formulated a resource allocation problem within FD-enabled small cell networks incorporating MEC and caching capabilities. To address this optimization problem, they employed the ADMM framework.

Salmani and Davidson [27] investigated uplink communication resource allocation in both binary and partial offloading scenarios. For indivisible tasks, they designed a customized greedy search algorithm, while for divisible tasks, they developed a low-complexity solution approach. To overcome the limitations associated with traditional schemes such as Time Division Multiple Access (TDMA) and Orthogonal Frequency Division Multiple Access (OFDMA), the authors proposed a "full" multiple access strategy. The computational complexity of their algorithm was $\mathcal{O}(M \log M)$, where M denotes the number of users. Alkhalaileh et al. [28] introduced an optimization approach aimed at minimizing both monetary cost and energy consumption. The problem was modeled as a mixed-integer linear programming (MILP) formulation and applied within a Mobile Edge Cloud Computing (MECC) context. To assess the effectiveness of their solution, the authors conducted two types of evaluations: practical experiments in an actual MECC environment and simulations using a synthetic dataset. Although they asserted that the proposed algorithm has low time complexity, no formal complexity analysis was provided.

Zhao et al. [29] formally introduced the Offloading of Dependent Tasks with Service Caching (ODT-SC) problem and demonstrated that it is a computationally hard problem with no algo-

rithm capable of achieving a constant approximation ratio. To address this, the authors develop an efficient solution based on convex programming, referred to as the CP algorithm. Additionally, they explored a special scenario involving a homogeneous MEC environment and propose a Favorite Successor (FS) algorithm, which achieves a competitive ratio of $\mathcal{O}(1)$. Through extensive simulations using real-world Google data traces, the results show that the proposed algorithms can reduce application completion times by approximately 21-47 percent when compared to other existing methods. Wu et al. [30] explored secure computation offloading in a wireless-powered mobile edge computing (MEC) system. The goal is to maximize the secrecy energy efficiency (SEE) by jointly optimizing several factors: transmission power used for wireless power transfer (WPT), task offloading, and jamming; time allocation between WPT and offloading; and the division of computation tasks. Given the complex and non-convex nature of the problem, the authors propose an effective two-layer optimization algorithm. This solution incorporates variable substitution, successive convex approximation, and the Dinkelbach method to make the problem more manageable. Simulation results confirm that the proposed approach significantly improves SEE compared to other benchmark methods.

Younis et al. [31] presented the Energy-Latency-aware Task Offloading and Approximate Computing (ETORS) problem, which focuses on optimizing the trade-off between energy consumption and task execution latency. Given the mixed-integer complexity of the problem, the authors adopt the Dual-Decomposition Method (DDM) to break it down into three interconnected subproblems: Task-Offloading Decision (TOD) to determine the optimal execution location for tasks, CPU Frequency Scaling (CFS) to adjust processor speed for efficient energy usage, and Quality of Computation Control (QoCC) to balance computational accuracy with resource consumption. The resultant problem was solved using convex optimization techniques.

2.2.2 Heuristic Computation Offloading Designs

In scenarios where offloading decisions must be made quickly, heuristic methods serve as practical and effective alternatives, even though they may not always yield optimal solutions. One key advantage of heuristic approaches is their ability to integrate well with other techniques, enhancing overall efficiency. In this subsection, we explore several notable applications of heuristic methods in addressing combinatorial optimization problems.

Yang et al. [32] employed a genetic algorithm in conjunction with a partial offloading model to minimize task completion time in mobile cloud computing environments. They introduced an offline heuristic algorithm designed to determine the optimal balance between local and cloud-based resource utilization. Yang et al. [33] developed an algorithm based on an enhanced Krill Herd metaheuristic approach to minimize the combined energy consumption and packet congestion. In [34], the authors tackled the issue of scheduling latency for delay-sensitive tasks with the objective of minimizing overall system cost. Their approach aimed to meet Quality of Service (QoS) requirements for all tasks while optimizing resource usage. They demonstrated that the resulting optimization problem is NP-hard. To address this challenge, they introduced a task scheduling method named two-stage scheduling cost optimization.

In [35], the authors investigated strategies for determining the optimal number of data replicas and their placement within an edge computing system. Their work focused on replica creation and data scheduling for deadline-constrained, dependency-aware workflow jobs. They introduced a replica management framework consisting of two main components: a data replica generator and a workflow-oriented job scheduler. By taking into account factors such as data block popularity and node workload, they proposed dynamic replica creation techniques to estimate the required number of replicas and reduce data access costs while adhering to timing constraints. The data replication and scheduling problem was formulated as an integer programming model, and a fast metaheuristic algorithm was developed to solve it, exhibiting polynomial time complexity. Li et al. [36] proposed a method to minimize download latency in heterogeneous networks equipped with edge computing. The core contribution of their work lies in a strategy that integrates both content prediction and user association. This approach was implemented using heuristic algorithms. Specifically, the content prediction was carried out using a cubic exponential smoothing technique, which is well-suited for forecasting tasks. The second component—dynamic user association—was executed through both rapid and delayed association mechanisms.

A cache-aware task scheduling approach for edge computing was introduced in [37]. The task scheduling problem was represented as a weighted bipartite graph, where the weights were determined based on the data location—whether in a local cache, local disk, or remote storage. The solution involved finding a maximum weight matching between tasks and available resources. In another study, Li et al. [38] proposed an optimal resource allocation strategy aimed at minimizing

the financial expenditure associated with renting edge nodes. Their method was evaluated using a real-world dataset and demonstrated notable cost savings compared to existing solutions.

In [39], the authors explored the optimization of task scheduling for cyber-physical system applications within a fog computing environment. Their primary objective was to reduce the overall execution time. To achieve this, they developed a task scheduling approach based on the moth-flame optimization algorithm, a type of metaheuristic technique. Yang et al. [40] introduced a hybrid approach that combines two offloading algorithms, ASO and Pro-ITGO, to minimize energy consumption while adhering to deadline constraints. Leveraging the strengths of both Edge Computing and Cloud Computing paradigms, they proposed a solution for an integrated architecture.

Mei et al. [41] developed an energy-efficient optimization problem under a delay constraint, taking into account the competition for wireless channels and MEC computing resources. Due to the high complexity of the original problem, the authors introduce a heuristic method that breaks the problem into a series of offloading sub-problems. Each sub-problem focuses on determining the optimal offloading strategy for a fixed set of tasks, which is refined iteratively using the proposed approach. Initially, a full offloading solution is explored and proven to be a convex problem. To address partial offloading, an iterative binary search method is utilized to identify the optimal solution, where resource allocation for both wireless channels and MEC processing frequency is optimized in a cyclical manner. Simulation results show that the proposed approach can reduce energy consumption by up to 14.20 percent while still meeting the delay requirements for all tasks.

2.2.3 Game Theory based Computation Offloading

Game theory explores the decision-making strategies of two or more participants. Widely applied across various fields, including economics, social sciences, engineering, and computer science, game theory is instrumental in solving a diverse range of problems.

Zhang et al. [42] addressed the combined optimization problem of combinatorial optimization and resource allocation in heterogeneous networks using game theory. In [43], a resource scheduling framework for cooperative cloudlets was introduced to minimize operators' costs while maintaining a high-quality user experience in Edge Computing with a centralized controller. The interactions between the cloudlets and the controller were modeled as a two-stage Stackelberg game, aiming to

determine the allocation of physical resources to each cloudlet during the deployment of cooperative cloudlets. Additionally, in [44], the authors proposed a method to enhance response time and resource utilization using Stackelberg game theory. The suggested scheduling framework was designed to predict real-time requests based on historical data. The approach demonstrated potential for broader application, with plans to scale the system for wider use in the future.

Zhu et al. [45] investigated issues related to bandwidth and power allocation, highlighting the problem of unfair resource distribution in existing solutions. To address this, they proposed a solution based on the Nash bargaining game. To mitigate the NP-hardness of the original problem, they transformed the discrete subchannel allocation into a continuous-time variable allocation optimization. This conversion resulted in a standard convex optimization problem. Today, a wide variety of services such as augmented/virtual reality and multimedia are in use, each with distinct resource requirements. To enhance Quality of Service (QoS), it is essential to consider the specific demands of these heterogeneous services. In [46], the authors proposed an algorithm based on coalition game theory to enhance the performance of diverse services in MEC, particularly focusing on reducing latency.

MEC and Non-orthogonal Multiple Access (NOMA) are key technologies in 5G networks, known for enabling low energy consumption and reduced latency. In [47], the authors pointed out that most existing research has primarily concentrated on single-carrier NOMA systems. To address this gap, they introduced a coalition game theory-based approach to solve the combinatorial optimization problem in multicarrier NOMA-enabled MEC environments. In [48], the authors jointly examined a computation offloading strategy, transmission scheduling mechanisms, and pricing models. To capture the dynamics of the packet-label network, they developed a queuing-based model. The primary objective of their work was to minimize the overall delay cost. In a related study, Cui et al. [49] applied a combination of evolutionary game theory and reinforcement learning to tackle multiuser computation offloading challenges in dynamic environments. Their proposed solution aimed at lowering both latency and energy consumption for IoT devices.

One direction of game theory is a game with incomplete information which deals with scenarios where participants lack complete knowledge about their opponents. This approach is particularly well-suited for optimizing offloading decisions in situations where multiple users compete for wireless bandwidth or MEC computational resources without having full awareness of each other's strategies.

The following studies discuss the application of game theory under incomplete information in the context of computation offloading within Edge Computing. Hu et al. [50] addressed the challenge of minimizing energy consumption under deadline constraints in a system with multiple mobile devices and MEC servers with limited capacities. They modeled the problem as a noncooperative game and introduced a game-based computation offloading algorithm called the greedy-pruning algorithm to identify which devices should offload their tasks. In [51], the authors proposed a distributed offloading strategy under incomplete information using a minority game framework, allowing devices to make offloading decisions independently in an edge computing environment. Liwang et al. [52] explored issues such as determining appropriate offloading rates, selecting optimal service providers, and devising pricing strategies in cloud-assisted Vehicle-to-Vehicle (V2V) communication systems. Their work offered solutions applicable to both complete and incomplete information scenarios.

In [53], Ding et al. categorizes End-Edge-Cloud Computing (EECC) into two architectural models based on how user equipment (UEs) can access and interact with the cloud: hierarchical EECC (Hi-EECC) and horizontal EECC (Ho-EECC). In the Hi-EECC model, UEs are limited to offloading tasks to edge servers (ESs); if those ESs run out of resources, they then communicate with the cloud to obtain additional computing capacity. Conversely, in the Ho-EECC setup, UEs have the flexibility to offload tasks directly to either ESs or the cloud. The study models the EECC environment using a potential game framework, where each UE independently aims to minimize its own cost. It explores the optimization of computation offloading strategies and introduces two game-theory-based algorithms tailored for Hi-EECC and Ho-EECC. Comprehensive experiments using real-world datasets are conducted to validate the effectiveness of the proposed methods.

While research on computation offloading design for different MEC settings has been very active in the last few years, collaborative offloading and resource allocation for efficient DNN inference has been quite underexplored. The current thesis aims to address this research gap.

2.3 Research Objectives and Contributions

This thesis focuses on the DNN inference task offloading and resource allocation design in UAV-assisted edge computing networks where our design aims to minimize the DNN inference latency. Two complementary scenarios are explored. In the first scenario, a centralized optimization-based al-

gorithm is developed to enhance resource allocation efficiency and minimize inference latency across UAV nodes. This approach leverages global network information to make optimal task distribution decisions. In the second scenario, a distributed method based on game theory is proposed, enabling UAVs to make autonomous resource allocation and offloading decisions while interacting with each other. This distributed approach supports scalability and adaptability in dynamic environments while balancing performance and fairness among UAVs.

The key contributions of this thesis stems from the development of efficient resource allocation and task offloading strategy based on optimization and game-theoretic techniques, which is presented in Chapter 4. Our design aims to minimize the DNN inference latency while accounting for the dynamic and decentralized nature of UAV-assisted edge computing networks. Our contributions can be summarized as follows:

- We formulate the joint DNN layer assignment and resource allocation problem for the UAV based wireless network that aims to minimize the total inference latency considering constraints on wireless connectivity, processing order of DNN layers, processing quotas, and computational resources at individual UAVs.
- We propose to employ an alternating optimization method to solve the optimization problem where we iteratively optimize the DNN-layer assignment subproblem, computing resource allocation subproblem, and rate allocation subproblem iteratively until convergence. Accordingly, we develop a DNN layer assignment and resource allocation framework (LARA) for efficient and collaborative processing of DNN inference requests.
- We design a novel matching game theory-based algorithm, called GAME-MIND, by modeling the DNN layer allocation problem as a matching game, ensuring efficient distribution of computation tasks among the UAVs. Appropriate utility functions are defined to capture the preferences of DNN layers towards UAVs as well as preferences of UAVs to accept and process DNN layers, considering the inference latency minimization goal and computational resource sharing for DNN layers processed at each UAV.
- Extensive simulations demonstrate the desirable delay performance achieved by LARA and GAME-MIND algorithms. Specifically, the VGG16 model, which is a high-performance CNN model, is considered in performance evaluation of the collaborative inference designs. We compare the performance of GAME-MIND with two heuristic-based methods that offload DNN

layers to the nearest neighboring UAVs. We show that GAME-MIND greatly outperforms the two heuristic-based methods, especially in the high-load settings.

The remainder of this thesis is organized as follows. Chapter 3 presents the background knowledge, which is essential for a better understanding of the main chapter of this thesis. Chapter 4 discusses our proposed LARA and GAME-MIND frameworks. Finally, Chapter 5 concludes the thesis and outlines future research directions.

Chapter 3

Background

In this chapter, we introduce important concepts related to mathematical optimization, deep learning, and matching game theory. First, we present the fundamental principles of mathematical optimization, including definitions, problem formulations, and solution methodologies commonly used in various applications. Next, we provide an overview of deep learning, focusing on its basic architectures, learning paradigms, and its role in solving complex tasks through data-driven approaches. Finally, we explore the basics of matching game theory, highlighting its theoretical foundations and its application to modeling and solving problems involving resource allocation, decision making, and strategic interactions between multiple agents. These theoretical underpinnings serve as the groundwork for the advanced techniques and solutions developed in the later chapter.

3.1 Mathematical Optimization

A mathematical optimization problem is commonly represented as [54]:

$$\min_{\mathbf{y}} \quad f(\mathbf{y}) \tag{3.1}$$

$$\text{subject to} \quad g_i(\mathbf{y}) \leq 0, \quad i = 1, 2, \ldots, n, \tag{3.2}$$

$$w_j(\mathbf{y}) = 0, \quad j = 1, 2, \ldots, m, \tag{3.3}$$

Here, the vector $\mathbf{y} \in \mathbb{R}^n$ is the *optimization variable*, and $f(\mathbf{y})$ is the *objective function* to be minimized. The inequalities $f_i(\mathbf{y}) \leq 0$, for $i = 1, 2, \ldots, n$, and equalities $w_j(\mathbf{y}) = 0$, for $j = 1, 2, \ldots, m$, are the *constraints* of the problem.

Let $\mathcal{D}$ denote the common domain of all functions $f_i(\mathbf{y})$ and $w_j(\mathbf{y})$, for $i = 0, 1, \ldots, n$ and $j = 1, 2, \ldots, m$. The *feasible set* consists of all vectors $\mathbf{y} \in \mathcal{D}$ that satisfy all constraints. A vector $\mathbf{y}^*$ is called an *optimal solution* if it lies within the feasible set and minimizes the objective function $f(\mathbf{y})$. The corresponding value $f(\mathbf{y}^*)$ is referred to as the *optimal value*. If the feasible set is empty, the problem is said to be *infeasible*. In such cases, the optimal value is conventionally taken to be $+\infty$. Note that the phrase "subject to" is often abbreviated as "s.t.".

3.1.1 Convex Optimization

Among the various types of optimization problems, convex optimization problems are particularly noteworthy due to their favorable properties. One key advantage is that for convex problems, any locally optimal solution is also globally optimal. This characteristic makes convex optimization more manageable compared to general optimization problems, which may have multiple local optima. As a result, solving convex problems often involves identifying just a local minimum. Moreover, many subclasses of convex optimization problems are well understood, and efficient solution techniques have been developed. These methods are mature and widely applicable in numerous practical fields.

Below are some essential definitions in convex optimization:

Convex Set: A set S is said to be convex if for any vectors $\mathbf{y}, \mathbf{z} \in S$, and for all $\theta \in [0, 1]$, the following condition holds:

$$\theta\mathbf{y} + (1 - \theta)\mathbf{z} \in S.$$

Convex Function: A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if its domain D is a convex set, and for any $\mathbf{y}, \mathbf{z} \in D$ and $\theta \in [0, 1]$, the following inequality holds:

$$f(\theta\mathbf{y} + (1 - \theta)\mathbf{z}) \leq \theta f(\mathbf{y}) + (1 - \theta)f(\mathbf{z}).$$

Convex Optimization Problem: An optimization problem is convex if it can be written in the following standard form:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{y}) \\ \text{subject to} \quad & g_i(\mathbf{y}) \leq 0, \quad i = 1, \ldots, n, \\ & w_j(\mathbf{y}) = 0, \quad j = 1, \ldots, m, \end{aligned}$$

where $f(\mathbf{y})$ and $g_i(\mathbf{y})$ for $i = 0, \ldots, n$ are convex functions, and each $w_j(\mathbf{y})$ for $j = 1, \ldots, m$ is a linear function.

Such problems are commonly referred to as *convex problems*.

3.1.2 Optimization Algorithms

There is no general-purpose algorithm that can solve nonconvex optimization problems optimally in polynomial time [54]. In contrast, a variety of techniques have been developed for specific types of convex optimization problems that allow solutions to be obtained with guaranteed accuracy and within polynomial time, relative to the problem size [55, 56]. Among these, the interior-point method is regarded as one of the most effective approaches for tackling large-scale convex problems. These methods are widely implemented in modern optimization solvers.

For the sake of brevity, we do not delve into the technical details of these algorithms. Readers interested in a deeper understanding of optimization theory and solution methods are referred to [54, 57], which provide comprehensive and rigorous explanations. Given that many reliable solvers are already available and perform well on commonly encountered convex problems, we utilize these existing tools rather than developing new numerical methods. Specifically, in our work, we use CVX [58] in MATLAB for solving convex problems, with Mosek (academic version) [59] serving as the underlying solver. However, not all problems in our research are convex. For optimization problems that are nonconvex, alternative strategies such as Successive Convex Approximation (SCA), Dinkelbach method, and Dual-Decomposition Method (DDM) could be employed to solve them.

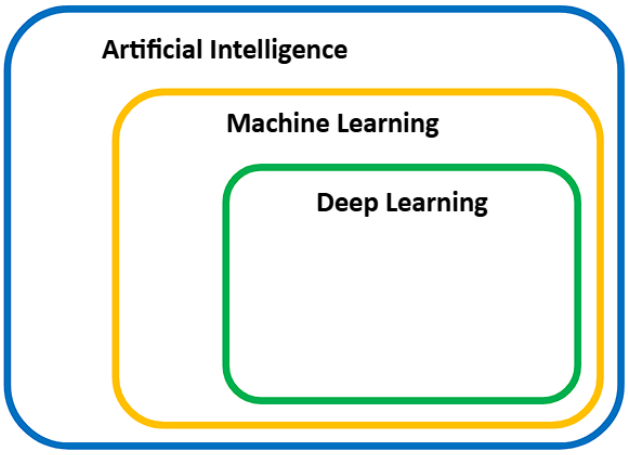

Figure 3.1: Relationship among AI, Machine Learning, and Deep Learning

3.2 Deep Learning

Artificial Intelligence (AI) plays a central role in embedding intelligence into computational systems. In the context of dynamic MEC offloading environments, enabling IoT devices to operate intelligently primarily relies on a focused subset of AI techniques known as machine learning (ML). As a prominent branch of machine learning, deep learning (DL) has shown exceptional capability in processing and analyzing large volumes of data to perform a variety of ML tasks. The success of DL is not only attributed to advancements in deep learning architectures but also to the increasing availability of large datasets and powerful computing resources. There have been an increasing body of research works that employ ML techniques for computation offloading design. In this section, we present several deep learning models that are relevant to our research.

3.2.1 Multilayer Perceptron

Multi-layer Perceptron (MLP) is another name for a modern feedforward neural network, used to approximate a function $f(\cdot) : \mathbb{R}^m \rightarrow \mathbb{R}^n$, where m is the dimensionality of the input data, and n is the number of output dimensions. An example of an MLP with two hidden layers is illustrated in Figure 3.2 [60]. MLPs are a specific kind of neural network. When a neural network contains multiple hidden layers, it is typically referred to as a deep neural network.

For a given input vector $\mathbf{x}$, the operation performed by a single MLP layer is expressed as

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{3.4}$$

Here, $\mathbf{W}$ and $\mathbf{b}$ represent the weights and bias, respectively while $\mathbf{y}$ represents the output of the perceptron, and $\sigma(\cdot)$ is an activation function used to introduce non-linearity into the model. Some commonly used activation functions are given in the following:

- Hyperbolic Tangent:
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{3.5}$$
- Sigmoid:
$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \tag{3.6}$$
- Rectified Linear Unit (ReLU):
$$\text{ReLU}(x) = \max(x, 0). \tag{3.7}$$

Moreover, the softmax function is typically used in the final layer of a neural network, producing probability distributions over output classes. The softmax function is defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad i = 1, 2, ..., n \tag{3.8}$$

where n represents the total number of output dimensions.

3.2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are designed to process sequential data where temporal dependencies exist between data points. These networks are capable of maintaining information from previous steps and leveraging it for current outputs. The unfolded architecture of an RNN is illustrated in Figure 3.3 [60].

Traditional RNNs are commonly trained using Backpropagation Through Time (BPTT). However, they often suffer from the vanishing gradient problem, which hampers their ability to learn long-range dependencies effectively. To overcome this limitation, Long Short-Term Memory (LSTM)

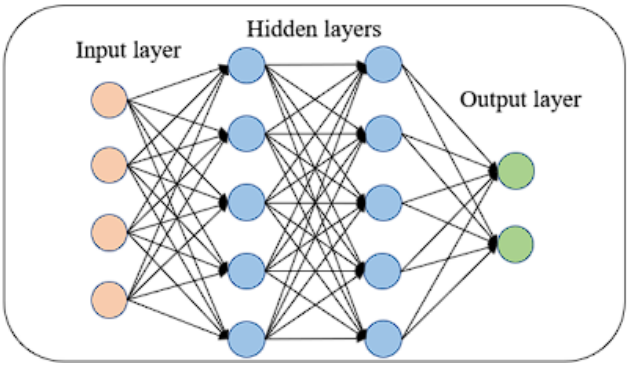

Figure 3.2: Structure of an MLP with 2 hidden layers

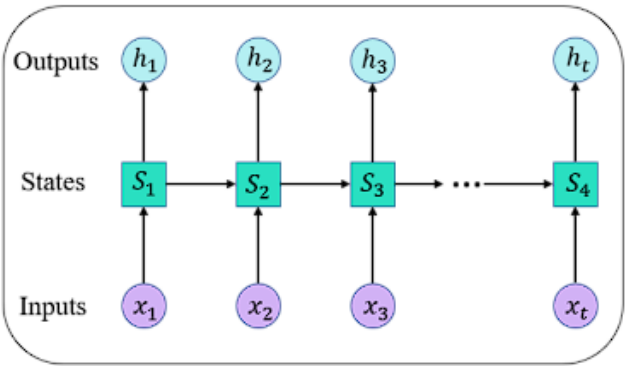

Figure 3.3: The architecture of a Recurrent Neural Network (RNN) includes x_t as the input sequence, s_t as the associated state vector, and h_t as the hidden layer output.

networks were introduced in [61]. LSTMs improve the capacity to model long-term dependencies by incorporating a memory cell and three gating mechanisms: the forget gate, the input gate, and the output gate. The memory cell stores information across time steps, justifying the use of the term "memory" in LSTM. The forget gate determines the extent to which information from the previous cell state should be discarded. The input gate regulates how much of the current input should be written to the cell state. The output gate controls how much information from the cell state is sent to the output. These gates are typically implemented using deep neural networks. The structure of an LSTM unit is depicted in Figure 3.4.

3.3 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAVs), commonly referred to as drones, have emerged as transformative assets in modern wireless communication, data processing, and mobile edge computing systems. Their mobility, ease of deployment, and flexibility make them highly suitable for a wide range of

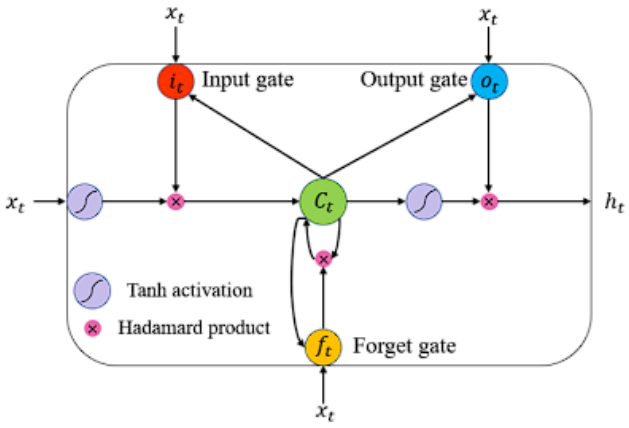

Figure 3.4: The inner structure of an LSTM layer. C_t denotes the cell outputs.

applications such as environmental monitoring, disaster management, and surveillance. In particular, UAVs offer significant potential in Mobile Edge Computing (MEC) environments by acting as flying edge servers, capable of supporting intelligent and latency-sensitive applications.

In traditional edge computing systems, computation tasks are often offloaded from user equipment to static edge servers or distant cloud servers. However, these infrastructures may not be feasible in remote or dynamically changing environments. UAVs overcome this limitation by providing on-demand, location-flexible edge computing capabilities. Their 3D mobility allows them to establish line-of-sight (LoS) communication links and dynamically reposition themselves to optimize resource utilization and minimize latency.

Despite their advantages, UAVs face critical challenges such as limited energy resources, constrained computational power, and restricted communication bandwidth. These limitations necessitate intelligent task offloading and resource allocation strategies, especially in scenarios involving Deep Neural Network (DNN) inference tasks. In such cases, a single UAV may not be able to process the entire DNN due to resource constraints. Therefore, partitioning DNN layers and distributing them across a cooperative network of UAVs can enable real-time inference while balancing computation loads and energy consumption.

This thesis focuses on optimizing the operation of UAV networks for efficient edge intelligence by investigating advanced methods for DNN layer offloading, resource allocation, and coordination among UAVs.

3.3.1 Types of UAVs

UAVs can be categorized based on their physical design, size, operational altitude, range, and autonomy level. These classifications help determine the suitability of UAVs for specific missions, including their role in MEC environments and cooperative DNN inference tasks.

3.3.1.1 Based on Airframe Design

- **Fixed-Wing UAVs:** These UAVs have rigid wings and require runways or catapults for takeoff and landing. They are capable of long-range, high-speed flights and are ideal for missions requiring wide-area coverage, such as mapping or surveillance. However, they lack the ability to hover and are unsuitable for tasks requiring stationary operations.
- **Rotary-Wing UAVs (Multirotors):** Including quadcopters, hexacopters, and octocopters, these UAVs can hover, take off, and land vertically. They provide high maneuverability, making them suitable for close-range missions like inspection and local edge computing. The primary drawback is their limited flight time and payload due to high energy consumption.
- **Hybrid UAVs (VTOL - Vertical Take-Off and Landing):** These UAVs combine the hovering capabilities of rotary-wing designs with the endurance of fixed-wing platforms. They are useful in applications requiring both vertical takeoff and long-range flight.

3.3.1.2 Based on Size and Payload Capacity

- **Nano UAVs:** Extremely small UAVs designed for indoor or short-range operations. They have minimal processing capabilities but can be used in swarm-based applications.
- **Micro and Mini UAVs:** Suitable for tactical missions and lightweight data collection. They can carry small sensors or cameras and may support low-power edge computing tasks.
- **Medium UAVs:** Offer moderate payload capacity and are commonly used in cooperative processing tasks involving DNN inference and data aggregation.
- **Large UAVs:** Equipped with high-capacity payload bays and long-endurance capabilities. These UAVs can host more powerful processors and are suitable for industrial and military-grade computation offloading applications.

3.3.1.3 Based on Operational Altitude and Range

- **Low-Altitude Short-Endurance (LASE):** Operate below 3,000 meters and are typically used for local, real-time applications such as traffic monitoring and crowd analysis.
- **Medium-Altitude Long-Endurance (MALE):** Operate between 3,000 and 9,000 meters with flight durations exceeding 24 hours. Suitable for continuous monitoring and relay tasks.
- **High-Altitude Long-Endurance (HALE):** Operate above 9,000 meters for strategic missions requiring days or weeks of continuous operation. They can serve as high-altitude communication relays or long-range reconnaissance platforms.

3.3.1.4 Based on Autonomy Level

- **Remotely Piloted UAVs:** Controlled by human operators via radio or satellite links. These are common in commercial applications with regulatory restrictions.
- **Autonomous UAVs:** Equipped with onboard sensors, processors, and AI algorithms, these UAVs can make decisions and navigate without real-time human intervention. They are best suited for dynamic and intelligent edge computing scenarios.

Understanding the characteristics and capabilities of different UAV types is crucial for designing efficient cooperative computation offloading architectures, especially when multiple UAVs are involved in executing distributed DNN tasks with minimal latency.

3.4 Matching Game Theory

Game theory is the study of mathematical frameworks that describe interactions among rational and intelligent decision-makers. These individuals, referred to as players or agents, participate in interactions that can involve both conflict and cooperation. Game theory provides general tools for analyzing scenarios where multiple players make decisions that influence each other's outcomes. A game models a situation in which individual players aim to achieve the most favorable results for themselves, while being fully aware that other players are doing the same [62].

One important subclass of game theory is the matching game, which focuses on pairing agents from two distinct sets based on preferences or utilities. A classic example is the stable marriage problem, where individuals are matched in such a way that no two players would prefer to be paired with each other over their assigned partners. Matching game theory is widely used in applications like resource allocation, job markets, and network association problems, where the goal is to find a stable and mutually beneficial assignment between participants.

3.4.1 One-to-One Matching: The Marriage Model

One well-known example in matching theory is the one-to-one matching problem, typically illustrated using the Marriage Model. In this setting, two disjoint sets of agents—commonly referred to as men and women—are matched based on their preferences over the agents on the other side of the market. Let the set of men be $M = \{m_1, \ldots, m_n\}$ and the set of women be $W = \{w_1, \ldots, w_p\}$. The market is two-sided: each man m_i has preferences over the set $W \cup \{m_i\}$, which includes all women and the option of remaining unmatched. A similar structure holds for women's preferences.

Each agent's preferences are assumed to be complete and transitive. For example, man m_i may have a preference ordering:

$$P(m_i) = w_k \succ_{m_i} w_l \succ_{m_i} \cdots \succ_{m_i} m_i \succ_{m_i} w_j \cdots$$

This indicates that m_i prefers woman w_k over w_l, and so on, and prefers to remain single rather than be matched with w_j. If $m_i \succ_{m_i} w_j$, then w_j is said to be unacceptable to m_i. When an agent has a strict ordering over all acceptable mates and clearly prefers being matched over remaining single, the preferences are said to be strict. Some of the theoretical results in matching theory assume strict preferences. In contrast, agents may also exhibit indifferences. For example, if man m_i is indifferent between w_l and w_m, his preference list could be written as:

$$P(m_i) = w_k \succ_{m_i} [w_l, w_m] \succ_{m_i} \cdots \succ_{m_i} m_i$$

indicating that m_i has no preference between w_l and w_m.

3.4.2 Stable Outcome

In the context of matching game theory, the outcome of a game is represented by a matching function

$$\mu : M \cup W \to M \cup W,$$

which assigns each individual either to a partner or to oneself if unmatched. This function must satisfy the following conditions:

- If man m is matched with woman w, then $\mu(m) = w$ and $\mu(w) = m$.
- The matching is two-sided, meaning $\mu(w) \in M \cup \{w\}$ and $\mu(m) \in W \cup \{m\}$; that is, each agent is either matched with someone from the opposite set or remains single.

A matching μ is said to be unstable if:

- It is blocked by an individual k who prefers being single over being matched with their assigned partner, i.e.,
$$k \succ_k \mu(k).$$
- It is blocked by a pair of agents (m, w) if both m and w prefer each other over their current partners under μ, i.e.,
$$w \succ_m \mu(m) \quad \text{and} \quad m \succ_w \mu(w).$$

A matching μ is called stable if it is not blocked by any individual or any pair of agents. Stable matchings are considered efficient and belong to the core of the market. In this simple model, the set of (pairwise) stable matchings is exactly equal to the core.

3.4.3 Deferred Acceptance Algorithm

The Deferred Acceptance Algorithm, introduced by Gale and Shapley in 1962, is a method for finding a stable matching between two equally sized sets of agents (commonly referred to as men and women) [3]. A matching is considered stable if there are no two individuals who would both prefer each other over their assigned partners as discussed earlier. This algorithm guarantees a

stable outcome and forms the foundation of many real-world matching systems, such as college admissions, task allocation and resource allocation. The main steps of the algorithm are described in the following:

1. **Preprocessing (step 0):** If any preference lists have ties, break them arbitrarily to ensure strict preferences.
2. **Initial Step (Step 1):**
   (a) Each man proposes to his most preferred acceptable woman.
   (b) Each woman:
      - Rejects all unacceptable proposals.
      - If she receives multiple acceptable proposals, she holds onto the one she prefers most and temporarily rejects the rest (deferred acceptance).
3. **Subsequent Steps (Step k, $k > 1$):**
   (a) Any man who was rejected in the previous round proposes to his next most preferred acceptable woman who has not yet rejected him. If no acceptable women remain, he does nothing.
   (b) Each woman:
      - Considers all current proposals along with the one she is holding.
      - Retains the most preferred among them and rejects the others.
4. **Termination:** The algorithm stops when no further proposals are made. Each woman is then matched with the man whose proposal she is holding (if any).

3.4.4 Many-to-one Matching: College Admissions Problem

The college admission problem, introduced by Gale and Shapley [3], is a fundamental model in matching theory. It provides a structured way to assign students to colleges based on mutual preferences while ensuring stable outcomes. This model underlies many real-world applications, including school choice systems and centralized college admissions.

A college admissions problem is defined as a 4-tuple:

$$(C, S, q, R)$$

where:

- $C = \{c_1, \ldots, c_m\}$ is a set of colleges,
- $S = \{s_1, \ldots, s_n\}$ is a set of students,
- $q = (q_1, \ldots, q_m)$ is a vector representing the capacity of each college,
- $R = (R_{c_1}, \ldots, R_{c_m}, R_{s_1}, \ldots, R_{s_n})$ is a collection of preferences.

Suppose:

- R_s: each student's preference relation over colleges and being unmatched.
- R_c: each college's preference relation over sets of students.
- P_c, P_s: strict preferences derived from R_c and R_s, respectively.

Colleges typically rank individual students, but since they admit multiple students, they must compare different subsets of applicants. Suppose:

- T is a set containing a college c's 2nd and 4th choices.
- T' contains the 3rd and 4th choices.

If the college prefers T over T', we write $T \succ_c T'$.

Now, if T'' includes the 1st and 5th choices, and the college prefers T'' to T, then $T'' \succ_c T$. Note that different preference relations over sets (P_c) can be consistent with the same ranking over individual students. However, this flexibility does not affect the fundamental definition of a stable matching. Following Roth (1985) [63], a college's preference relation R_c is said to be responsive if:

- The acceptability of a student to the college is independent of which other students are present.

- The comparison between any two students' desirability is unaffected by other students in the class.

Responsive preferences simplify college decision-making by focusing only on individual merit, regardless of the composition of the rest of the admitted group. A college's preference relation R_c is said to be responsive if the following conditions hold:

1. For any subset $T \subseteq S$ with $|T| < q_c$, and any student $s \in S \setminus T$,

$$T \cup \{s\} \succ_c T \quad \text{and} \quad \{s\} \succ_c \emptyset$$

. This implies that adding an acceptable student is always better than not having them.

2. For any $T \subseteq S$ with $|T| < q_c$, and for any two students $s, s' \in S \setminus T$,

$$T \cup \{s\} \succ_c T \cup \{s'\} \quad \Leftrightarrow \quad \{s\} \succ_c \{s'\}$$

. This ensures that the college's ranking between any two students is independent of other students in the set.

The outcome of a college admission problem can be described by a matching. Formally, matching is a correspondence $\mu : C \cup S \to C \cup S$ such that:

(1) For each college $c \in C$, the set of matched students $\mu(c) \subseteq S$ satisfies:

$$|\mu(c)| \leq q_c$$

(Colleges may have no students assigned, i.e., $\mu(c) = \emptyset$).

(2) For each student $s \in S$, we have:

$$\mu(s) \in C \cup \{s\}, \quad \text{with } \mu(s) = s \text{ indicating that } s \text{ is unmatched.}$$

(3) The matching is mutual:

$$s \in \mu(c) \iff \mu(s) = c \quad \text{for all } c \in C, s \in S.$$

A matching μ is considered unstable if there exists a student or college (or both) that can improve their outcome by deviating from the match. Otherwise, the matching is stable.

A matching μ is blocked if:

- **By a college** c: There exists $s \notin \mu(c)$ such that:

$$\{s\} \succ_c \mu(c)$$

- **By a student** s: The student prefers another college over their current match:

$$c \succ_s \mu(s)$$

- **By a pair** (c, s): The student and the college mutually prefer each other:

$$c \succ_s \mu(s), \quad \text{and}$$

$$\text{either } \left(s \notin \mu(c) \text{ and } |\mu(c)| < q_c\right), \quad \text{or } \left(\exists s' \in \mu(c) \text{ such that } \{s\} \succ_c \{s'\}\right)$$

A matching is stable if it is not blocked by any student, college, or pair. Application of matching game theory to resource allocation and task offloading in edge computing has received great attention by the research community in recent years.

Chapter 4

Collaborative Offloading and Resource Allocation for Efficient Inference of DNNs

4.1 Abstract

Novel integration of Deep Neural Networks (DNNs) into Unmanned Aerial Vehicles (UAVs) based systems has enabled numerous smart civilian and military applications. However, the inherent computational complexity of DNNs often results in significant inference delay, which could violate the delay requirements of practical UAV operations and applications. In this work, we show how the joint offloading of DNN layers and resource allocation to support DNN inference in UAV-based wireless systems can be formulated as an optimization problem, aiming to minimize the total inference latency considering constraints on the processing order of DNN layers, wireless connectivity, limited computational resources, and processing quotas of individual UAVs. We then propose an optimization based algorithm (called LARA) that employs the alternating optimization method by tackle the formulated problem by iteratively solving the three subproblems until convergence. For potential decentralized implementation, a novel matching game theory-based algorithm (called GAME-MIND) is designed to enable efficient allocation of DNN layers of different inference requests to the UAVs. By modeling the problem as a matching game, GAME-MIND ensures efficient

resource utilization while balancing the computation load across UAVs in the network. Numerical results show that GAME-MIND achieves desirable delay performance compared to the centralized LARA counterpart. In addition, GAME-MIND greatly outperforms two different heuristic-based methods that offload DNN layers to nearest neighboring UAVs where these two heuristic baselines can achieve at least twice the inference delay due to GAME-MIND in high-load conditions. We also demonstrate the efficacy of GAME-MIND in terms of convergence and study its achieved communications, computation latency, and total inference delay under different network settings.

4.2 Introduction

Recent advancements in deep neural networks (DNN) [64] have revolutionized many application domains such as image recognition, speech processing, and genomics data analysis [65]. DNNs, which were trained offline with large datasets, can be deployed on unmanned aerial vehicles (UAVs) to carry out different recognition and surveillance tasks and applications including precision agriculture, disaster response and management, environmental monitoring, smart city surveillance, and rescue missions thanks to the flexibility, mobility, pervasive data collection capability, low-cost deployment and maintenance of the UAVs [66, 67]. UAVs can be equipped with high-quality cameras that capture images and videos which are used for onboard recognition tasks. In fact, UAVs have been widely used to detect events in difficult, hardly accessed areas such as forests, war zones, flooding, and earth-quake effected areas.

When a group of UAVs, which can also be called a UAV swarm, is dispatched for carrying out a particular mission, they gather a lot of data using different onboard sensors such as temperature, humidity, images, and videos [68]. Sending the data collected by UAVs' sensors to remote servers for processing can increase communication cost and processing latency and this can even be impossible in scenarios where the ground communication infrastructure is not available or damaged (e.g., natural disasters). To overcome these challenges, computing hardware such as GPU can be installed on UAVs for onboard data processing. Although rapid advances in DNN can be leveraged to enable processing for data collected by UAVs [69] (i.e., running inference of the deployed pre-trained DNN models), onboard UAVs' computational resources are typically limited to handle the heavy DNN inference tasks and it can be challenging to meet delay requirements of most practical applications [70]. To this end, computation offloading and resource allocation design to enable

efficient collaborative inference of the deployed DNN models is an important but still underexplored research problem.

Collaborative inference design involves breaking down the DNN model into distinct segments, such as layers or computation tasks, and assigning each segment to a specific participant or UAV in this context. In the current work, we assume that the whole DNN is broken down into DNN layers and these layers must be distributed across the UAVs for processing where the underlying UAVs form a connected UAV wireless network where the design aims to minimize the total latency of all inference requests. In particular, different DNN layers can be processed by different UAVs determined by the underlying collaborative computation offloading algorithm to effectively cope with limited computational resources at individual UAVs. This design requires an appropriate modeling of the wireless connectivity constraints, sequential processing constraints of different DNN layers, which have not fully addressed by the existing literature.

Efficient integration of DNNs into UAV based systems to enable intelligent and efficient handling of complicated recognition tasks without any human intervention has been an active research topic. Computation offloading under different edge computing settings has been explored in recent years [71]. Computation offloading can generally be classified into two categories: full offloading and partial offloading. The key distinction between them is whether the entire task is transferred to the edge or cloud server. Several studies have focused on full offloading. For instance, Long et al. [72] developed an efficient strategy to offload computation requests for multiple users with the aim of minimizing both overall resource consumption and user-perceived latency. Additionally, other research efforts have investigated offloading in mobile edge computing environments involving multiple servers [1, 73–75]. These scenarios present greater complexity, as they require not only the decision of whether to offload, but also the determination of the optimal offloading destination. Partial offloading has also garnered significant attention. Liu et al. [76], for example, introduced a price-based distributed approach for managing offloaded tasks, allowing tasks to be partitioned at the bit level to enable flexible partial offloading.

At present, a wide range of algorithms have been proposed by researchers to address the computation offloading problem. Some have formulated this problem as a Mixed-Integer Linear Program (MILP). For example, Ning et al. [77] modeled a single-user offloading problem as an MILP and used a branch-and-bound algorithm to minimize latency. Other studies have also formulated the

computation offloading problems as MILP, aiming to minimize the energy consumption [78–80]. Online algorithms are also widely adopted in computation offloading design. Huang et al. [81], for instance, developed an online algorithm aimed at minimizing total task completion time. Their approach considers multiple wireless devices executing dynamic, computation-intensive tasks and leverages a meta learning to develop their framework. Lyapunov optimization theory also serves as a foundational tool for online algorithm design. Using this theory, Abbas et al. [82] introduced user-centric strategies for real-time task offloading and resource allocation, aiming to reduce both energy consumption and financial cost, while maximizing task completion. Another line of research focuses on dynamic environments. Merluzzi et al. [83] presented an energy-efficient algorithm for dynamic computation offloading in a multi-access edge computing environment where multiple mobile users share communication and computing resources. Their solution was developed by using stochastic optimization where they solve a convex problem in each time slot using a fast iterative method, effectively balancing service delay and energy consumption at mobile devices. Moreover, reinforcement learning (RL) techniques have gained popularity in recent years in computation offloading design [84–89]. In these studies, the computation offloading and resource allocation problem is modeled as a Markov Decision Process (MDP), and RL algorithms are used to engineer the decision-making process.

Game theory has been adopted in computation offloading research [90]. In particular, some existing studies have modeled the computation offloading problem as a cooperative game. For example, He et al. [91] introduced a game-theoretic collaborative offloading strategy designed to reduce latency across multiple devices while meeting delay constraints. Other researchers have addressed the problem using the Stackelberg game approach [92–94], where the service provider acts as a leader by first committing to a strategy while users make optimal response decisions based on the provider's service offerings and resource allocation plan. Furthermore, the negotiation game model has been explored in this context [95], focusing on negotiations between users and service providers to establish mutually agreeable terms for resource usage and pricing. Recently, there has also been an increasing interest in integrating game theory with reinforcement learning to develop advanced computation offloading strategies [96, 97].

The research most relevant to our work is the one done by Dhuheir et al. [98], who explored the application of multiple UAVs for surveillance and monitoring in dynamic, real-time image recognition scenarios. Their research aimed to reduce the latency involved in reaching a final classification

Table 4.1: Related work on computation offloading and resource allocation in edge computing

Ref.	**Objective**	**Inference Latency Minimization**	**Centralized Optimization based Approach**	**Distributed Optimization based Approach**	**Resource Allocation Constraint**	**Task Dependency**	**Connectivity Constraint**
[77]	Minimize the execution latency		✓		✓		
[78]	Minimize the weighted sum-energy consumption		✓		✓		
[80]	Minimize the energy consumption for completing the tasks		✓		✓		
[82]	Maximizing the number of completed tasks while Minimizing energy consumption		✓		✓		
[83]	Minimizing the long-term average energy consumption		✓		✓		
[91]	Minimize the weighted sum of execution offloading latencies			✓	✓		
[98]	Minimize total inference latency	✓		✓		✓	
[99]	Minimize total inference latency	✓	✓		✓	✓	✓
Our work	Minimize total inference latency	✓	✓	✓	✓	✓	✓

decision by optimizing the placement of DNN layers among participating UAVs. However, their approach was based on a fixed data rate and computation resource allocation strategy, which does not result in optimized inference latency. Moreover, they did not consider wireless connectivity of the participating UAVs which affect the reliability of the framework for the real-world environment.

Our offloading and resource allocation designs for DNN inference tasks aim to address the above mentioned limitations of existing works where we consider the sequential processing order of DNN layers, constraints on limited computational resources at UAVs, and wireless connectivity among the collaborating UAVs. The novel contributions of this work can be summarized as follows:

- We formulate the joint DNN layer assignment and resource allocation problem for the UAV based wireless network that aims to minimize the total inference latency considering constraints on wireless connectivity, processing order of DNN layers, processing quotas, and computational resources at individual UAVs.
- We propose to employ an alternating optimization method to solve the optimization problem where we iteratively optimize the DNN-layer assignment subproblem, computing resource allocation subproblem, and rate allocation subproblem iteratively until convergence. Accordingly, we develop a DNN layer assignment and resource allocation framework (LARA) for efficient and collaborative processing of DNN inference requests.
- We design a novel matching game theory-based algorithm, called GAME-MIND, by modeling the DNN layer allocation problem as a matching game, ensuring efficient distribution of computation tasks among the UAVs. Appropriate utility functions are defined to capture the preferences of DNN layers towards UAVs as well as preferences of UAVs to accept and process DNN layers, considering the inference latency minimization goal and computational resource sharing for DNN layers processed at each UAV.
- Extensive simulations demonstrate the desirable delay performance achieved by LARA and GAME-MIND algorithms. Specifically, the VGG16 model, which is a high-performance CNN model, is considered in performance evaluation of the collaborative inference designs. We compare the performance of GAME-MIND with two heuristic-based methods that offload DNN layers to the nearest neighboring UAVs. We show that GAME-MIND greatly outperforms the two heuristic-based methods, especially in the high-load settings.

Preliminary results of this work have been accepted for publication on [99]. However, this journal paper makes several major extensions compared to the conference version as can be summarized as follows. First, the matching game based algorithm is presented in this journal version but it was not included in the conference paper. Second, we provide more detailed literature survey and description

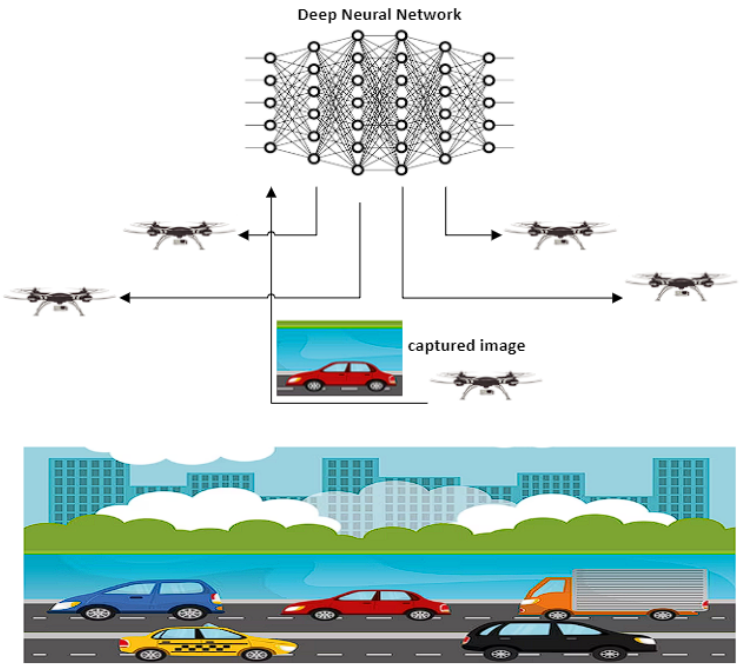

Figure 4.1: System Model

of novel contributions of this work compared to those in the conference version. Finally, much more extensive numerical results are presented in this work compared to the conference counterpart.

The remainder of this chapter is organized as follows. In Section II, we present the system model and problem formulation. We describe our optimization based algorithm in Section III. The matching game based algorithm is presented in Section IV. Numerical results are discussed in Section V, followed by conclusion in Section VI.

4.3 System Model

We consider a connected UAV wireless network as shown in Figure 4.1 in which each UAV can transmit data to any other UAV in the network via single or multi-hop communications.[1] We assume that positions of all UAVs are fixed (i.e., UAVs hover on a designated area and maintain connectivity with other UAVs) during the execution of the considered DNN inference requests. We further assume that the UAVs collaboratively perform inference of a pre-trained DNN based on the data collected by a subset of UAVs. The inference tasks for this set of UAVs correspond to the set

[1]The placement of UAVs to form the connected network is outside the scope of this work and it can be solved by using the technique in [2].

of computation requests. Specifically, the DNN is made up of several layers and the computation required by the inference task corresponds to the layer-by-layer processing of the DNN. For convenience, we denote the sets of UAVs, inference requests, and DNN layers as $\mathcal{U} = \{1, 2, 3, ..., N\}$, $\mathcal{R} = \{1, 2, 3, ..., R\}$, and $\mathcal{L} = \{1, 2, 3, ..., L\}$, respectively.

Each UAV initiating the corresponding inference request can perform the computation required by all DNN layers; however, offloading certain DNN layers to neighboring UAVs with abundant computational resources for processing can improve the inference latency. Such offloading design must consider the transmission of data related to offloaded DNN layers and optimization of the compute resource allocation. Details of the offloading design together with the resource allocation are described in the following.

4.3.1 Communication Model

To capture the connectivity among UAVs, we introduce binary parameters for each pair of UAVs, $\lambda_{i,k}, \forall i, k \in \mathcal{U}$, such that,

$$\lambda_{i,k} = \begin{cases} 1 & \text{if UAV } i \text{ and } k \text{ are connected,} \\ 0 & \text{otherwise.} \end{cases}$$

For brevity, we denote the connected link from UAV i to k as $\{(i, k), \text{ if } \lambda_{i,k} = 1, \forall i, k \in \mathcal{U}\}$. In our design, we assume that the UAVs stay connected for the duration of the inference session.

Let $g_{i,k,c}$ denote the channel gain of the UAV link (i, k) on the assigned channel and N_0 be the power density of the Gaussian noise at the receiver. Then, the data rate $\delta_{i,k}$ of link (i, k) can be calculated as

$$\delta_{i,k} = \omega \log_2 \left(1 + \frac{p_{i,k,c} g_{i,k,c}}{\omega . N_0}\right), \tag{4.1}$$

where ω denotes the bandwidth of each channel, $p_{i,k,c}$ represents the transmit power, and we have assumed orthogonal channel assignments for different network links.

4.3.2 DNN Inference Requests

Recall that there are R inference requests generated by different UAVs. Each inference request consists of L computation sub-tasks corresponding to different layers of the DNN, ranging from layer $l = 1$ to layer $l = L$. To offload a particular sub-task corresponding to layer l of request r, we must transmit the data associated with previous layer of the DNN (i.e., these data are features extracted by the involved layer). Note that the data associated with different sub-tasks (r, l) can be transmitted over the same link (i, k). Let $\pi_{r,l,i,k}$ denote the data rate allocated to sub-task (r, l) to transmit the associated data over link (i, k). Then, we have

$$\sum_{r=1}^{R}\sum_{l=1}^{L}\pi_{r,l,i,k} \leq \delta_{i,k}.\lambda_{i,k}. \tag{4.2}$$

To capture the DNN layer assignments for different UAVs, let $q_{r,l,i}$ be a binary variable capturing whether layer l of request r is assigned to UAV i:

$$q_{r,l,i} = \begin{cases} 1 & \text{if layer } l \text{ of request } r \text{ is assigned to UAV } i, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, suppose $\rho_{r,l,(l+1)}$ represent the amount of data which must be transmitted if layer $l + 1$ of request r is offloaded and processed by a UAV different from the UAV i processing layer l. Then, the communication delay $\tau_{r,l,i,k}$ involved in transferring of data associated with layer l of request r from UAV i to UAV k can be calculated as

$$\tau_{r,l,i,k} = \frac{q_{r,l,i}.q_{r,(l+1),k}.\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}. \tag{4.3}$$

4.3.3 Computing Model

Suppose the computation capacity of UAV $i \in \mathcal{U}$ is S_i. Then, the computation capacity must be shared to process different sub-tasks (r, l) assigned to each UAV. Let the computation resource

allocated to process the sub-task (r, l) at UAV i be $S_{r,l,i}$. Then, we have

$$\sum_{r=1}^{R}\sum_{l=1}^{L} S_{r,l,i} \leq S_i. \tag{4.4}$$

Let $\gamma_{r,l}$ denote the computation demand of sub-task (r, l). Then, the computation delay for this sub-task (r, l) at UAV i can be calculated as

$$\sigma_{r,l,i} = \frac{q_{r,l,i} \cdot \gamma_{r,l}}{S_{r,l,i}}. \tag{4.5}$$

4.3.4 Problem Formulation

Let us define DNN layer assignment variables, computation resource allocation variables and data rate allocation variables as $\mathbf{Q} = \{q_{r,l,i}, \forall r \in \mathcal{R}, i \in \mathcal{U}, l \in \mathcal{L}\}$, $\mathbf{S} = \{S_{r,l,i}, \forall r \in \mathcal{R}, l \in \mathcal{L}, i \in \mathcal{U}\}$ and $\mathbf{R} = \{\pi_{r,l,i,k}, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, i, k \in \mathcal{U}\}$, respectively. Then, the joint optimization problem to minimize the total inference latency, which is called problem **(P)**, can be formulated as

$$\begin{aligned}\min_{\{\mathbf{Q},\mathbf{R},\mathbf{S}\}} & \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N} \frac{q_{r,l,i} \cdot q_{r,(l+1),k} \cdot \gamma_{r,l}}{S_{r,l,i}} \\ + & \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} q_{r,l,i} \cdot q_{r,(l+1),k} \cdot \frac{\rho_{r,l,(l+1)} \cdot \lambda_{i,k}}{\pi_{r,l,i,k}}\end{aligned}$$

subject to

$$\sum_{i=1}^{N} q_{r,l,i} = 1, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \tag{4.6}$$

$$\sum_{r=1}^{R} \sum_{l=1}^{L} q_{rlk} \leq \zeta_{th,k}, \forall k \in \mathcal{U}, \tag{4.7}$$

$$\sum_{r=l}^{R} \sum_{l=l}^{L} S_{r,l,i} \leq S_i, \forall i \in \mathcal{U}, \tag{4.8}$$

$$q_{r,l,i} \cdot q_{r,(l+1),k} \leq \lambda_{i,k}, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i, k \in \mathcal{U}, \tag{4.9}$$

$$\sum_{r=1}^{R} \sum_{l=1}^{L} \pi_{r,l,i,k} \leq \delta_{i,k} \lambda_{i,k}, \forall i, k \in \mathcal{U}, \tag{4.10}$$

$$q_{r,l,i} \in \{0, 1\}, \ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i \in \mathcal{U}, \tag{4.11}$$

$$S_{r,l,i} \geq 0, \ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i \in \mathcal{U}, \tag{4.12}$$

$$\pi_{r,l,i,k} \geq 0, \ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i, k \in \mathcal{U}, \tag{4.13}$$

where constraints (4.6) capture the fact that each sub-task (r, l) is assigned and processed by exactly one UAV, constraints (4.7) ensure that the total number of DNN layers processed by each UAV k is upper-bounded by its pre-determined limit $\zeta_{th,k}$ (also called processing quota) for balanced computation processing and congestion avoidance, constraints (4.8) mandate that the total computing resource allocated to support different sub-tasks at each UAV i must be bounded by the available computing capacity, constraints (4.9) imply that a sub-task can only be offloaded from UAV i to UAV k if they are connected, (4.10) captures the wireless capacity constraint for each link (i, k). Finally, constraints (4.11), (4.12), (4.13) capture the binary or non-negative characteristics of different optimization variables. This is a mixed non-linear integer optimization problem, which is very hard to solve.

4.4 Optimization Based Algorithm

We propose to employ the alternating optimization method to solve the optimization problem presented in the previous section. Specifically, we design a novel framework called DNN layer assignment and resource allocation (LARA) which decomposes the problem into multiple subproblems, each with one set of optimization variables given the values of other optimization variables. These

subproblems are solved iteratively until convergence. We discuss how to solve these subproblems in the following.

4.4.1 DNN Layer Assignment Subproblem

For given values of **S** and **R**, the DNN layer assignment subproblem can be written as

$$\textbf{(P1)}: \min_{\mathbf{Q}} \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N} \frac{q_{r,l,i}.q_{r,(l+1),k}.\gamma_{r,l}}{S_{r,l,i}} + \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} q_{r,l,i}.q_{r,(l+1),k}.\frac{\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}$$

subject to constraints (4.6), (4.7), (4.9), and (4.11).

Problem **(P1)** is a non-linear integer program due to the quadratic terms in the objective function and constraints (4.9). To linearize this problem, we introduce auxiliary variables $y_{r,l,i,k}$ defined as follows:

$$y_{r,l,i,k} = q_{r,l,i}.q_{r,(l+1),k}, \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i,k \in \mathcal{U}. \tag{4.14}$$

Then, the following constraints must be imposed to make sure that $y_{r,l,i,k}$ equals $q_{r,l,i}.q_{r,(l+1),k}$:

$$y_{r,l,i,k} \leq q_{r,l,i}, \tag{4.15}$$

$$y_{r,l,i,k} \leq q_{r,(l+1),k}, \tag{4.16}$$

$$y_{r,l,i,k} \leq q_{r,l,i} + q_{r,(l+1),k} - 1. \tag{4.17}$$

For convenience, let us define $\mathbf{Y} = \{y_{r,l,i,k}.\forall r,l,i,k\}$. Then, problem (**P1**) can be transformed into the following problem:

$$\textbf{(P1.1)}: \min_{\{\mathbf{Q},\mathbf{Y}\}} \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N} \frac{y_{r,l,i,k}.\gamma_{r,l}}{S_{r,l,i}} + \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} y_{r,l,i,k}.\frac{\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}$$

subject to constraints (4.6), (4.7), (4.9), (4.11), (4.15), (4.16), (4.17), and

$$y_{r,l,i,k} \in \{0,1\},\ \forall r \in \mathcal{R}, \forall l \in \mathcal{L}, \forall i,k \in \mathcal{U}. \tag{4.18}$$

Problem **(P1.1)** is a linear integer program, which can be solved efficiently by using the Python tool CVXPY.

4.4.2 Computing Resource Allocation Subproblem

For given values of $\mathbf{Q}, \mathbf{R}$, the computing resource allocation subproblem can be written as

$$\begin{aligned}\textbf{(P2)}: \min_{\mathbf{S}} & \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N} \frac{q_{r,l,i}.q_{r,(l+1),k}.\gamma_{r,l}}{S_{r,l,i}} \\ & + \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} q_{r,l,i}.q_{r,(l+1),k}.\frac{\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}\end{aligned}$$

subject to constraints (4.8) and (4.12).

Problem (**P2**) is a convex problem because $1/S_{r,l,i}$ is convex with respect to $S_{r,l,i}$ and all constraints (4.8) and (4.12) are linear. Therefore, it can be solved by using the Python tool CVXPY.

4.4.3 The Data Rate Allocation Subproblem

For given values of $\mathbf{Q}, \mathbf{S}$, the rate allocation subproblem can be expressed as

$$\begin{aligned}\textbf{(P3)}: \min_{\mathbf{R}} & \sum_{r=1}^{R}\sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{N} \frac{q_{r,l,i}.q_{r,(l+1),k}.\gamma_{r,l}}{S_{r,l,i}} \\ & + \sum_{r=1}^{R}\sum_{l=1}^{L-1}\sum_{i=1}^{N}\sum_{k=1}^{N} q_{r,l,i}.q_{r,(l+1),k}.\frac{\rho_{r,l,(l+1)}.\lambda_{i,k}}{\pi_{r,l,i,k}}\end{aligned}$$

subject to constraints (4.10) and (4.13).

Problem (**P3**) has the convex objective function and linear constraints (4.10). Therefore, it can be solved by using the Python tool CVXPY. Details of the proposed algorithm is summarized in Algorithm 4.1. The STOPPING CONDITION for the proposed iterative algorithm can be achieved when the difference of the total inference latency between two consecutive iterations is smaller than 1% of the resulting inference latency.

Algorithm 4.1. DNN Layer Assignment and Resource Allocation Algorithm (LARA)

INPUT: N, $\mathcal{R}$, $\mathcal{L}$, $\{\lambda_{i,k}, \forall i, k \in \mathcal{U}\}$
OUTPUT: Optimized variables **Q**, **S**, **R**, and total inference time
1: **while** STOPPING CONDITION = FALSE **do**
2: Solve problem **(P1.1)** to obtain DNN layer assignment solution **Q**
3: Solve problem **(P2)** obtain computing allocation solution **S**
4: Solve problem **(P3)** to obtain data rate allocation solution **R**
5: **end while**
6: Retrieve optimized values of **Q**, **S**, **R**
7: Calculate total inference time

4.5 Matching Game Based Algorithm

In this section, we show how problem **(P)** can be solved by using the matching game theory. Matching game theory is a technique that addresses the challenge of matching agents from two groups (e.g., workers and jobs, resources and devices in wireless communication systems) in a way that meets predetermined requirements, often leading to a stable and effective result [3]. The Gale-Shapley method, sometimes referred to as the Deferred Acceptance algorithm, is the most well-known application of matching game theory. In particular, we propose a Gale-Shapley method based decentralized algorithm to minimize DNN inference delay, which is referred to as GAME-MIND in the following. The GAME-MIND framework would be more suitable for large-scale, dynamic systems and more resilient against the single-point-of-failure issue compared to the centralized optimization approach.

4.5.1 DNN Layer Assignment Formulation as a Matching Game

In a matching game, each agent from one set is matched with a different agent from the other set. For a given matching, if no agent would rather be matched with another agent than its current match, the matching is deemed stable [3]. Specifically,every agent in one set in a matching game has preference of its potential partners where each agent indeed ranks agents in the other set. These preference relations are established based on utility functions as will be specified later. In the context of collaborative DNN inference, the matching game theory can be used to solve the problem of assigning DNN layers to UAVs. The objective is to assign DNN layers in such a way that minimizes the total inference delay while taking into account dependency among different DNN layers, limited computation capacity and connectivity of the UAVs.

Algorithm 4.2. Matching Game for DNN Layer Assignment to Minimize Inference Delay (GAME-MIND)

Input: N, $\mathcal{R}$, $\mathcal{L}$, $\mathcal{U}$.
Discovery
- Each UAV $i \in \mathcal{U}$ discovers the set of its one-hop neighbors (called potential helpers) $\mathcal{U}_i$ based on the connectivity information including itself in this set
Initialization:
- All layers (r, l), $r \in \mathcal{R}, l \in \mathcal{L}$ are free (unassigned)
- Each UAV $k \in \mathcal{U}$ has an empty list of accepted DNN layers: $\mathcal{A}_k = \emptyset, k \in \mathcal{U}$.
- Let $i_{r,l}$ be the UAV processing DNN layer $(r, l-1)$ if $l > 1$ or $i_{r,l}$ be the UAV initiating the inference request (r, l) if $l = 1$
for $l = 1 : L$ **do**
  while there exists DNN layer $(r, l), \forall r \in \mathcal{R}$ not accepted and there exists potential one-hop helpers to be requested **do**
    for $r \in \mathcal{R}$ **do**
      For each DNN layer (r, l):
      - UAV $i_{r,l}$ constructs preference relations of its potential one-hop helpers $\mathcal{U}_{i_{r,l}}$ based on (4.19)
      - Find $k^* = \arg\max_{k \in \mathcal{U}_{i_{r,l}}} \nabla^{(r,l)}_{(i_{r,l})}(k)$
      - UAV $i_{r,l}$ sends a computation or matching request to UAV k^* for executing layer (r, l) by setting $R_{(r,l)\to k^*} = 1$
    end for
    For each UAV $k \in \mathcal{U}$:
    - Update the set of requesting DNN layers: $\mathcal{N}^{req}_k := \{(r, l) : R_{(r,l)\to k} = 1\}$
    if $|\mathcal{N}^{req}_k| + |\mathcal{A}_k| \leq \zeta_{th,k}$ **then**
      - UAV k accepts all requests in $\mathcal{N}^{req}_k$
      - Update $\mathcal{A}_k := \mathcal{A}_k \cup \mathcal{N}^{req}_k$
    else
      - Update the set of requesting DNN layers: $\mathcal{N}^{req}_k := \mathcal{A}_k \cup \mathcal{N}^{req}_k$
      - Choose $\zeta_{th,k}$ DNN layers with highest utility $\nabla'_{(i_{r,l},k)}(r, l)$ among those in $\mathcal{N}^{req}_k$ and assign them to $\mathcal{A}_k$
      - Reject the remaining DNN layers and update: $\mathcal{N}^{rej}_k := \mathcal{N}^{req}_k \backslash \mathcal{A}_k$
      - For each DNN layer $(r, l) \in \mathcal{N}^{rej}_k$, UAV k sends a rejection notification to the requesting UAV $i_{r,l}$
      - Upon receiving the rejection notification from UAV k, each requesting UAV $i_{r,l}$ removing UAV k from its set of potential helpers: $\mathcal{U}_{i_{r,l}} := \mathcal{U}_{i_{r,l}} \backslash \{k\}$
    end if
  end while
end for
Return: Sets of DNN layers processed by different UAVs $\mathcal{A}_k$, $\forall k \in \mathcal{U}$.

A matching μ is a function μ: $\mathcal{R} \times \mathcal{L} \to \mathcal{U}$ that assigns a particular UAV $k \in \mathcal{U}$ to process each DNN layer $l \in \mathcal{L}$ of inference request $r \in \mathcal{R}$ in such a way that each DNN layer is assigned to exactly one UAV, and each UAV could handle several DNN layers based on its capacity and processing quota. Because DNN layers must be processed in the sequential manner (i.e., DNN layer

$l+1$ of any inference request can only be processed after layer l has been executed), we can break the DNN layer assignments into L matching phases, each requiring to assign one corresponding DNN layer of all inference requests in $\mathcal{R}$ to appropriate UAVs for processing.

For convenience, we denote the DNN layer l of inference request r as (r, l) in the following.

In matching game based design, utility functions are used to capture preferences of agents from one set to their partners on the other set. To make the assignment decision of DNN layer (r, l) to certain UAV k, we need to specify the UAV processing the previous DNN layer $(r, l-1)$, except if we consider the first layer. Specifically, suppose UAV i processes DNN layer $(r, l-1)$ if $l > 1$ or UAV i is the one initiating the inference request r if $l = 1$. Then, we denote the utility function $\nabla_{(i)}^{(r,l)}(k)$ of DNN layer (r, l) for UAV k while denoting the utility function of UAV k for DNN layer (r, l) as $\nabla'_{(i,k)}(r, l)$. Given these utilities, we can say that the considered DNN layer (r, l) prefers UAV k_1 over UAV k_2 if $\nabla_{(i)}^{(r,l)}(k_1) > \nabla_{(i)}^{(r,l)}(k_2)$ and this preference is denoted by $k_1 \succ_{(r,l)} k_2$. Similarly, UAV k prefers DNN layer (r_1, l) over DNN layer (r_2, l) if $\nabla'_{(i,k)}(r_1, l) > \nabla'_{(i,k)}(r_2, l)$ and this preference is expressed as $(r_1, l) \succ_k (r_2, l)$.

Now we define the utility function $\nabla_{(i)}^{(r,l)}(k)$ as

$$\nabla_{i}^{(r,l)}(k) = -\left(\frac{\rho_{r,(l-1),l}}{\alpha_{(i,k)}^{(r,l)}.\delta_{(i,k)}} + \frac{\gamma_{r,l}}{\beta_{k}^{(r,l)}.S_k}\right), \tag{4.19}$$

where the first term inside brackets estimates the transmission delay of output data associated with the DNN layer $(r, l-1)$ over the UAV link (i, k) while the second term inside brackets estimates the computation delay; the rate and computing sharing factors, denoted as $\alpha_{(i,k)}^{(r,l)}$ and $\beta_{k}^{(r,l)}$, respectively can be calculated based on the fractions of data or computation load sharing the corresponding UAV link rate or UAV's computation capacity as follows:

$$\alpha_{(i,k)}^{(r,l)} = \frac{\rho_{r,(l-1),l}.\lambda_{i,k}}{\rho_{r,(l-1),l} + \sum_{r'=1}^{R}\sum_{l'=2}^{l} q_{r',(l'-1),i}.q_{r',l',k}.\rho_{r',(l'-1),l'}}, \tag{4.20}$$

$$\beta_{k}^{(r,l)} = \frac{\gamma_{r,l}}{\gamma_{r,l} + \sum_{r'=1}^{R}\sum_{l'=1}^{l} .q_{r',l',k}.\gamma_{r',l'}}, \tag{4.21}$$

where (r', l') represents the DNN layer which is potentially processed by UAV k. By defining the utility function based on the total data transmission and computation delay, our matching based

design encourages DNN layer assignments with smaller inference delay, which is indeed our design goal.

Similarly, the utility function of UAV k for DNN layer (r, l) can be defined as

$$\nabla'_{(i,k)}(r, l) = -\frac{\gamma_{r,l}}{\beta_k^{(r,l)}.S_k}. \tag{4.22}$$

The utility function $\nabla'_{(i,k)}(r, l)$ provides a quantitative measure for matching UAVs with DNN layers of inference requests. Again, the computing resource at each UAV is assumed to be allocated proportionally to the computational demand of assigned DNN layers, ensuring a fair distribution of resources. Given a UAV k with total computation capacity S_k, the computation resource assigned to each layer (r, l) is assumed to be based on its computation workload $\gamma_{r,l}$ relative to the total workload of all DNN layers assigned to UAV k. Moreover, a UAV will accept additional DNN layers only if its total number of assigned DNN layers does not exceed its predefined processing quota $\zeta_{th,k}$, preventing excessive processing delay. We now provide a more formal definition of the matching stability.

Definition 1. An assignment solution or matching of DNN layers to UAVs will be called unstable if there are two DNN layers (r, l_1) and (r, l_2) which are assigned to UAVs k_1 and k_2, respectively, although DNN layer (r, l_2) prefers UAV k_1 to UAV k_2 and UAV k_1 prefers DNN layer (r, l_2) to DNN layer (r, l_1).

The matching mentioned in the above definition is unstable because one can enhance the assignment solution by matching the DNN layer (r, l_2) to UAV k_1 and let the DNN layer (r, l_1) to be matched with other UAV with available quota. A matching is then called stable if there are no assignment/matching pairs as described in the above definition in the underlying matching. Interestingly, there exists an efficient matching strategy, called "deferred acceptance" procedure [3], which can be employed to design a matching algorithm, leading to a stable and efficient matching solution. This is indeed pursued in our design presented in the following.

Algorithm 4.2 describes our proposed utility-driven process for assigning DNN layers to UAVs based on the matching game theory. Initially, all DNN layers are marked as unassigned, and each UAV k starts with an empty set of accepted layers $\mathcal{A}_k$. The algorithm operates iteratively and it is executed over L matching phases, corresponding to different DNN layers. In each matching

phase and for each free DNN layer (r, l), its representing UAV $i_{r,l}$ identifies the UAV k among the potential one-hop helpers that maximizes the utility defined in (4.19) and sends a matching request to that UAV. For each UAV k, upon receiving all matching requests in each matching phase, it verifies if accepting all these new requests in $\mathcal{N}_k^{req}$ can still maintain its processing constraint with up to accepted $\zeta_{th,k}$ DNN layers. If the constraint is still satisfied, UAV k accepts and adds all DNN layers in $\mathcal{N}_k^{req}$ to $\mathcal{A}_k$. Otherwise, UAV k just accepts exactly $(\zeta_{th,k})$ DNN layers it prefers the most according to the utility expressed in (4.22), updates $\mathcal{A}_k$, and rejects the remaining DNN layers. For each rejected DNN layer $(r, l) \in \mathcal{N}_k^{rej}$, the UAV k is removed from the set of potential one-hop helpers, which will be considered in the future matching attempts. This iterative procedure is continued for all DNN layers until each DNN layer is either accepted by some UAV or it is rejected by all potential one-hop helpers.

We state a desirable property of the DNN layer assignment solution achieved by GAME-MIND in the following proposition.

Proposition 1. *For the DNN layer assignment solution achieved by GAME-MIND, each DNN layer (r, l) is at least as well off in terms of achievable utility as under other stable assignment solution.*

Proof. The GAME-MIND algorithm was designed based on the "deferred acceptance" procedure described in [3]. As shown in [3], for the stable matching obtained by this procedure, each DNN layer (r, l) is at least as well off in terms of achievable utility as under other stable assignment solution. Therefore, we have proved the proposition. □

Remark 1. It can be shown that the GAME-MIND algorithm indeed converges. In fact, each DNN layer (r, l) only needs to explore a finite number of UAVs (i.e., one-hop neighboring UAVs of the UAV $i_{r,l}$ processing the previous DNN layer); therefore, only a finite number of matching requests must be sent for each DNN layer and they can end up being accepted or rejected. Because there are exactly RL DNN layers to be matched with appropriate UAVs, the GAME-MIND algorithm converges after a finite number of matching attempts.

Remark 2. The GAME-MIND algorithm can be implemented in a distributed manner through appropriate message exchanges among UAVs. Specifically, the transmissions of matching requests

and acceptance/rejection messages of underlying UAVs to their one-hop neighbors in each matching phase can be readily realized by using any standard wireless broadcast mechanism.

4.5.2 Complexity Analysis

We provide complexity analysis of the proposed algorithms in the following. For the centralized optimization framework LARA, its complexity is due to the complexity involved in solving the three subproblems in each iteration. since CVX [58] invokes the interior-point method to solve the underlying optimization problem, the involved complexity is $\mathcal{O}(m_1^{1/2}(m_1+m_2)m_2^2)$, where m_1 is the number of inequality constraints, m_2 denotes the number of variables [55], and $\mathcal{O}$ denotes the big-O notation. Hence, the complexity of the subproblem **(P1.1)** is $\mathcal{O}(R^{3.5}L^{3.5}N^7)$ the complexity of the subproblem **(P2)** is $\mathcal{O}(R^{3.5}L^{3.5}N^{3.5})$ and the complexity of the subproblem **(P3)** is $\mathcal{O}(R^{3.5}L^{3.5}N^7)$. Since the lower order terms are dominated by higher order terms, therefore, the complexity of LARA algorithm is $\mathcal{O}(IR^{3.5}L^{3.5}N^7)$ where I denotes the number of iterations required to achieve convergence.

For the proposed matching game based algorithm, recall that the number of DNN layers and UAVs are RL and N, respectively. Also, the set of potential one-hop helpers for any UAV has its size bounded by N. For each DNN layer (r, l), we need to calculate the utilities for all potential one-hop helpers, and search for the one with the highest utility to send the computation request. This has the complexity of $\mathcal{O}(N)$. There are exactly R DNN layers to be matched with appropriate UAVs in each matching phase, leading to the total complexity of $\mathcal{O}(RN)$. Each UAV k upon receiving all computation requests may need to rank all DNN layers in the requesting set $\mathcal{N}_k^{req}$ whose size is bounded by $R + \zeta_{th,k}$ to choose the $\zeta_{th,k}$ most preferred DNN layers. This has the complexity of $\mathcal{O}((R + \zeta_{th,k}) \log \zeta_{th,k})$. For each DNN layer in each matching phase, one may need to send computation requests to all potential one-hop helpers whose size is bounded by N if its computation requests keep rejected. Therefore, the overall complexity of the proposed matching based algorithm considering all L DNN layers is $\mathcal{O}[LN((R + \zeta_{max}) \log \zeta_{max} + RN)]$ where $\zeta_{max} = \max_{k \in \mathcal{U}} \zeta_{th,k}$.

Table 4.2: Simulation Setup

Parameters	Value
Network area	500mx500m
UAV altitude	100 m
Channel bandwidth ω	15 KHz
UAV transmit power	20 dBm
Noise power density N_0	-174 dBm/Hz
Number of UAVs N	10, 15, 20
Number of DNN layers L (VGG16)	16
Number of inference requests R	[1:1:20]
Input Image	595x326 RGB
Computing Capacity S_i	[10:5:20] GFLOPs
Number of rounds	10

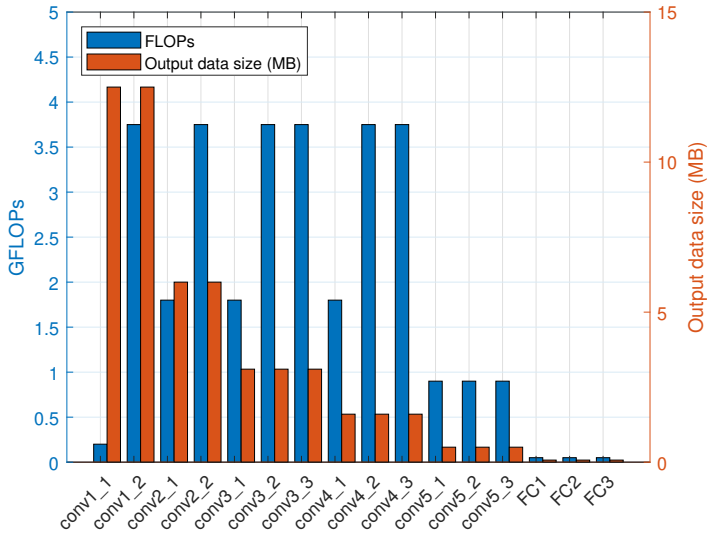

Figure 4.2: VGG16 layer-wise FLOPS and output data size

4.6 Numerical Results

In this section, we evaluate the performance of the proposed LARA and GAME-MIND algorithms. The simulation parameter settings are summarized in Table 4.2. We randomly place the UAVs over the network area of 500m x 500m and any two UAVs with inter-distance smaller than 50m are assumed to be connected. The power channel gains are simulated considering the path loss where the path loss for link (i, k) is assumed to be $PL_{i,k} = PL_o d_{i,k}^{-2}$ with $d_{i,k}$ being the distance between UAVs i and k and PL_0 being the path loss at reference distance of 1m. The DNN under consideration is the VGG16 for which the computing demand in FLOPS and data associated with different layers are presented in Figure 4.2. Simulations were conducted many times with different

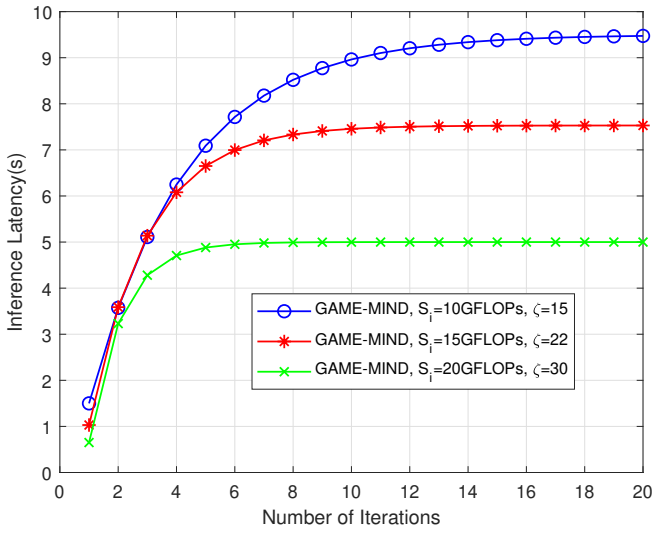

Figure 4.3: Inference latency of GAME-MIND for different values of UAVs' computation capacities vs matching iterations for $R = 10$ and $N = 10$

random factors including random placement of UAVs and inference requests. The results presented in this section are the averaged results over several independent simulations (UAV placements and inference requests). We set the processing quota of individual UAVs (i.e., the maximum number of DNN layers to be accepted) to be the same for different UAVs and we denote this common parameter as ζ in the following. We will also set the maximum computation capacity of individual UAVs (S_i) to be the same to obtain the results presented in each figure.

Figure 4.3 demonstrates how the inference latency varies over matching iterations for different values of UAVs' computation capacity (S_i) and processing quota (ζ). The scenario in which UAVs have smallest computation capacity ($S_i = 10$ GFLOPs, $\zeta = 15$) exhibits the slowest convergence, requiring around 12 matching iterations before inference latency stabilizes. In fact, more limited UAVs' quotas would require UAVs to explore and send more matching requests to multiple neighboring UAVs for each DNN layer, causing slower convergence for the matching algorithm. As computation capacity and quota increase to ($S_i = 15$ GFLOPs, $\zeta = 22$), the GAME-MIND algorithm converges faster, stabilizing in approximately 10 matching iterations. This improvement is due to faster workload distribution among UAVs with increasing UAVs' quotas, reducing the need for excessive exchanges of matching requests. The fastest convergence is observed when UAVs have the highest computation capacity and quota ($S_i = 20$ GFLOPs, $\zeta = 30$), where the inference latency stabilizes within 8 iterations. Here, UAVs can process more DNN layers locally, reducing

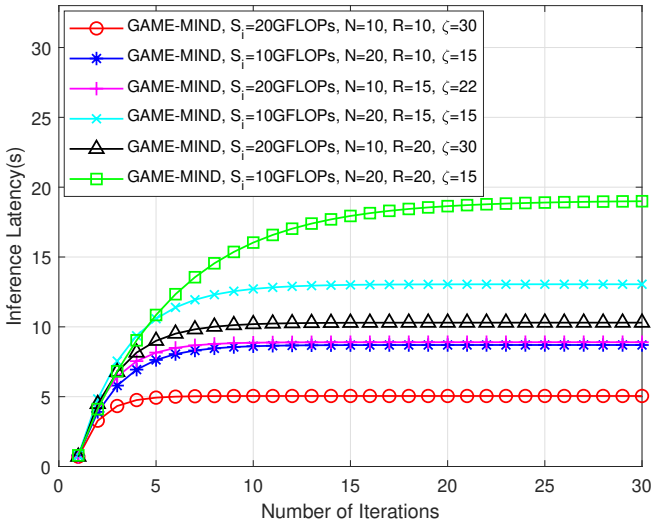

Figure 4.4: Inference latency of GAME-MIND for different network sizes, UAVs' computation capacities, and UAVs' quotas vs matching iterations

dependency on external resources and communication latency. Also, it enables the DNN layers to be matched quickly due the availability of high UAVs' computation resources and processing quotas. These results highlight that higher computation capacity and quota accelerate convergence of the GAME-MIND algorithm and help reduce both computation and communication delay.

Figure 4.4 illustrates the impacts of computation capacity (S_i), network size (N), number of inference requests (R), and UAVs' processing quota (ζ) on the convergence of the GAME-MIND algorithm. The scenario with ($S_i = 20$ GFLOPs, $N = 10$, $R = 10$, $\zeta = 30$) exhibits the lowest inference latency at convergence, around 5s within 10 matching iterations, as UAVs with higher computation capacity and larger quotas can process more DNN layers locally and for nearby UAVs, minimizing offloading overhead and accelerating convergence. In contrast, the setting with ($S_i = 10$ GFLOPs, $N = 20$, $R = 10$, $\zeta = 15$) achieves slightly higher latency (about 7s) at convergence since, despite the larger network size, UAVs with lower computation capacity and smaller quotas offload tasks more frequently, increasing communication delay and number of matching requests.

As the number of inference requests becomes larger ($R = 15, 20$), inference latency rises due to the increased processing demand. For the considered settings with ($S_i = 20$ GFLOPs, $N = 10$, $R = 15$, $\zeta = 22$) and ($S_i = 20$ GFLOPs, $N = 10$, $R = 20$, $\zeta = 30$), we can see that even with higher computation capacity, increasing R leads to a gradual rise in inference latency and convergence time. For two scenarios with ($S_i = 10$ GFLOPs, $N = 20$, $R = 15$, $\zeta = 15$) and ($S_i = 10$ GFLOPs,

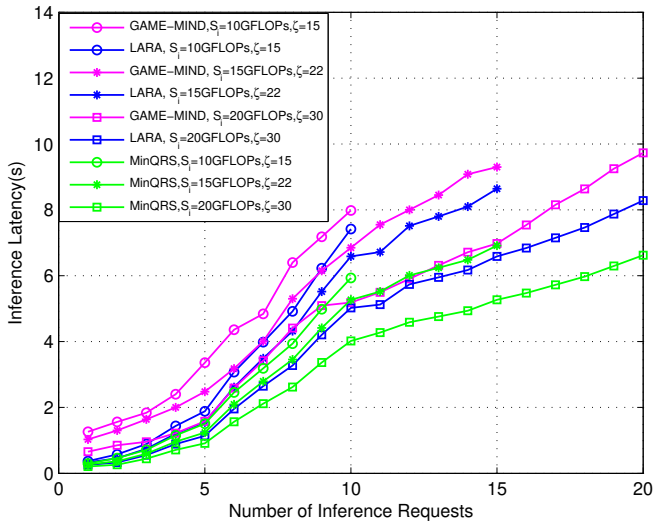

Figure 4.5: Inference latency of GAME-MIND, LARA and MinQRS vs the number of DNN inference requests for $N = 10$

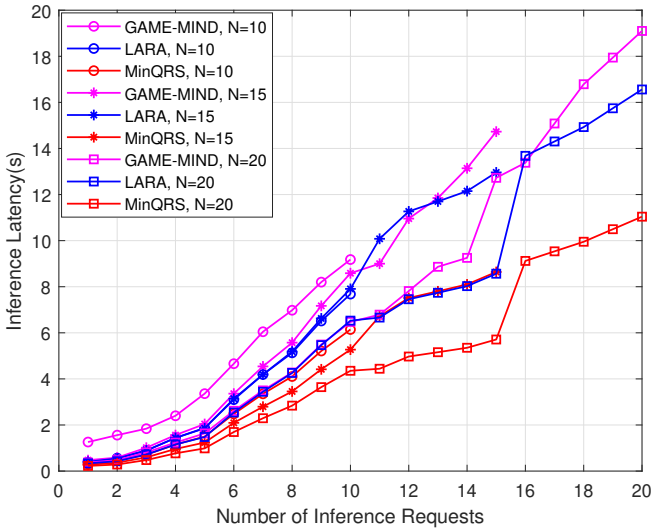

Figure 4.6: Inference latency of GAME-MIN, LARA and MinQRS vs number of DNN inference requests for different number of UAVs, $\zeta = 15$**, and** $S_i = 10$

$N = 20$, $R = 20$, $\zeta = 15$) where UAVs have lower computation capacity ($S_i = 10$ GFLOPs), the system may struggle to handle a larger number of inference requests, leading to high inference latency values. The lower quota ($\zeta = 15$) in these cases forces UAVs to offload tasks more frequently, increasing congestion and communication delay while the limited computation capacity ($S_i = 10$ GFLOPs) leads to high computation delay. Moreover, higher quotas ($\zeta = 30$) reduce the need of computation offloading and, therefore, improve the convergence time. Furthermore, the larger

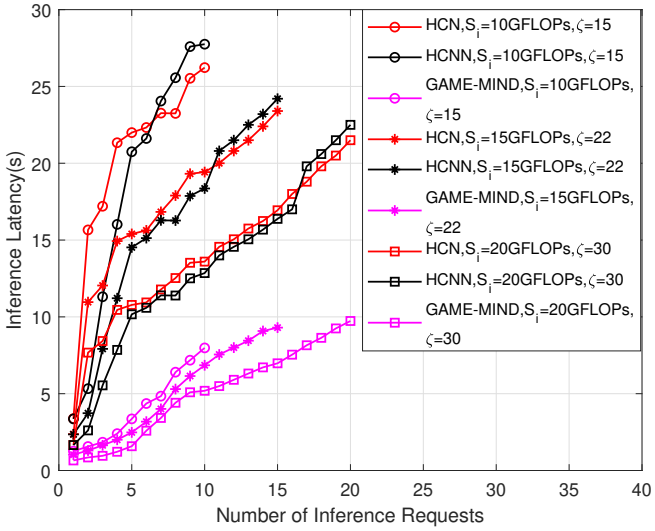

Figure 4.7: Inference latency of GAME-MIND and heuristics HCN and HCNN vs number of inference requests for $N = 10$

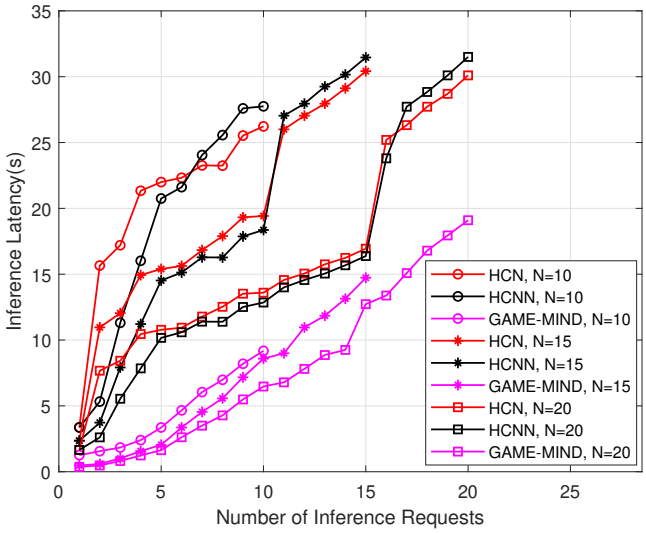

Figure 4.8: Inference latency of GAME-MIND and heuristics HCN and HCNN vs number of DNN inference requests for different number of UAVs N, $\zeta = 15$, and $S_i = 10$

UAV network size ($N = 20$) enhances task offloading opportunities but do not necessarily result in lower inference latency when UAVs have limited resources, and have to handle a larger number of inference requests ($R = 15, 20$), especially if UAVs have limited computation capacity. Thus, a balance between computation capacity, network size, and processing quota is essential for fast convergence and low inference latency of GAME-MIND.

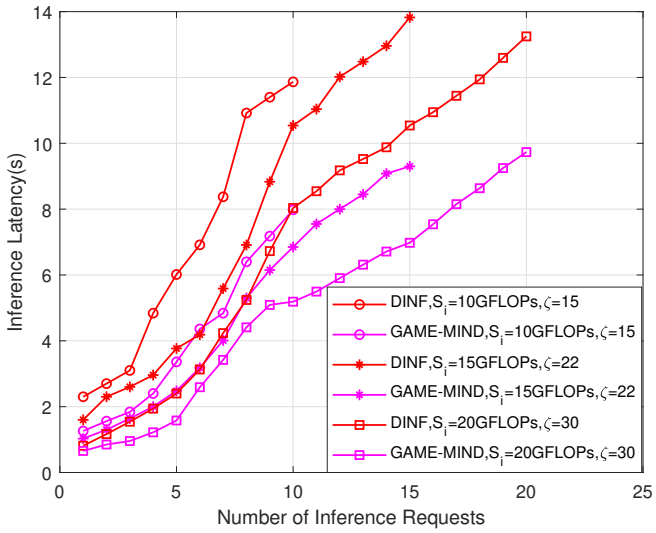

Figure 4.9: Inference latency of GAME-MIND and DINF vs the number of DNN inference requests for $N = 10$

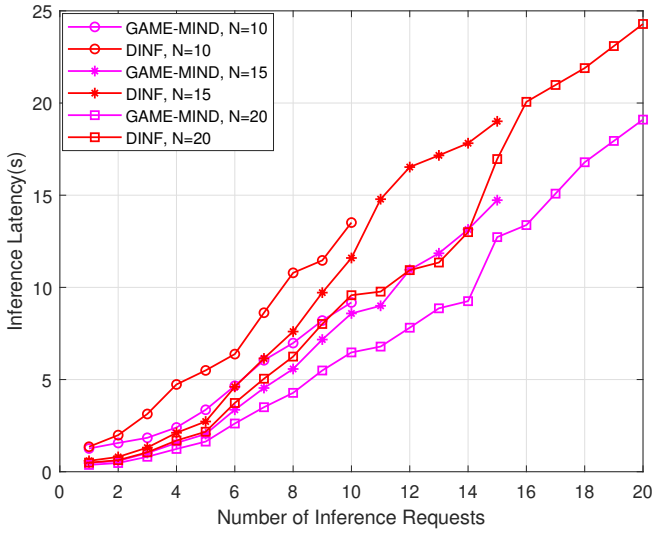

Figure 4.10: Inference latency of GAME-MIND and DINF vs number of DNN inference requests for different number of UAVs, $\zeta = 15$**, and** $S_i = 10$

In Figure 4.5, we compare the inference latency of GAME-MIND, LARA, and the baseline called MinQRS for varying number of inference requests under different parameter settings. Specifically, the MinQRS baseline solves the whole problem **P** using a solver without decomposing it into multiple subproblems as in LARA. Despite the high computation complexity, MinQRS achieves a near optimal solution, which can be used to benchmark other lower-complexity algorithms. As can be seen, for the setting with $S_i = 20$ GFLOPs the inference latency due to GAME-MIND remains very

close to that of LARA and MinQRS with $S_i = 15$ GFLOPs and $S_i = 10$ GFLOPs as the number of inference requests increases from 1 to 15. This trend suggests that if the number of inference requests is relatively low, the computation capacities of the UAVs are sufficient to handle the workload efficiently for all the algorithms LARA, MinQRS and GAME-MIND. Since the computation demand is not yet high in these cases, for all the three algorithms, most DNN layers are processed by the source UAVs of inference requests or assigned to UAVs close to the source UAVs; therefore, the communication delay does not make any significant contribution to the total inference latency for all the three algorithms. However, as the number of inference requests is larger than 15, a divergence in delay performance between the three algorithms begins to emerge. In particular, GAME-MIND, achieves slightly higher inference latency compared to LARA and MinQRS in this regime. This is attributed to the decentralized matching decisions made by GAME-MIND for DNN layer assignments, which can be less efficient than the centralized optimization approach under the high computation load settings. When UAVs have higher computation capacities ($S_i = 20$ GFLOPs), all the three algorithms achieve improved inference latency, but LARA and MinQRS still outperform GAME-MIND as the number of inference requests is sufficiently large. While LARA and MinQRS achieves lower inference latency than GAME-MIND, its centralized nature introduces scalability and robustness challenges in dynamic and large-scale UAV networks. GAME-MIND, despite its higher latency, offers a more scalable and flexible approach, making it suitable for decentralized and autonomous UAV operations.

Figure 4.6 compares the inference latency of GAME-MIND, LARA, and MinQRS vs the number of inference requests as different numbers of UAVs are deployed in the network. The results show that LARA and MinQRS achieve consistently lower inference latency than GAME-MIND across all values of N, highlighting the efficiency of centralized optimization in handling DNN inference requests. This is because LARA and MinQRS, as a centralized optimization frameworks, has a better global view of the network, enabling more effective computation offloading and resource management, whereas GAME-MIND operates in a distributed manner, making DNN layer assignments based on local information exchanges among neighboring UAVs and sub-optimal communication rate and computation resource allocation. For smaller network sizes ($N = 10$), the performance gap between GAME-MIND, LARA, and MinQRS is the more significant. As the number of inference requests increases, the inference latency in GAME-MIND grows at a steeper rate compared to LARA and MinQRS. In fact, LARA and MinQRS achieve lower inference latency compared to GAME-MIND

due to its centralized control and decision-making, which enables more efficient offloading and resource allocation across all UAVs.

For networks with larger numbers of UAVs ($N = 15$ and $N = 20$), the inference latency for all the three decreases, demonstrating the benefits of a larger network in handling inference requests. However, even with increased number of UAVs, LARA and MinQRS continue to outperform GAME-MIND as expected. Notably, for lower inference load (e.g., approximately 1 to 10 inference requests), the performance difference between GAME-MIND, LARA, and MinQRS, is relatively small, particularly as $N = 20$. However, as the number of inference requests increases beyond 10, GAME-MIND experiences a sharper rise in inference latency, while LARA and MinQRS maintains a more moderate increase, further demonstrating the performance advantage of the centralized optimization approach. Even though LARA and MinQRS are more efficient in minimizing inference latency, particularly in high-load scenarios, GAME-MIND remains a viable alternative for scenarios where a decentralized approach is required, such as when UAVs operate autonomously with limited or no communication to a central controller (e.g., a drone swarm).

Figure 4.7 presents a comparative study of GAME-MIND and two other heuristic approaches, HCN (High Computation Neighbor) and HCNN (High Computation Nearest Neighbor). In HCN, each requesting UAV simply sends its inference request to the immediate neighbor with the highest computation capacity to potentially reduce the computation and inference delay. However, HCN does not take into account the distance between each UAV and its neighbors which may result in large communication delay of the offloaded data and, therefore, high inference latency. In the HCNN baseline, the source UAV sends its inference request to the nearest neighbor with the highest computation capacity. It can be seen that with 6 or more inference requests, the inference latency due to HCNN is higher than that of the HCN. This is probably because of the shortage of computation resources in the proximity of the source UAVs. Note that GAME-MIND consistently outperforms both heuristics by maintaining lower inference latency across all studied scenarios. Specifically, for settings with lower computation capacity (S_i=10 GFLOPs), both HCN and HCNN experience significant increases in inference latency as the number of inference requests grows, with HCN performing the worst. The steep rise in inference latency for HCN indicates inefficient workload allocation and resource utilization of this baseline. HCNN performs better than HCN due to its improved computation offloading mechanism, but it still lags behind GAME-MIND, particularly at higher inference loads.

It can be observed that GAME-MIND achieves significantly lower latency compared to those due to the two baselines. The latency of GAME-MIND remains substantially lower, especially when UAVs have higher computation capacity (S_i=20 GFLOPs). For S_i=20 GFLOPs, GAME-MIND exhibits the best performance, with a gradual and controlled increase in latency as the number of inference requests increases, highlighting the efficiency of GAME-MIND in balancing computation and communication resource management. This comparison confirms that GAME-MIND is more scalable and robust than heuristic approaches, particularly in high-load scenarios.

Figure 4.8 compares the inference latency of the proposed GAME-MIND framework with two heuristic-based methods, HCN and HCNN, for networks with different numbers of UAVs. The results indicate that GAME-MIND consistently outperforms both HCN and HCNN across all considered values of N in terms of inference latency. Specifically, as the number of inference requests increases, the inference latency due to GAME-MIND grows at a significantly slower rate compared to those due to the heuristic methods. This advantage is attributed to the adaptive and decentralized optimization strategy employed by GAME-MIND, which efficiently allocates computation resources and offloads computation loads associated with the DNN layers.

For the small network size ($N = 10$), Figure 4.8 shows that the inference latency due to all strategies is relatively high due to the limited computation resources available for processing the DNN inference requests. Among the heuristics, HCNN performs slightly better than HCN, as it likely employs a more efficient task allocation strategy. However, both heuristics still exhibit higher latency compared to GAME-MIND, which better distributes the workload among UAVs. As N increases to 15 and 20, the performance gap between GAME-MIND and the heuristic methods is still significant. The additional UAVs in the network provide greater computation capacity, allowing GAME-MIND to more effectively balance inference tasks in comparison to HCN and HCNN, leading to reduced latency and improved scalability. For the networks with $N = 10$ and 15 UAVs, the two heuristic-based methods HCN and HCNN have inference latency at least twice of that due to GAME-MIND in the high inference load conditions where the numbers of inference requests equal 10 and 15, respectively.

Furthermore, the heuristic-based methods show a steep increase in inference latency as the number of requests grows beyond 10. This suggests that these heuristics face significant limitations in handling high-load scenarios, due to suboptimal resource allocation and increased congestion within

the network. On the other hand, GAME-MIND demonstrates a more gradual increase in inference latency, highlighting its robustness in managing varying inference loads. Notably, for $N = 20$, GAME-MIND maintains significantly lower latency compared to the two heuristics, reinforcing the benefits of deploying more UAVs in the network when using our proposed GAME-MIND algorithm. Overall, the results confirm that GAME-MIND provides superior inference latency performance compared to heuristic-based methods, especially as the inference load increases. The distributed and intelligent task allocation approach in GAME-MIND enables it to efficiently scale with the number of UAVs, making it a more effective solution for real-time DNN inference in UAV-assisted networks.

Next we compare GAME-MIND with DINF [98]. In DINF, the distribution of DNN layers among the available UAVs follows a greedy strategy, where layers are assigned sequentially to the swarm. For each layer, the UAV that provides the lowest latency while maintaining the highest remaining computational capacity is selected. However, the selected UAV must satisfy both memory and computational resource constraints. This greedy allocation method operates without prior knowledge of the computational demand of individual layers. To guide the selection process, a normalized metric is defined as $\text{nrm}(i) = \alpha \cdot t(j) + \beta/\bar{c}_i$, which introduces a trade-off between latency and residual computational power. Once a UAV is selected based on this metric, it is checked against its current memory and computational resource availability. If the resource constraints are not met, the UAV is excluded from the allocation process. Consequently, any UAV that becomes resource-constrained during allocation is removed from further consideration.

Figure 4.9 illustrates the inference latency of the baseline (DINF) and GAME-MIND as the number of DNN inference requests increases, under varying UAV computational capacities ($S_i = 10, 15, 20$ GFLOPs) and processing quotas ($\zeta = 15, 22, 30$), with $N = 10$ UAVs. Across all configurations, GAME-MIND consistently outperforms DINF by achieving significantly lower latency, demonstrating its effectiveness in managing computation offloading and resource allocation. As the number of inference requests increases, the latency of both algorithms rises; however, GAME-MIND exhibits a more gradual increase, indicating better scalability. Higher computational capacity and quota values further reduce latency for both algorithms, but GAME-MIND benefits more from these enhancements due to its ability to exploit additional resources more effectively. Notably, DINF exhibits a latency plateau at higher request levels, suggesting that UAVs hit their processing or quota limits, whereas GAME-MIND maintains a steady performance, avoiding saturation. These results

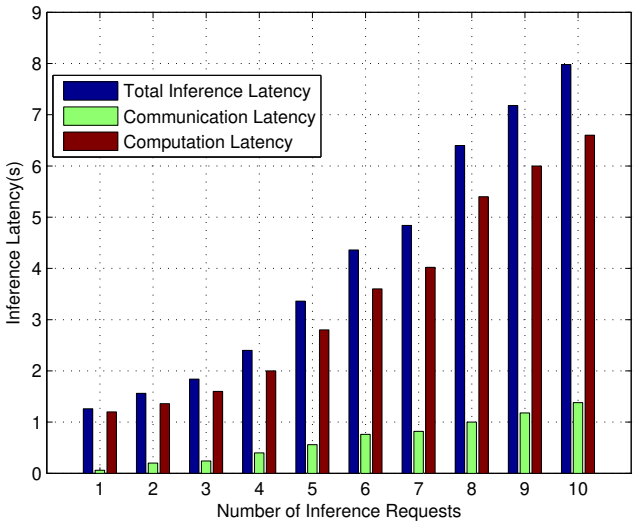

Figure 4.11: Inference, communications, and computation latency of GAME-MIND vs number of inference requests for N=10, ζ = 15 and S_i = 10 GFLOPs

confirm that GAME-MIND provides superior performance in terms of inference latency, especially under heavy workloads, making it a robust and scalable solution for UAV-based edge computing.

Figure 4.10 presents the inference latency of the baseline DINF and GAME-MIND as the number of DNN inference requests increases, evaluated under different numbers of participating UAVs ($N = 10, 15, 20$). The results clearly demonstrate that GAME-MIND consistently achieves lower inference latency than DINF across all network sizes, showcasing its effectiveness in distributing computation resources among UAVs. As the number of UAVs increases, the latency for both algorithms decreases due to the limited resources available for parallel processing. However, GAME-MIND benefits more significantly from the increase in N, maintaining smoother and more scalable performance. In contrast, DINF exhibits higher latency and earlier saturation, particularly for smaller N, highlighting its limitations in adapting to growing task loads. These findings confirm that the proposed GAME-MIND algorithm is better suited for dynamic UAV networks, offering improved scalability and reduced inference latency through intelligent task offloading and resource management. Therefore, we would like to provide more detailed studies on the delay performance of GAME-MIND in the following.

Figure 4.11 demonstrates the total inference, communication, and computation latency due to GAME-MIND for different number of inference requests. It can be observed that as the number of inference requests increases from one to four, the inference delay increases gradually because

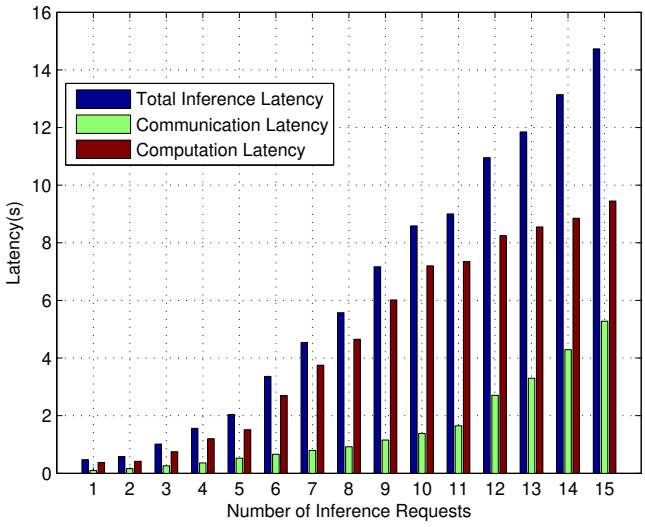

Figure 4.12: Inference, communication, and computation latency of GAME-MIND vs number of inference requests for S_i = 10 GFLOPs, ζ = 15 and N=15

the computation and communication resources are quite underutilized in this range. By efficiently distributing the computation tasks associated with different DNN layers across the UAVs, GAME-MIND ensures that UAVs are not overloaded. However, the delay increases noticeably but steadily as the number of inference requests increases from four and ten. This is because the need for computation and communication resources increases with larger number of inference requests. In this regime, the larger cumulative load begins impacting the overall inference latency more significantly. In fact, the computation and communication latency grows with additional inference requests, gradually increasing the total inference latency.

We present the total inference, communication, and computation latency achieved by GAME-MIND for a larger network size with $N = 15$ UAVs and the number of inference requests up to 15 in Figure 4.12. In fact, networks with a larger number of UAVs could potentially provide more computing resources to better support inference requests but at the cost of high communication latency. This is because UAVs with limited computation capacity may not be able to process all DNN layers and have to offload certain DNN layers and transfer the required data to their neighboring UAVs. Despite increasing the number of UAVs, which helps better distribute the workload, the total inference latency still grows significantly as the number of inference requests increases, suggesting that computation remains the primary bottleneck in the considered settings. While communication latency remains relatively low for a small number of inference requests, it

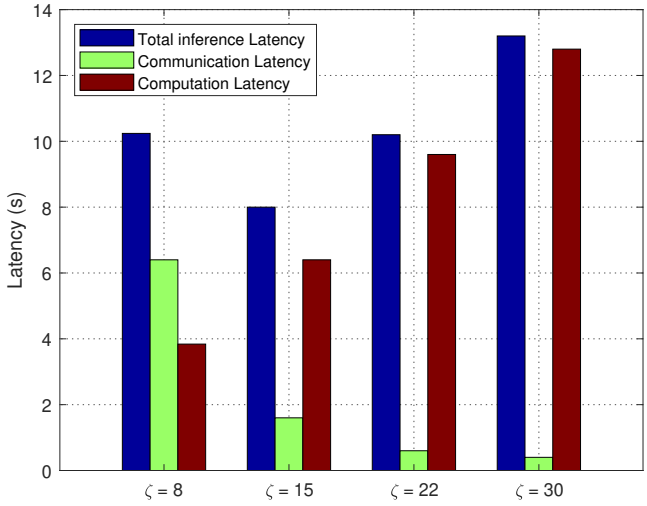

Figure 4.13: Inference, communication, and computation latency of GAME-MIND for different values of processing quota ζ, $R = 10$, $N = 10$ and $S_i = 10$

increases significantly as the number of inference requests becomes larger than 11. This trend suggests that as the system must handle more inference requests, it could experience shortage of radio resources, leading to higher communication latency. Even with 15 UAVs, the relatively limited computation capacity (10 GFLOPs per UAV) means that UAVs must take longer to process assigned computation tasks, causing a backlog in communication data and larger inference delay. The sharp rise in communication latency beyond 11 inference requests indicates a critical threshold beyond which the system's communication resources become a limiting factor, impacting overall latency performance.

Figure 4.13 illustrates the impact of UAVs' processing quota ζ on the communication, computation, and total inference latency due to GAME-MIND. As ζ increases from 8 to 30, the communication latency decreases because with higher processing quota, UAVs can accept and process more DNN layers locally, reducing the need for task offloading to distant UAVs. However, this leads to a rise in computation latency, as fewer UAVs handle more layers, increasing processing load and prolonging execution time. For low quota values (e.g., $\zeta = 8$), UAVs reach their allowed limits quickly, forcing computation offloading and data transmissions to their neighboring UAVs, which results in higher communication delay but relatively small computation latency. At higher quotas (e.g., $\zeta = 30$), UAVs take on more DNN layers rather than offloading to neighboring UAVs, leading to smaller communication latency but significantly increased computation delay, causing the

highest total inference latency (about 14s). The results suggest that lower quota values cause communication bottlenecks, while higher quotas can lead to computational overload, causing excessive computation latency. A good balance between communications and computation can be achieved with $\zeta = 15$, leading the best inference latency for the considered settings.

Chapter 5

Conclusion and Future Work

5.1 Conclusion Remarks

In this thesis, we have addressed the computation offloading and resource allocation problem, in a UAV based edge computing environment from which we made some major contributions. In addition, we have performed a detailed literature survey which categorizes the works based on the design approaches.

In particular, we formulated the optimization problem of DNN layer assignments and resource allocation, considering the connectivity between UAVs to ensure reliable data transfer of intermediate layer outputs and the constraints on the limited computational capacity of individual UAVs. Our formulation aims to minimize overall inference latency for all DNN inference requests. Then, a novel framework, called LARA, was developed by using the alternating optimization techniques, which significantly outperforms two heuristic baselines, namely HCN and HCNN. Moreover, we showed that the LARA framework achieves the latency performance close to that due to the near-optimal solution (from the baseline called MinQRS), which is obtained by solving the whole optimization problem with a solver without decomposing it into multiple subproblems as in LARA.

Additionally, we introduced GAME-MIND, a decentralized matching game theory-based algorithm designed to allocate DNN layers efficiently among UAVs while balancing the network load. Our simulation results demonstrated that GAME-MIND significantly reduces inference delays compared to conventional heuristic methods, ensuring faster and more reliable real-time decision-making

in UAV networks. By considering key factors such as computation capacity limitations, network connectivity, and sequential task dependencies of DNN inference, our proposed GAME-MIND framework enhances the scalability and adaptability of DNN inference in dynamic environments.

5.2 Future Research Directions

While this thesis provides substantial contributions toward efficient computation offloading and resource allocation design for deep neural network (DNN) inference in UAV-based edge computing environments, several avenues remain open for future exploration and enhancement.

Future research can build upon the foundations laid in this thesis by exploring several promising directions to further enhance the efficiency and robustness of computation offloading and DNN inference in UAV-based edge computing. One important extension involves incorporating energy efficiency into the optimization framework, given the critical battery constraints in UAV operations. Adapting the proposed methods to dynamic environments with real-time mobility, changing network topologies, and varying computational workloads would improve practical applicability. The integration of federated learning could enable decentralized model training and updates, preserving data privacy while reducing communication overhead.

Addressing security and privacy challenges, such as secure data transmission and defense against adversarial attacks, is also essential for sensitive applications. Moreover, extending the framework to support heterogeneous UAV networks with diverse hardware capabilities and resource constraints would enhance its applicability. Hardware-aware optimization that considers device-specific limitations, such as memory or processing power, could lead to more efficient deployments. Finally, real-world implementation and testing, either through actual UAV networks or high-fidelity simulations, would validate the effectiveness of the proposed solutions and offer valuable insights for practical deployment in scenarios like disaster response, environmental monitoring, and smart surveillance.

5.3 List of Publications

1. Muhammad Ismail and Long Bao Le, "Computation Offloading and Resource Allocation for Deep Neural Network Inference in UAV Wireless Networks," in *Proc. IEEE ICC 2025*, June 2025.

2. Muhammad Ismail and Long Bao Le, "Collaborative Offloading and Resource Allocation for Efficient Inference of Deep Neural Networks in Wireless Networks," journal under submission.

References

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Sep. 2017.

[2] M. Baysan, K. Sarac, R. Chandrasekaran, and S. Bereg, "A polynomial time solution to minimum forwarding set problem in wireless networks under unit disk coverage model," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 7, pp. 913–924, Jul. 2008.

[3] D. Gale and L. Shapley, "College admissions and the stability of marriage," *Amer. Math. Monthly*, vol. 39, pp. 9–15, Jan. 1962.

[4] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," pp. 68–73, Dec. 2008.

[5] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. Int. Conf. Mobile Syst., Appl. Services*, Jun. 2010, pp. 49–62.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. MCC Workshop Mobile Cloud Compt.*, Aug. 2012, pp. 13–16.

[7] M. Finnegan, "Boeing 787s to create half a terabyte of data per flight, says virgin atlantic," *Computerworld UK*, vol. 6, pp. 1–2, Sep. 2013.

[8] [Online]. Available: http://www.computerworlduk.com/news/data/boeing-787screate-half-terabyte-of-data-per-flight-says-virgin-atlantic-3433595/.

[9] L. Gupta, R. Jain, and G. Vaszkun, "Survey of important issues in uav communication networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1123–1152, Nov. 2015.

[10] J. Zhou, X. Hou, Y. Zeng, P. Cong, W. Jiang, and S. Guo, "Quality of experience and reliability-aware task offloading and scheduling for multi-user mobile-edge computing systems," *IEEE Trans. Services Comput.*, Jul. 2025.

[11] B. Liu, C. Liu, and M. Peng, "Computation offloading and resource allocation in unmanned aerial vehicle networks," *IEEE Trans. Veh. Technol.*, vol. 72, no. 4, pp. 4981–4995, Nov. 2022.

[12] M. Yan, R. Xiong, Y. Wang, and C. Li, "Edge computing task offloading optimization for a uav-assisted internet of vehicles via deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 73, no. 4, pp. 5647–5658, Nov. 2023.

[13] H. Wang, "Collaborative task offloading strategy of uav cluster using improved genetic algorithm in mobile edge computing," *J. Robotics*, vol. 2021, no. 1, p. 3965689, Dec. 2021.

[14] H. Yuan, M. Wang, J. Bi, S. Shi, J. Yang, J. Zhang, M. Zhou, and R. Buyya, "Cost-efficient task offloading in mobile edge computing with layered unmanned aerial vehicles," *IEEE Internet Things J.*, Oct. 2024.

[15] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "Sustainable task offloading in uav networks via multi-agent reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 5, pp. 5003–5015, May. 2021.

[16] M. Messous, S. Senouci, H. Sedjelmaci, and S. Cherkaoui, "A game theory based efficient computation offloading in an uav network," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4964–4974, May. 2019.

[17] T. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Nov. 2018.

[18] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Proce. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[19] Y. Mao, J. Zhang, and K. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *Proc. IEEE Wirel. Commun. Netw. Conf. (WCNC)*. IEEE, Jan. 2017, pp. 1–6.

[20] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wirel. Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2016.

[21] Z. Song, Y. Liu, and X. Sun, "Joint radio and computational resource allocation for noma-based mobile edge computing in heterogeneous networks," *IEEE Commun. Let.*, vol. 22, no. 12, pp. 2559–2562, Dec. 2018.

[22] P. Liu, G. Xu, K. Yang, K. Wang, and X. Meng, "Jointly optimized energy-minimal resource allocation in cache-enhanced mobile edge computing systems," *IEEE Access*, vol. 7, pp. 3336–3347, Jan. 2018.

[23] H. Yu, Q. Wang, and S. Guo, "Energy-efficient task offloading and resource scheduling for mobile edge computing," in *IEEE Int. Conf. Netw. Architect. Storage (NAS)*. IEEE, Jan. 2018, pp. 1–4.

[24] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Apr. 2018.

[25] Z. Tan, F. Yu, X. Li, H. Ji, and V. Leung, "Virtual resource allocation for heterogeneous services in full duplex-enabled scns with mobile edge computing and caching," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1794–1808, Oct. 2017.

[26] M. Liu, F. Yu, Y. Teng, V. Leung, and M. Song, "Distributed resource allocation in blockchain-based video streaming systems with mobile edge computing," *IEEE Trans. Wirel. Commun.*, vol. 18, no. 1, pp. 695–708, Dec. 2018.

[27] M. Salmani and T. Davidson, "Uplink resource allocation for multiple access computational offloading," *Signal Processing*, vol. 168, p. 107322, Mar. 2020.

[28] M. Alkhalaileh, R. Calheiros, Q. Nguyen, and B. Javadi, "Data-intensive application scheduling on mobile edge cloud computing," *J. Netw. Comput. Appl.*, vol. 167, p. 102735, Oct. 2020.

[29] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, Nov. 2021.

[30] M. Wu, Q. Song, L. Guo, and I. Lee, "Energy-efficient secure computation offloading in wireless powered mobile edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 72, no. 5, pp. 6907–6912, May. 2023.

[31] A. Younis, S. Maheshwari, and D. Pompili, "Energy-latency computation offloading and approximate computing in mobile-edge computing networks," *IEEE Trans. Netw. Serv. Management*, vol. 21, no. 3, pp. 3401–3415, Jan. 2024.

[32] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, Aug. 2014.

[33] Y. Yang, Y. Ma, W. Xiang, X. Gu, and H. Zhao, "Joint optimization of energy consumption and packet scheduling for mobile edge computing in cyber-physical networks," *IEEE Access*, vol. 6, pp. 15 576–15 586, Apr. 2018.

[34] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, "Cost efficient scheduling for delay-sensitive tasks in edge computing system," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*. IEEE, Jan. 2018, pp. 73–80.

[35] Y. Shao, C. Li, Z. Fu, L. Jia, and Y. Luo, "Cost-effective replication management and scheduling in edge computing," *J. Netw. Comput. Appl.*, vol. 129, pp. 46–61, Mar. 2019.

[36] Y. Li, H. Ma, L. Wang, S. Mao, and G. Wang, "Optimized content caching and user association for edge computing in densely deployed heterogeneous networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 2130–2142, Jun. 2020.

[37] C. Li, J. Tang, H. Tang, and Y. Luo, "Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment," *Future Gen. Comput. Syst.*, vol. 95, pp. 249–264, Jun. 2019.

[38] C. Li, J. Bai, Y. Chen, and Y. Luo, "Resource and replica management strategy for optimizing financial cost and user experience in edge cloud computing system," *Inf. Sciences*, vol. 516, pp. 33–55, Apr. 2020.

[39] M. Ghobaei-Arani, A. Souri, F. Safara, and M. Norouzi, "An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing," *Trans. Emerging Telecom. Technol.*, vol. 31, no. 2, pp. 37–70, Jan. 2020.

[40] L. Yang, C. Zhong, Q. Yang, W. Zou, and A. Fathalla, "Task offloading for directed acyclic graph applications based on edge computing in industrial internet," *Inf. Sciences*, vol. 540, pp. 51–68, Nov. 2020.

[41] J. Mei, Z. Tong, K. Li, L. Zhang, and K. Li, "Energy-efficient heuristic computation offloading with delay constraints in mobile edge computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 4404–4417, Dec. 2023.

[42] J. Zhang, W. Xia, F. Yan, and L. Shen, "Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing," *IEEE Access*, vol. 6, pp. 19 324–19 337, Apr. 2018.

[43] X. Guan, J. Yin, X. Wan, T. Wang, and G. Bai, "A stackelberg game model for dynamic resource scheduling in edge computing with cooperative cloudlets," in *Proc. IEEE Int. Conf. Sens., Commun. & Netw. (SECON)*. IEEE, May. 2018, pp. 1–2.

[44] Y. Jie, X. Tang, K. Choo, S. Su, M. Li, and C. Guo, "Online task scheduling for edge computing based on repeated stackelberg game," *J. Parallel & Distrib. Comput.*, vol. 122, pp. 159–172, Dec. 2018.

[45] Z. Zhu, J. Peng, X. Gu, H. Li, K. Liu, Z. Zhou, and W. Liu, "Fair resource allocation for system throughput maximization in mobile edge computing," *IEEE Access*, vol. 6, pp. 5332–5340, Feb. 2018.

[46] J. Zhou, X. Zhang, and W. Wang, "Joint resource allocation and user association for heterogeneous services in multi-access edge computing networks," *IEEE Access*, vol. 7, pp. 12 272–12 282, Feb. 2019.

[47] Q. Pham, H. Nguyen, Z. Han, and W. Hwang, "Coalitional games for computation offloading in noma-enabled multi-access edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1982–1993, Feb. 2019.

[48] C. Yi, J. Cai, and Z. Su, "A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications," *IEEE Trans. Mobile Comput.*, vol. 19, no. 1, pp. 29–43, Dec. 2019.

[49] Y. Cui, D. Zhang, T. Zhang, L. Chen, M. Piao, and H. Zhu, "Novel method of mobile edge computation offloading based on evolutionary game strategy for iot devices," *AEU-Int. J. Elect. Commun.*, vol. 118, p. 153134, May. 2020.

[50] J. Hu, K. Li, C. Liu, and K. Li, "Game-based task offloading of multiple mobile devices with qos in mobile edge computing systems of limited computation capacity," *ACM Trans. Embed. Comput. Syst. (TECS)*, vol. 19, no. 4, pp. 1–21, Jul. 2020.

[51] M. Hu, Z. Xie, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Heterogeneous edge offloading with incomplete information: A minority game approach," *IEEE Trans. Parallel & Distrib. Syst.*, vol. 31, no. 9, pp. 2139–2154, Sep. 2020.

[52] M. Liwang, J. Wang, Z. Gao, X. Du, and M. Guizani, "Game theory based opportunistic computation offloading in cloud-enabled iov," *IEEE Access*, vol. 7, pp. 32 551–32 561, Mar. 2019.

[53] Y. Ding, K. Li, C. Liu, and K. Li, "A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1503–1519, Sep. 2021.

[54] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, Sep. 2004.

[55] A. Nemirovski, *Interior-Point Polynomial Algorithms in Convex Programming*. Atlanta, GA, USA: Georgia Institute of Technology, Mar. 2004, lecture Notes.

[56] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. SIAM, Oct. 2001.

[57] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Science & Business Media, Jun. 2006.

[58] M. Grant and S. Boyd, *CVX: Matlab Software for Disciplined Convex Programming, Version 2.1*. [Online]. Available: http://cvxr.com/cvx/, Mar. 2014.

[59] MOSEK ApS, *The MOSEK Optimization Toolbox for MATLAB Manual, Version 9.0*, May. 2019, [Online]. [Online]. Available: http://docs.mosek.com/9.0/toolbox/index.html

[60] K. Zhang, "Task offloading and resource allocation using deep reinforcement learning," Ph.D. dissertation, Université d'Ottawa/University of Ottawa, Sep. 2020.

[61] S. H. and J. S., "Memory, long short-term," *Neural Comput.*, vol. 9, no. 8, p. 1735, Nov. 1997.

[62] Y. Narahari, *Game Theory and Mechanism Design*. World Scientific, Apr. 2014, vol. 4.

[63] A. Roth, "The college admissions problem is not equivalent to the marriage problem," *J. Economic Theory*, vol. 36, no. 2, pp. 277–288, Dec. 1985.

[64] F. Y. et al., "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2633–2642.

[65] M. W. Libbrecht and W. S. Noble, "Machine learning applications in genetics and genomics," *Nature Reviews Genetics*, vol. 16, no. 6, pp. 321–332, Jun. 2015.

[66] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam, and M. Debbah, "A tutorial on uavs for wireless networks: Applications, challenges, and open problems," *IEEE Commun. Surveys Tut.*, vol. 21, no. 3, pp. 2334–2360, Mar. 2019.

[67] S. Hu, W. Ni, X. Wang, A. Jamalipour, and D. Ta, "Joint optimization of trajectory, propulsion, and thrust powers for covert uav-on-uav video tracking and surveillance," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 1959–1972, Jan. 2021.

[68] S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou, "Networking for big data: A survey," *IEEE Commun. Surveys Tut.*, vol. 19, no. 1, pp. 531–549, Mar. 2017.

[69] M. Jouhari, A. Al-Ali, E. Baccour, A. Mohamed, A. Erbad, M. Guizani, and M. Hamdi, "Distributed cnn inference on resource-constrained uavs for surveillance systems: Design and optimization," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1227–1242, Nov. 2021.

[70] H. Huang, Y. Yang, H. Wang, Z. Ding, H. Sari, and F. Adachi, "Deep reinforcement learning for uav navigation through massive mimo technique," *IEEE Trans. Veh. Technol.*, vol. 69, no. 1, pp. 1117–1121, Jan. 2020.

[71] Z. Zhou, L. Pan, and S. Liu, "Potential game-based computation offloading in edge computing with heterogeneous edge servers," *IEEE Trans. Netw. Science Engineer.*, vol. 12, no. 1, pp. 290–301, Dec. 2024.

[72] L. Long, Z. Liu, Y. Zhou, L. Liu, J. Shi, and Q. Sun, "Delay optimized computation offloading and resource allocation for mobile edge computing," in *Proc. IEEE 90th Veh. Technol. Conf. (VTC)*, Sep. 2019, pp. 1–5.

[73] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys & Tut.*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.

[74] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, "Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 360–374, Jan. 2020.

[75] S. Jošilo and G. Dán, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 667–680, Feb. 2020.

[76] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-edge computing with computation capacity constraints," *IEEE Wireless Commun. Lett.*, vol. 7, no. 3, pp. 420–423, Jun. 2017.

[77] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, Sep. 2018.

[78] C. Li, H. Wang, and R. Song, "Intelligent offloading for noma-assisted mec via dual connectivity," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2802–2813, Sep. 2020.

[79] H. Chen, D. Zhao, Q. Chen, and R. Chai, "Joint computation offloading and radio resource allocations in wireless cellular networks," in *Proc. Conf. Wireless Commun. Signal Process.*, Mar. 2018, pp. 1–6.

[80] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2651–2664, Oct. 2018.

[81] L. Huang, L. Zhang, S. Yang, L. P. Qian, and Y. Wu, "Meta-learning based dynamic computation task offloading for mobile edge computing networks," *IEEE Commun. Lett.*, vol. 25, no. 5, pp. 1568–1572, Dec. 2020.

[82] N. Abbas, W. Fawaz, S. Sharafeddine, A. Mourad, and C. Abou-Rjeily, "Svm-based task admission control and computation offloading using lyapunov optimization in heterogeneous mec network," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 3121–3135, Mar. 2022.

[83] M. Merluzzi, P. D. Lorenzo, S. Barbarossa, and V. Frascolla, "Dynamic computation offloading in multi-access edge computing via ultra-reliable and low-latency communications," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 6, pp. 342–356, Mar. 2020.

[84] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5g beyond," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12 175–12 186, Aug. 2020.

[85] S. Li, S. Zhang, Z. Wang, Z. Zhou, X. Wang, S. Mumtaz, M. Guizani, and V. Frascolla, "Asynchronous fdrl-based low-latency computation offloading for integrated terrestrial and non-terrestrial power iot," *IEEE Netw.*, vol. 37, no. 5, pp. 33–41, Feb. 2023.

[86] J. Park and K. Chung, "Rl-based computation offloading scheme for improving qoe in edge computing environments," in *Proc. IEEE 9th World Forum on Internet of Things*, May. 2023, pp. 1–6.

[87] S. Zhou, S. Fei, and Y. Feng, "Deep reinforcement learning based uav-assisted maritime network computation offloading strategy," in *Proc. IEEE/CIC Int. Conf. Commun. China*, Jun. 2022, pp. 890–895.

[88] N. Fofana, A. B. Letaifa, and A. Rachedi, "Intelligent task offloading in vehicular networks: a deep reinforcement learning perspective," *IEEE Trans. Veh. Technol.*, vol. 74, no. 1, pp. 201 – 216, Jan. 2025.

[89] J. Zhang, X. Deng, J. Gui, X. Chen, S. Wan, and G. Min, "Personalized cloud gaming: Multi-objective optimization for resource utilization and video encoding," *IEEE Trans. Cloud Comput.*, pp. 1–14, Feb. 2025.

[90] Y. Zhang and M. Guizani, *Game Theory for Wireless Communications and Networking.* CRC press, Jun. 2011.

[91] Z. He, Y. Guo, X. Zhai, M. Zhao, W. Zhou, and K. Li, "Joint computation offloading and resource allocation in mobile-edge cloud computing: A two-layer game approach," *IEEE Trans. Cloud Comput.*, vol. 13, no. 1, pp. 411–428, Jan. 2025.

[92] F. Tütüncüoğlu and G. Dán, "Joint resource management and pricing for task offloading in serverless edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 7438–7452, Nov. 2024.

[93] T. Lyu, H. Xu, and Z. Han, "Multi-leader multi-follower stackelberg game based resource allocation in multi-access edge computing," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2022, pp. 4306–4311.

[94] L. Zhang and C. Bu, "Stackelberg game-driven computational offloading in edge computing scenarios," in *Proc. Int. Conf. Commun. Softw. Netw.*, May. 2023, pp. 161–165.

[95] S. Kim, "Bargaining game based offloading service algorithm for edge-assisted distributed computing model," *IEEE Access*, vol. 10, pp. 63 648–63 657, Jun. 2022.

[96] M. D. Konovalov and L. A. Uvarova, "Application of reinforcement learning algorithms and game theory to optimize the offloading of a mobile edge computing system," in *Proc. Int. Conf. Qual. Manag., Transport Inf. Secur., Inf. Technol.*, Jul. 2023, pp. 23–26.

[97] P. Teymoori and A. Boukerche, "Dynamic multi-user computation offloading for mobile edge computing using game theory and deep reinforcement learning," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2022, pp. 1930–1935.

[98] M. Dhuheir, E. Baccour, A. Erbad, S. Sabeeh, and M. Hamdi, "Efficient real-time image recognition using collaborative swarm of uavs and convolutional networks," in *Proc. Int. Wirel. Commun. Mobile Comput.*, Sep. 2021, pp. 1954–1959.

[99] M. Ismail and L. B. Le, "Computation offloading and resource allocation for deep neural network inference in uav wireless networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2025.