EURECOM
Sophia Antipolis

SORBONNE UNIVERSITÉ

Structured Distributed Computation under Communication and Computation Constraints

Dissertation

submitted to

Sorbonne University

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Author:

Ahmad Tanha

Reviewer	**Dr. Rawad Bitar**	Technical University of Munich, DE
Reviewer	**Prof. Ana Bušić**	Inria Paris Centre and DI ENS, FR
Examiner	**Dr. Maxime Guillaud**	INRIA Lyon, FR
Jury President/Examiner	**Prof. Melek Önen**	EURECOM, FR
Thesis Co-supervisor	**Prof. Derya Malak**	EURECOM, FR
Thesis Supervisor	**Prof. David Gesbert**	EURECOM, FR

The research has been conducted in the Communication Systems Department at EURECOM (Sophia Antipolis, France) from January 2023 to June 2026.

Manuscript compiled with pdfLaTeX on September 10, 2026.

EURECOM
Sophia Antipolis

SORBONNE UNIVERSITÉ

Calcul distribué structuré sous contraintes de communication et de calcul

Thèse

présentée à

Sorbonne Université

pour l'obtention du grade de docteur
en Sciences et Technologies de l'Information et de la Communication

Auteur :

Ahmad Tanha

Rapporteur	**Dr Rawad Bitar**	Technical University of Munich, DE
Rapporteur	**Pr Ana Bušić**	Inria Paris Centre et DI ENS, FR
Examinateur	**Dr Maxime Guillaud**	INRIA Lyon, FR
Président du jury/Examinateur	**Pr Melek Önen**	EURECOM, FR
Co-encadrant de thèse	**Pr Derya Malak**	EURECOM, FR
Directeur de thèse	**Pr David Gesbert**	EURECOM, FR

Les travaux de recherche présentés dans cette thèse ont été réalisés au sein du département Systèmes de Communication d'EURECOM (Sophia Antipolis, France) de janvier 2023 à juin 2026.

Manuscrit compilé avec pdfLaTeX le 10 septembre 2026.

Dedication

To whom it may concern,

my true friends,

and my family.

In Memoriam

In memory of Ali Maniei, my friend and former colleague during our Bachelor's studies at K. N. Toosi University of Technology, whose scientific attitude, curiosity, dedication, and encouragement inspired me to pursue a research career. He passed away far too soon at age $\simeq 24$, during his Master's studies at Sharif University of Technology, but his memory remains with us forever.

Here is his last self-portrait before leaving us:

"What is not surrounded by uncertainty
cannot be the truth."

Richard P. Feynman

"You cannot discover new oceans unless you have
the courage to lose sight of the shore."

André Gide

Abstract

Distributed computing has emerged as a key enabler of modern communication and data-driven systems and networks, supporting large-scale applications such as wireless networks, machine learning, and real-time data processing. As data volumes and computational demands continue to grow, distributing computations across multiple users and processing nodes has become increasingly essential. In such systems, servers must collaboratively execute computational tasks while exchanging intermediate results over communication links with limited bandwidth. This intrinsic interplay between local computation in servers and information transmission from the servers to the user(s) leads to a fundamental challenge: how to efficiently balance computation and communication costs in multi-user distributed computing systems, which naturally introduces a trade-off between computation and communication costs.

Existing approaches have mainly studied this trade-off for linear and linearly-separable computations, and matrix multiplications. However, we examine a broader class of functions (non-linearly separable functions) as well by leveraging their structural properties. Particularly, this thesis investigates the communication-computation trade-off in a distributed computing framework with a master node that coordinates multiple servers to perform user demand(s) through three complementary approaches, detailed below.

Sensitivity-based Approach: We study the role of the influence of datasets in the distributed computation of Boolean functions. By adopting a sensitivity-based perspective, we show that the placement of input datasets across servers fundamentally impacts both computation and communication costs. Leveraging tools from Boolean function analysis, we derive bounds and design schemes that enable more efficient task allocation than the state of the art by prioritizing highly influential datasets, thereby reducing redundant computations and communication overheads.

Sparse Tensor Factorization Approach: We address the problem of non-linearly separable distributed computation by introducing a sparse tensor factorization framework. In this setting, users demand arbitrary real-valued multivariate polynomials, whose coefficients are modeled by high-dimensional

tensors. We develop achievable schemes based on fixed-support tensor decompositions and structured multi-dimensional tiling, complemented by combinatorial assignment techniques that eliminate redundant computations across servers. This approach significantly reduces the required number of servers compared to classical matrix factorization-based methods, highlighting the importance of exploiting higher-dimensional structure in distributed systems.

Straggler-Resilient Coded Distributed Matrix Multiplication: We study distributed matrix multiplication in the presence of straggling workers. Building on prior structured source-coding techniques for distributed matrix computation, we consider a distributed structured PolyDot (DSPolyDot) code construction that combines structured source-side processing with polynomial-based coded distributed computing. The resulting scheme enables recovery of the desired matrix product from a sufficiently large subset of worker responses, thereby removing the need to wait for the slowest workers while preserving the asymptotic orders of the communication and computation requirements.

Overall, this thesis establishes new connections among distributed computing, Boolean function analysis, tensor-based methods, information theory, source coding, and coding theory.

Résumé

Le calcul distribué s'est imposé comme un élément clé des systèmes et réseaux modernes de communication et de traitement des données, soutenant des applications à grande échelle telles que les réseaux sans fil, l'apprentissage automatique et le traitement de données en temps réel. À mesure que les volumes de données et les besoins de calcul continuent de croître, la répartition des tâches computationnelles entre plusieurs utilisateurs et nœuds de traitement devient de plus en plus indispensable. Dans de tels systèmes, les serveurs doivent exécuter de manière collaborative des tâches de calcul tout en échangeant des résultats intermédiaires via des liens de communication à bande passante limitée. Cette interaction intrinsèque entre le calcul local effectué par les serveurs et la transmission d'information des serveurs vers le(s) utilisateur(s) conduit à un défi fondamental : comment équilibrer efficacement les coûts de calcul et de communication dans les systèmes de calcul distribué multi-utilisateurs, ce qui introduit naturellement un compromis entre ces deux ressources.

Les approches existantes ont principalement étudié ce compromis dans le cadre de calculs linéaires, de fonctions linéairement séparables et de la multiplication matricielle. Toutefois, nous considérons dans cette thèse une classe plus générale de fonctions, incluant les fonctions non linéairement séparables, en exploitant leurs propriétés structurelles. Plus précisément, cette thèse analyse le compromis calcul–communication dans un cadre de calcul distribué comprenant un nœud maître coordonnant plusieurs serveurs afin de satisfaire les demandes des utilisateurs, à travers trois approches complémentaires détaillées ci-dessous.

Approche fondée sur la sensibilité. Nous étudions le rôle de l'influence des ensembles de données dans le calcul distribué de fonctions booléennes. En adoptant une perspective fondée sur la sensibilité, nous montrons que le placement des données d'entrée sur les serveurs impacte de manière fondamentale les coûts de calcul et de communication. En mobilisant des outils issus de l'analyse des fonctions booléennes, nous établissons des bornes théoriques et proposons des schémas d'allocation plus efficaces que l'état de l'art, en

privilégiant les données les plus influentes, ce qui permet de réduire les calculs redondants et les surcoûts de communication.

Approche par factorisation tensorielle creuse. Nous abordons le problème du calcul distribué non linéairement séparable en introduisant un cadre de factorisation tensorielle creuse. Dans ce contexte, les utilisateurs demandent des polynômes multivariés réels arbitraires, dont les coefficients sont modélisés par des tenseurs de grande dimension. Nous développons des schémas réalisables fondés sur des décompositions tensorielles à support fixe et des structures de pavage multidimensionnel, complétés par des techniques combinatoires d'affectation permettant d'éliminer les redondances de calcul entre serveurs. Cette approche réduit significativement le nombre de serveurs requis par rapport aux méthodes classiques fondées sur la factorisation matricielle, mettant en évidence l'intérêt d'exploiter les structures de dimension supérieure dans les systèmes distribués.

Multiplication matricielle distribuée codée et résiliente aux nœuds retardataires. Nous étudions la multiplication matricielle distribuée en présence de nœuds de calcul retardataires. En nous appuyant sur des techniques antérieures de codage de source structuré pour le calcul matriciel distribué, nous considérons une construction de codes PolyDot structurés distribués (DSPolyDot) qui combine un traitement structuré au niveau des sources avec des techniques de calcul distribué codé fondées sur des codes polynomiaux. Le schéma obtenu permet de reconstruire le produit matriciel recherché à partir des réponses d'un sous-ensemble suffisamment grand de nœuds de calcul, évitant ainsi d'attendre les nœuds les plus lents, tout en préservant les ordres asymptotiques des coûts de communication et de calcul.

Dans son ensemble, cette thèse établit de nouveaux liens entre le calcul distribué, l'analyse des fonctions booléennes, les méthodes tensorielles, la théorie de l'information, le codage de source et la théorie des codes.

Acknowledgements

Reaching the end of the doctoral journey, I would like to take a moment to sincerely thank all those who, in one way or another, have supported and accompanied me throughout this path.

First and foremost, I am honored and sincerely grateful to the jury members of my thesis for accepting our invitation. I would like to express my special thanks to Dr. Rawad Bitar and Prof. Ana Bušić for taking the time to review this manuscript and to provide detailed feedback. I am also grateful to Dr. Maxime Guillaud and Prof. Melek Önen for their participation in the examination of this thesis. It was a privilege to benefit from their expertise, insights, and thoughtful evaluations.

I would like to express my sincere appreciation to Prof. David Gesbert, Dean and Director at EURECOM and my thesis supervisor at L'EDITE de Paris (ED130), Sorbonne Université, for his continuous support and for creating an environment that is both intellectually stimulating and genuinely inspiring. His visionary management has made EURECOM a great place where ambitious ideas can be pursued and developed.

This dissertation is a direct result of working with my thesis co-supervisor at EURECOM, Prof. Derya Malak, over the past $\gtrsim 3.5$ years, whom I appreciate for her unwavering support, thoughtful guidance, and high standards. Her proficiency in writing and presenting technical papers has substantially influenced the quality of this thesis. I am especially thankful for her patience and encouragement over the past years. My heartfelt thanks go as well to Prof. Petros Elia for his collaboration, insightful discussions, and valuable feedback, all of which have enriched parts of this thesis. I am equally grateful to Prof. Ali Khalesi at Institut Polytechnique des Sciences Avancées (IPSA) and LINCS Lab for his collaboration and for the insightful exchanges that have contributed to shaping part of this thesis.

Earlier in my academic journey, I had the privilege of working under the supervision of my Master's thesis advisor, Prof. Lotfollah Beygi, at K. N. Toosi University of Technology (KNTU). I would like to sincerely thank him for his mentorship and support. He played a paramount role in sparking my interest

in digital communications, applied coding, and information theory, and his practical perspectives helped me bridge theory with real-world applications in a meaningful way.

I am also thankful to Prof. Mahmoud Ahmadian Attari and Prof. Bahareh Akhbari at KNTU for their inspiring teaching during my graduate studies. Their courses in coding theory and information theory were decisive in shaping my academic interests. Their passion for the subject and depth of knowledge left a lasting impression on me and guided me toward pursuing further studies in this field. Furthermore, during my Master's studies at KNTU, I had the privilege of meeting Dr. Mostafa Ayaz, whose thoughtful and profound scientific discussions had a meaningful impact on my academic journey and led to important milestones along the way. I am also grateful to him for his sincere friendship, intellectual generosity, and the many inspiring exchanges we had during our studies at KNTU and beyond.

I would also like to warmly thank the administrative staff and HR team at EURECOM for their constant support throughout my doctoral studies. I am especially grateful to Madam Sophie Salmon for her kindness, responsiveness, and invaluable assistance in handling administrative matters related to Sorbonne University, including the procedures that ultimately made this thesis possible.

I want to express my deepest gratitude to my family and close friends, including the Iranian community at EURECOM and those around the globe, including Iran. Their unwavering support and encouragement have been my greatest source of resilience and strength throughout this journey. Their belief in me has meant more than words can express.

Last but not least, I want to thank myself for believing in me, for doing all this hard work, for having $\sim$ no days off, for never quitting, for always being a giver and trying to give more than I receive, for trying to do more right than wrong, and for just being me at all times.

Finally, I would like to conclude these acknowledgments with a Persian poem—a quatrain from the *Rubaiyat* of Omar Khayyam—whose words first awakened in me a deep curiosity and a lifelong pursuit of truth.

English version:

"The secrets eternal, neither you know nor I
And answers to the riddle, neither you know nor I
Behind the veil, there is much talk about me and you
When the veil falls, neither you remain nor I."

Persian version:

« اسرار اَزَل را نه تو دانی و نه من
وین حرفِ معمّا، نه تو خوانی و نه من
هست از پس پرده گفت‌وگوی من و تو
چون پرده برافتد، نه تو مانی و نه من »

Biot, September 10, 2026, Ahmad Tanha

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

LLMs	Large Language Models
ML	Machine Learning
CDC	Coded Distributed Computing
LCC	Lagrange Coded Computing
FCCs	Function-Correcting Codes
ECCs	Error-Correcting Codes
i.i.d.	Independent and Identically Distributed
Bern	Bernoulli
TDC	Tessellated Distributed Computing
NP	Non-deterministic Polynomial-time
SVD	Singular Value Decomposition
MLSVD	Multilinear Singular Value Decomposition
TCP	Tensor Contraction Product
DMM	Distributed Matrix Multiplication
Poly	Polynomial Codes
MatDot	MatDot Codes
PolyDot	PolyDot Codes
DSMatDot	Distributed Structured MatDot Codes
DSPolyDot	Distributed Structured PolyDot Codes

Notations

$\mathbb{N}$	Set of natural numbers.
$\mathbb{Z}^+$	Set of positive integers.
$\mathbb{R}$	Field of real numbers.
$\mathbb{F}_q$	Finite field of order q, where q is a prime power. In particular, $\mathbb{F}_2 = \{0, 1\}$.
$\mathbb{F}_q^n$	Set of length-n vectors whose entries belong to $\mathbb{F}_q$.
$[n]$	Set of integers $\{1, \ldots, n\}$, for $n \in \mathbb{Z}^+$.
$[a : b]$	Set of integers $\{a, \ldots, b\}$.
$[[a : b]]$	Ordered set of integers $\{a, a + 1, \ldots, b\}$, for integers $a \leq b$.
$\lvert\mathcal{S}\rvert$	Cardinality of a finite set $\mathcal{S}$.
$\mathcal{S} \subseteq \mathcal{A}$	Indicates that $\mathcal{S}$ is a subset of $\mathcal{A}$.
$\mathcal{A} \cup \mathcal{B}$	Union of the sets $\mathcal{A}$ and $\mathcal{B}$.
$\mathcal{A} \cap \mathcal{B}$	Intersection of the sets $\mathcal{A}$ and $\mathcal{B}$.
$\mathcal{A} \setminus \mathcal{B}$	Set of elements belonging to $\mathcal{A}$ but not to $\mathcal{B}$.
$\Pr\{\mathcal{E}\}$	Probability of an event $\mathcal{E}$.
$\mathbb{1}\{\mathcal{E}\}$	Indicator of an event or condition $\mathcal{E}$, equal to 1 when $\mathcal{E}$ holds and 0 otherwise.
$\mathbb{E}[X]$	Expected value of the random variable X.
$\lceil x \rceil$	Ceiling of a real number x.
$\lfloor x \rfloor$	Floor of a real number x.

x	Scalar quantity or realization of a scalar random variable, depending on the context.
X	Scalar random variable.
$\mathbf{x}$	Column vector.
$\mathbf{x}^\intercal$	Transpose of the column vector $\mathbf{x}$, yielding a row vector.
$\mathbf{X}$	Matrix.
$\mathbf{X}^\intercal$	Transpose of the matrix $\mathbf{X}$.
$\mathbf{X}(i, j)$	Entry of $\mathbf{X}$ located in row i and column j.
$\mathbf{X}(i, :)$	i-th row of $\mathbf{X}$.
$\mathbf{X}(:, j)$	j-th column of $\mathbf{X}$.
$\mathbf{X}(\mathcal{I}, \mathcal{J})$	Submatrix of $\mathbf{X}$ formed by the rows indexed by $\mathcal{I}$ and the columns indexed by $\mathcal{J}$.
$[\mathbf{A}, \mathbf{B}]$	Horizontal concatenation of matrices $\mathbf{A}$ and $\mathbf{B}$ with compatible row dimensions.
$\mathbf{0}_n$	All-zero column vector of length n.
$\mathbf{1}_n$	All-one column vector of length n.
$\mathbf{I}_n$	Identity matrix of dimension $n \times n$.
$\|\mathbf{x}\|_0$	Number of non-zero entries of the vector $\mathbf{x}$.
$\|\mathbf{x}\|_2$	Euclidean norm of the vector $\mathbf{x}$.
$\langle \mathbf{A}, \mathbf{B} \rangle_{\mathrm{F}}$	Frobenius inner product of two matrices $\mathbf{A}$ and $\mathbf{B}$.

Chapter 1

Introduction

Recent predictions indicate that global network traffic will continue to rise dramatically over the coming years [1], [2], [3], reinforcing the need to process data closer to where it is generated rather than relying solely on centralized computation. Furthermore, in many large-scale systems, transferring raw datasets or reconstructing them at a single location is prohibitively expensive. This requires the efficient allocation of computational tasks across multiple distributed nodes, rather than being centrally stored and processed.

The rapid growth of data-intensive applications has fundamentally reshaped the architecture of modern computing systems. In such systems, multiple servers or devices must collaborate to compute functions of decentralized datasets, often under stringent constraints on storage, communication, computation [4], latency [5], [6], and reliability [7].

Distributed computing systems are therefore increasingly shaped by two fundamental limitations:

1. The cost of data transmission across the communication network,
2. The limited computational capability of each distributed node.

These challenges have become more prominent with the explosive growth of data traffic generated by numerous modern applications, e.g., large language models (LLMs) [8], [9], mixture of experts [10], over-the-air function computation [11], [12], [13], [14], federated learning [15], [16], privacy-preserving, secure, and robust machine learning (ML) [17], [18], [19], [20], [21], ML-based data-driven modeling and prediction in complex physical systems [22], cloud services [23], [24], edge computing [25], [26] and internet-of-things [27], compressed sensing [28], [29], distributed learning in wireless networks [30], and coded computing in vehicular networks [31]. These aspects are closely related to broader questions of decision and control under resource and demand uncertainty in large-scale networked systems [32].

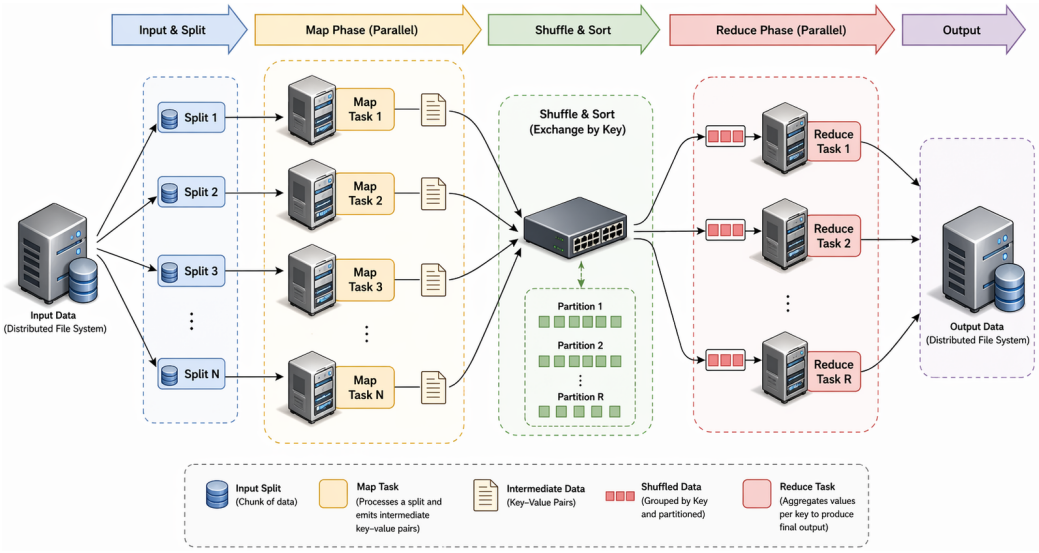

Figure 1.1: MapReduce architecture in a distributed computing cluster [33]. Input data is partitioned into splits, processed in parallel by mapper nodes, reorganized through a shuffle stage, and finally aggregated by reducer nodes to generate the corresponding output.

To address these challenges, several distributed computing frameworks have been developed [33], [34], [35], [36], [37], [38], [39]. Among them, MapReduce [33] remains one of the most widely adopted paradigms due to its simple abstraction, scalability, and foundational implication in many modern distributed computing systems, e.g., [40], [41], [42], [43]. As illustrated in Figure 1.1 [1], it structures computation into three main stages:

1. *Map*, where nodes process their local data and generate intermediate outputs;

2. *Shuffle*, where these intermediate results are exchanged across nodes;

3. *Reduce*, where the final outputs are aggregated.

This decomposition not only enables scalable and fault-tolerant computation but also highlights the central role of communication in distributed computing over communication networks.

While existing distributed architectures, including MapReduce [33], Apache Hadoop [34], [35], TeraSort [36], [37], AlphaSort [38], and Apache Spark [39], provide powerful and scalable solutions, they also bring to the forefront a

[1]The figure was created by the author using ChatGPT (OpenAI, GPT-5.5).

fundamental challenge. At the heart of these systems lies a critical question: How should data be encoded and decoded across distributed nodes to balance *computation* and *communication* in an efficient and principled manner?

Information theory has traditionally focused on the reliable transmission and reconstruction of data [44], [45]. In many modern systems, however, the goal is not to recover the entire dataset but rather to compute a task-relevant function of it. To that end, various approaches have been proposed from both information-theoretic and coding-theoretic viewpoints to exploit computational structure and reduce communication overhead in distributed systems, which are described as follows.

Körner introduced the notion of *graph entropy*, using graph-theoretic representations of source ambiguity to characterize the minimum communication rate required for zero-error compression [46]. This framework established a fundamental connection between coding, graph coloring, and entropy, and later inspired the use of *characteristic graphs* in distributed function computation and functional compression with side information, e.g., [47], [48], [49]. Information-theoretic studies also established fundamental limits for distributed function computation from a source coding perspective for special classes of functions by exploiting their structure. In particular, Körner and Marton demonstrated that computing certain functions, such as the modulo-two sum of correlated sources, can require substantially lower communication rates than reconstructing the underlying sources [50]. Later, Han and Kobayashi further elaborated the relationship between function computation and source reproduction by identifying conditions under which computing a function may be strictly easier than recovering the sources themselves [51]. Other structured coding frameworks exploit algebraic structure and source dependencies to improve communication efficiency in distributed computation [52], [53]. For example, index coding [54], [55], [56] leverages side information and lattice-based coding techniques [57], [58], [59] benefits from algebraic structures to reduce communication load. Moreover, network coding [60], [61] further exploits network interactions across network nodes and statistical correlations across distributed data sources to exploit the fundamental limits of compression for function computation. More details about this research direction can be found in [62].

In another line of research, coding-theoretic approaches have been utilized for computing functions to mitigate communication bottlenecks, improve resilience, and exploit coding advantages across servers. In coded distributed computing (CDC) [63], [64], [65], [66], [67], [68], [69], coding schemes are created through carefully designed data placement and computation assignment across servers, enabling coded multicasting during the shuffle phase and thereby reducing communication load. Similarly, Lagrange coded computing

(LCC) [70], [71] introduces structured redundancy into the computation itself by encoding data through polynomial-based constructions, providing resilience against straggling workers while also enabling, in certain settings, privacy guarantees. More recently, function-correcting codes (FCCs) [72], [73] have been introduced to safeguard computing a function of a message against computational errors by exploiting the structure of the function itself, providing a function-aware framework by utilizing error-correcting codes (ECCs) for reliable function computation, similarly as in [74], [75], [76]. Additionally, sketching-based methods and task-oriented compression and quantization techniques [77], [78], [79], [80], [81], [82], [83], [84], [85], [86] enable compressed task-aware representations for approximate distributed computing or learning, which reduce communication costs while allowing controlled lossy reconstruction of the demanded functions.

A major portion of the existing literature on CDC has focused on settings in which the demanded functions are linearly-separable or have algebraically convenient structures. Representative examples include gradient aggregation and gradient coding [87], [88], [89], [90], convolutional coding [91], [92], [93], and matrix multiplication [94], [95], [96], [97], [98]. Although these models have led to elegant coding constructions, many practically relevant computations are not linearly-decomposable. Non-linear functions, multivariate polynomial evaluations, and structured distributed processing may involve interactions among the inputs that cannot be properly captured through simple additive decompositions. In such settings, data placement, functional structure, system heterogeneity, and the joint design of computation and communication become particularly important. This thesis is motivated by the need to study such computation models and to develop coding and representation techniques that enable scalable distributed computing under stringent communication and computation constraints.

This dissertation studies the communication–computation tradeoff in distributed computing through three complementary approaches, each corresponding to a distinct modeling viewpoint and application regime. The first approach investigates the role of computation placement in distributed computation of Boolean functions, focusing on how function sensitivity and data allocation affect communication costs. The second approach studies non-linearly separable distributed computation through a structured tensor-based framework, exploiting computation structure in distributed system design under communication and computation constraints. The third approach considers coded distributed matrix multiplication (DMM) in the presence of straggling workers, developing a structured coding strategy that combines source-side transformations with polynomial-based redundancies to enable recovery from a sufficiently large subset of worker responses. Together, these

three directions provide the foundations for the subsequent chapters of this thesis.

We next present an overview of the first proposed approach.

1.1 Sensitivity-based Approach

The first technical part of the thesis in Chapter 2 adopts a *sensitivity-based perspective* and studies the influence of data placement on the distributed computation of Boolean functions, which is based on the author's publications in [99], [100]. In this setting, the key observation is that communication cost is not determined solely by the number of datasets stored at each server, but also by how strongly the demanded Boolean function depends on each input dataset and how these influential datasets are distributed across the servers. By exploiting the notions of sensitivity and influence of Boolean functions, this work reveals that placement is not merely a storage decision; it is also an intrinsic component of the communication design itself. The resulting framework identifies placement strategies that minimize communication cost and clarifies how leveraging the function structure can help to jointly optimize placement and communication cost.

In particular, in this chapter, we explore a distributed setting, where a user seeks to compute a linearly-separable Boolean function of degree M from N servers, each with a cache size M. Exploiting the fundamental concepts of sensitivity and influences of Boolean functions, we devise a novel approach to capture the interplay between dataset placement across servers and server transmissions and to determine the optimal solution for dataset placement that minimizes the communication cost.

In particular, for the partitioned placement setting considered in Chapter 2, we showcase the achievability of the minimum average joint sensitivity [2], which is $\frac{N}{2^{M-1}}$, as a measure for the communication cost.

1.2 Sparse Tensor Factorization Approach

The second part of the thesis in Chapter 3 develops a *tensor-theoretic framework* for distributed computation of non-linearly separable functions, which is based on the author's publications in [101], [102], [103]. Here, the objective is to move beyond linearly-separable functions and matrix-based abstractions [104], [105] and to capture the higher-order structure of multi-user poly-

[2]Average joint sensitivity quantifies how, on average, different combinatorial groups of datasets affect the function outcome, as detailed in Chapter 2.

nomial computations naturally based on the structure of the users' demands. Particularly, this chapter considers the N-server distributed computing setting with K users requesting functions that are arbitrary multi-variable polynomial evaluations of L real (potentially non-linear) basis subfunctions, where each function output is raised to a bounded power. Our aim is to seek efficient task allocation and data communication techniques that reduce computation and communication costs. Through this process, tensor structure becomes a natural tool for representing non-linearly separable functions and for deriving explicit achievability constructions under computation and communication constraints. Towards this, we take a tensor-theoretic approach, in which we represent the requested non-linearly decomposable functions using a properly designed tensor $\bar{\mathcal{F}}$, whose sparse decomposition into a tensor $\bar{\mathcal{E}}$ and matrix $\mathbf{D}$ directly defines the computational task assignment, connectivity, and communication patterns.

In Chapter 3, the designed *lossless* achievable construction integrates fixed-support singular value decomposition (SVD)-based tensor factorization with multi-dimensional tiling of $\bar{\mathcal{E}}$ and $\mathbf{D}$, and a bipartite graph matching-based recursive assignment of tiles. This step transforms an overlapping decomposition into a disjoint one and reduces the resulting sum-rank of tiles, thereby decreasing the number of required servers. Under mild dimensionality conditions, we derive an explicit zero-error characterization of the achievable system rate K/N_{ach}. Numerical simulations demonstrate the computational and communication savings over existing state-of-the-art matrix factorization approaches across a wide range of system parameters.

1.3 Straggler-Resilient Coded Distributed Matrix Multiplication

The third part of the thesis, presented in Chapter 4, focuses on *straggler-resilient coded DMM*, which is based on the author's publications in [106], [107], [108]. In distributed computing systems, differences in worker computation times and communication delays may cause some stragglers to dominate the overall completion time. This setting motivates a hybrid construction that combines prior structured source coding techniques for distributed matrix computation, particularly Körner–Marton-style coding and its application to structured matrix computation [50], [109], [110], with polynomial-based CDC techniques for straggler-resilient matrix multiplication [111]. The resulting framework illustrates how source-side structure and worker-side computation redundancy can be jointly incorporated into the design of a DMM scheme.

More specifically, building on these prior structured coding techniques, we consider the *distributed structured PolyDot code* (DSPolyDot code) construction for the distributed computation of the matrix product $\mathbf{A}^\intercal\mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ have entries in the finite field $\mathbb{F}_q$ and are stored at two separate source nodes. The construction applies structured transformations independently at the two sources and combines the resulting descriptions with the PolyDot-code framework in [112]. The workers then compute coded intermediate matrix products, from which the receiver can recover the desired result after collecting a sufficiently large subset of responses. Consequently, the receiver need not wait for the slowest workers. In the asymptotic regime with a large number of workers, the DSPolyDot construction preserves the communication and computation orders of conventional PolyDot codes while providing a structurally different encoding mechanism for straggler-resilient DMM.

1.4 Summary and Thesis Outline

This thesis explores sensitivity-based methods, tensor-theoretic approaches, and structured coded computing in three distributed computation scenarios. Although these three parts address different problems, they are unified by a common principle:
Distributed computation can be made significantly more communication and computation-efficient by exploiting the structure of the demanded functions, such as separability of the desired functions in their arguments (i.e., inputs representing the datasets), non-separability, where capturing the structure allows us to optimize data placement, transmission design, and coded computation strategies, and finally understand the role of task allocation with respect to the nature of the functions in achieving performance guarantees.

This structure is captured through sensitivity and influence in the first part, sparse tensor representations and structured factorization in the second part, and the combined use of source coding and polynomial coding for straggler-resilient DMM in the third part.

Across all three directions, the thesis shows that the performance of distributed systems depends critically on how computation tasks are represented, how datasets are placed, and how coding is adapted to the underlying functional dependencies.

As a summary, the contributions of this dissertation advance the understanding of distributed computing beyond conventional linearly-separable models. They provide new sensitivity-based approaches for analyzing the role of placement in distributed computation, new tensor-based methods for non-linearly separable distributed computing, and new hybrid coding designs

for cooperative matrix multiplication. Through these contributions, this thesis develops new approaches to distributed computation by drawing on tools from information theory, coding theory, combinatorics, and high-dimensional representations, and highlights several new directions for efficient non-linear function computation over communication networks.

The rest of this thesis is organized as follows.

In Chapter 2, we investigate the role of dataset placement in the distributed computation of Boolean functions. Leveraging a sensitivity-based perspective, we analyze how the influence of input datasets on the demanded function affects both computation and communication costs. We derive fundamental bounds and propose efficient placement strategies from a communication perspective that prioritize highly influential datasets, thereby reducing redundancy and improving overall system efficiency.

Chapter 3 addresses the problem of non-linearly separable distributed computing. We introduce a tensor-based formulation that captures the multi-dimensional structure of user demands and develop an achievable scheme based on sparse tensor factorization and structured multi-dimensional tiling, which reduces the number of required servers. To further reduce the number of required servers and eliminate redundant computations induced by overlapping supports, we incorporate combinatorial assignment techniques for disjoint allocation of the computational tasks to the servers. We also provide theoretical bounds and demonstrate the gains of the proposed approach over conventional matrix factorization-based methods in [105]. Chapter 3 is developed in more detail than the other technical chapters because it contains the main mathematical framework of the thesis, including the tensor formulation, two achievable constructions, the combinatorial tile-assignment algorithm, and the extension to the multi-shot setting.

In Chapter 4, we study DMM in the presence of straggling workers, where some worker nodes may respond slowly or fail to return their assigned computations to the receiver. Building on prior structured source coding techniques for distributed matrix computation, particularly Körner–Marton-style coding and its application to structured matrix computation [50], [109], [110], we combine these techniques with polynomial-based coded computation for straggler-resilient DMM. The resulting DSPolyDot construction in [108] enables the receiver to recover the desired matrix product from a sufficiently large subset of worker responses, thereby reducing the need to wait for slow or unavailable workers. It also preserves the asymptotic orders of the communication and computation requirements relative to the PolyDot-code construction in [112].

Finally, Chapter 5 concludes the thesis by summarizing the main findings and discussing several promising directions for future research.

Relation to author's prior publications. The technical chapters of this dissertation are based on research works carried out during the PhD and on manuscripts that have been published or are currently being prepared for submission in collaboration with co-authors. Chapter 2 is mainly based on the conference paper in [100], Chapter 3 is mainly based on the conference paper in [102] and the extended manuscript in [103], and Chapter 4 is mainly based on the conference paper in [108]. The presentation in this dissertation has been revised and adapted to provide a unified thesis narrative, consistent notation, detailed explanations of the system models, and additional clarifications of the assumptions, proofs, and limitations. Some technical statements, theorem formulations, proofs, and figures are reused or adapted from these publications where appropriate, with citations provided within the corresponding chapters.

Chapter 2

Sensitivity-based Approach

In this chapter, we introduce a novel perspective for analyzing distributed computing systems based on the sensitivity of functions to their underlying datasets. Rather than treating all input data identically, this approach quantifies how variations in individual datasets influence the overall computation, thereby capturing the relative importance of different input datasets. Building on this sensitivity-based viewpoint, we develop a framework that links the influence of datasets to the allocation of computational tasks and the resulting communication overhead.

This perspective reveals that the placement of highly influential datasets plays a decisive role in determining system efficiency. By prioritizing such datasets in the design of computation and communication strategies, one can significantly reduce redundant operations and unnecessary data exchanges. The proposed analysis thus provides both fundamental insights and practical guidelines for optimizing distributed computing systems under resource constraints.

Relation to the author's prior publications. This chapter is mainly based on [99], [100]. The presentation has been substantially revised and expanded for this thesis, with unified notation, a reformulated system model, additional explanations, and updated conclusions. Unless otherwise indicated, the technical statements, theorem formulations, proofs, examples, and figures in this chapter are adapted from [100].

2.1 Introduction

Recent technological developments have made distributed computing increasingly important for handling computationally intensive tasks. Instead of

processing an entire workload at a single node, the task is partitioned into smaller components and assigned to multiple workers connected through a network. This paradigm appears in applications such as distributed ML over server networks [113] and large-scale cloud computing platforms [42]. Practical frameworks such as MapReduce [33] have demonstrated the effectiveness of distributed processing. However, supporting continuously growing computational demands requires a more fundamental understanding of how data should be placed, compressed, and communicated across the network to enable efficient computation. These issues form the main focus of this chapter.

2.1.1 Related Work

We begin by reviewing the literature on functional compression, followed by existing methods for distributing datasets across workers to support different computational tasks.

From source compression to functional compression The fundamental limits of conventional data compression are well studied in both centralized [44] and distributed settings [45]. Functional compression, however, is concerned with recovering a desired function of the sources rather than reconstructing the sources themselves. Its analysis therefore requires techniques that account for the specific structure of the target computation. One such approach is based on graph entropy, introduced by Körner to distinguish source symbols according to whether they lead to different function values [46], [114], [115]. Graph coloring techniques have subsequently been applied to several distributed functional compression problems, including those studied in [48], [49], [109], [116], [117]. Nevertheless, graph-based formulations are not readily applicable to arbitrary functions. An alternative line of work exploits the algebraic structure of particular functions through specially designed coding schemes [50], [51]. Since these constructions generally depend on the function being computed, a different encoder may be required for each task, which can limit their practical flexibility.

Graph-based functional compression and the sensitivity-based formulation considered in this chapter address complementary aspects of function-aware distributed computation. Characteristic-graph approaches distinguish source realizations according to whether they must remain distinguishable for the computation of a specific function, with the objective of reducing the required source coding rate. In contrast, the present chapter focuses on *dataset placement*: the datasets are distributed across servers with limited storage, and the objective is to understand how the placement of the Boolean function's

inputs affects the communication required for exact computation. Sensitivity and influence are therefore used here to characterize the dependence of the demanded function on its inputs and to guide the placement phase, rather than as part of a graph-based construction. We discuss graph-based functional compression to position the proposed approach among function-aware communication methods while highlighting this difference in the underlying design objective.

On the other hand, building on the CDC framework of [118], several studies have examined the interplay among storage, computation, and communication. These include cyclic placement strategies for linearly-separable functions [119], [120], secure computation of Boolean functions [65], linear coding approaches for jointly optimizing data placement and transmission [104], placement delivery arrays [4], and tessellation-based constructions [105]. Although neural network-based methods have been used to estimate communication rate regions in certain settings, as in [121], characterizing the achievable rate region for general distributed computing systems remains largely unresolved.

The role of placement in distributed computation A widely used strategy for assigning datasets to distributed servers is cyclic placement, as considered, for example, in [119]. Under this scheme, the data assigned to each server are obtained by applying a fixed circular shift to the placement at the preceding server. This structured replication ensures that an appropriately chosen subset of servers collectively stores all datasets required to evaluate the functions requested by the user.

Although most existing studies focus on the encoding, transmission, and decoding stages under a fixed data assignment, the manner in which datasets are placed across servers can itself have a substantial impact on the resulting communication cost [104], [105].

2.1.2 Motivation and Contributions

Motivated by the role of dataset placement in determining the communication cost of distributed computing systems [104], [105], we employ the notions of sensitivity and influence of Boolean functions [122], [123] to identify placement configurations that minimize the required communication. Our analysis focuses specifically on linearly-separable Boolean functions.

In this chapter, we develop a distributed computing framework consisting of a master node, multiple servers, and a user seeking the exact evaluation of a linearly-separable Boolean function. The master node assigns the datasets to the servers, which then carry out local computations on their stored data. The proposed framework accounts for the joint influence of different subsets

of datasets on the requested function under an arbitrary number of servers with equal cache capacities. This characterization allows us to establish the relationship between data placement and communication cost and to determine placement configurations that exploit the structure of the target Boolean function.

2.1.3 Organization

The remainder of this chapter is structured as follows. Section 2.2 introduces the distributed Boolean computation model and explains how dataset placement affects the required transmissions. Section 2.4 then develops an influence-based method for evaluating the communication cost of linearly-separable Boolean functions under different placement configurations. Finally, Section 2.5 outlines possible directions for extending this framework to broader classes of functions.

Notation. We denote by $\mathbf{W} = (W_1, \ldots, W_K) \in \mathbb{F}_2^K$ the random vector collecting the K binary datasets. For each server n, the set $\mathcal{S}_n \subseteq \mathbf{W}$ contains the datasets assigned to that server. The k-th standard basis vector is denoted by $\mathbf{e}_k \triangleq (0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{F}_2^K$, where its k-th entry equals 1 and all other entries equal 0. The symbols $\oplus$ and $\bigoplus$ denote addition and summation over $\mathbb{F}_2$, respectively. Hence, $\mathbf{W} \oplus \mathbf{e}_k$ denotes the binary vector obtained from $\mathbf{W}$ by flipping its k-th entry.

We next detail the system model and key system parameters.

2.2 System Model

We consider a distributed computing system composed of a master node, N distributed servers indexed by $[N]$, and a user. The system contains K independent and identically distributed datasets,

$$W_k \sim \text{Bern}\left(\frac{1}{2}\right), \qquad k \in [K].$$

The master assigns a possibly overlapping subset of these datasets to each server. All servers have the same storage capacity and can store at most M datasets.

The user wishes to compute a Boolean function [1]

$$f : \mathbb{F}_2^K \longrightarrow \mathbb{F}_2$$

[1] A function is Boolean if both its domain and codomain are binary [123].

of the complete dataset vector

$$\mathbf{W} = (W_1, W_2, \ldots, W_K).$$

Throughout this chapter, we represent a general Boolean function in its multilinear polynomial form over $\mathbb{F}_2$ [123]:

$$f(\mathbf{W}) = \bigoplus_{\mathcal{P}\subseteq[K]} c_{\mathcal{P}} \prod_{k\in\mathcal{P}} W_k, \tag{2.1}$$

where $c_{\mathcal{P}} \in \mathbb{F}_2$ is the coefficient associated with the subset $\mathcal{P} \subseteq [K]$. For $\mathcal{P} = \emptyset$, we adopt the convention

$$\prod_{k\in\emptyset} W_k = 1,$$

so that $c_\emptyset$ represents the constant term.

Throughout this chapter, the demanded Boolean function is assumed to be known to the master node and to the servers before the encoding phase. This allows the servers to design their local subcomputations in a function-aware manner. The placement is uncoded, datasets are binary and independent unless otherwise stated, and the user is required to recover the demanded Boolean function with zero error. The objective is not to characterize the communication-computation trade-off for arbitrary Boolean functions, but rather to investigate the role of placement through sensitivity and influence for the considered class of linearly-separable Boolean functions.

2.2.1 Distributed Computing Phases

The distributed computation of $f(\mathbf{W})$ proceeds in three phases: dataset placement, server encoding and transmission, and user decoding.

Dataset Placement During the placement phase, the master assigns a subset of the datasets to each server without coding across different datasets. This strategy is commonly referred to as *uncoded placement*; see, for example, [124]. For each server $n \in [N]$, let

$$\mathcal{S}_n \subseteq \mathbf{W}\,, \qquad |\mathcal{S}_n| = M\,,$$

denote the datasets assigned to that server. The corresponding placement function is

$$\rho_n : \mathbb{F}_2^K \longrightarrow \mathbb{F}_2^M\,, \qquad n \in [N]\,, \tag{2.2}$$

with

$$\mathcal{S}_n \triangleq \rho_n(\mathbf{W}) = \Big(W_k : W_k \in \mathcal{S}_n\Big). \tag{2.3}$$

Therefore, server n stores the M datasets in $\mathcal{S}_n$. The assigned subsets may overlap across different servers. We denote the overall placement configuration by

$$\mathcal{S} \triangleq \{\mathcal{S}_n : n \in [N]\}.$$

Encoding and Transmission After the placement phase, each server computes a message using its locally stored datasets and the requested Boolean function f. We assume that all servers know the computation task in advance and can design their local subcomputations accordingly.

For server $n \in [N]$, the encoding operation is represented by

$$E_n^f : \mathbb{F}_2^M \longrightarrow \mathbb{F}_2^{|Z_n|}, \tag{2.4}$$

where

$$Z_n \triangleq E_n^f(\mathcal{S}_n) = \Big(Z_{n,1}, \ldots, Z_{n,|Z_n|}\Big) \tag{2.5}$$

denotes the message transmitted by server n to the user. The collection of all server transmissions is denoted by

$$Z \triangleq (Z_1, \ldots, Z_N), \tag{2.6}$$

whose total length is

$$|Z| \triangleq \sum_{n\in[N]} |Z_n|. \tag{2.7}$$

The user combines these transmissions to recover the requested function value.

Decoding Once the user has received the messages from all servers, it applies a decoding function designed according to the placement configuration and the server encoding functions. The decoder is represented by

$$D : \mathbb{F}_2^{|Z|} \longrightarrow \mathbb{F}_2. \tag{2.8}$$

We require lossless recovery of the requested Boolean function for every input vector $\mathbf{W} \in \mathbb{F}_2^K$. Therefore, a valid placement, encoding, and decoding scheme must satisfy

$$D(Z) = D\Big(E_1^f(\mathcal{S}_1), \ldots, E_N^f(\mathcal{S}_N)\Big) = f(\mathbf{W}), \qquad \forall\, \mathbf{W} \in \mathbb{F}_2^K. \tag{2.9}$$

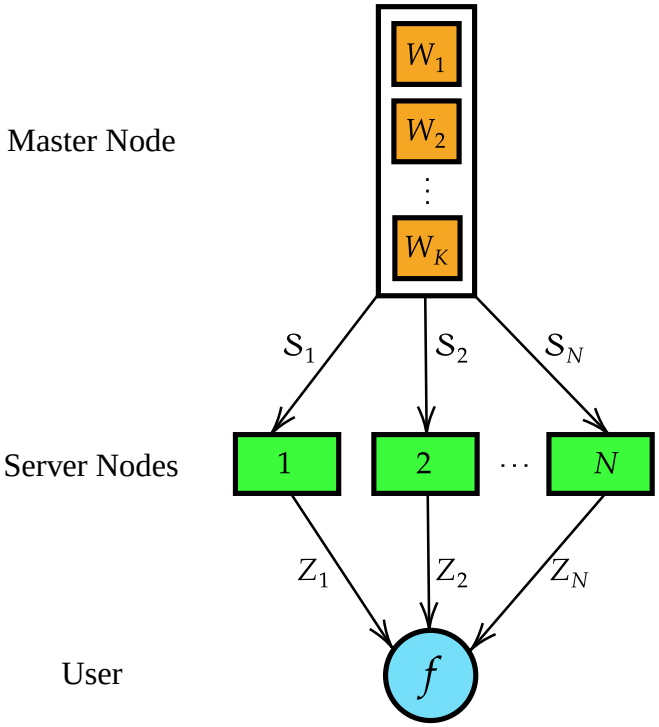

Figure 2.1: A generic distributed computing system model.

We illustrate the distributed computing model in Figure 2.1. In this chapter, we focus on linearly-separable Boolean functions whose component functions are monomials of degree d. Specifically, we consider

$$f(\mathbf{W}) = \bigoplus_{n=1}^{N} f_n(\mathbf{W}), \qquad f_n(\mathbf{W}) = \prod_{k \in \mathcal{P}_n} W_k, \tag{2.10}$$

where

$$\mathcal{P}_n \in \mathcal{K}_d, \qquad n \in [N],$$

and

$$\mathcal{K}_d \triangleq \{\mathcal{P} \subseteq [K] : |\mathcal{P}| = d\}. \tag{2.11}$$

denotes the collection of all d-element subsets of $[K]$.

For the main analytical result in Section 2.4, we specialize to the homogeneous setting $d = M$ and $K = NM$. However, the illustrative example presented next is not restricted to these assumptions. Furthermore, for the main analytical result in Section 2.4, we further assume that the supports of

the component functions form a partition of the dataset indices, i.e.,

$$\mathcal{P}_n \cap \mathcal{P}_{n'} = \emptyset, \qquad n \neq n', \tag{2.12}$$

$$\bigcup_{n\in[N]} \mathcal{P}_n = [K], \tag{2.13}$$

$$|\mathcal{P}_n| = M, \qquad n \in [N]. \tag{2.14}$$

We also restrict the admissible placement configurations in the main analytical result to disjoint partitions of the datasets:

$$\mathcal{S}_n \cap \mathcal{S}_{n'} = \emptyset, \qquad n \neq n', \tag{2.15}$$

$$\bigcup_{n\in[N]} \mathcal{S}_n = \{W_1, \ldots, W_K\}, \tag{2.16}$$

$$|\mathcal{S}_n| = M, \qquad n \in [N]. \tag{2.17}$$

Thus, every dataset is assigned to exactly one server in the main analytical setting. However, the illustrative example in Subsection 2.2.2 is not restricted to this partitioned placement model.

We refer to the resulting model as a $(K, N, M, \mathcal{S}, f)$ distributed computing scheme, where $\mathcal{S} = \{\mathcal{S}_n : n \in [N]\}$ denotes the placement configuration. We next define achievability for lossless distributed computation of $f(\mathbf{W})$.

Definition 1 (Achievable distributed computing scheme)**.** *A $(K, N, M, \mathcal{S}, f)$ distributed computing scheme is said to be achievable if, under the placement configuration $\mathcal{S}$, there exist encoding functions E_n^f, $n \in [N]$, and a decoding function D such that the user recovers $f(\mathbf{W})$ without error for every $\mathbf{W} \in \mathbb{F}_2^K$. Equivalently,*

$$D\Big(Z_1, \ldots, Z_N\Big) = f(\mathbf{W}), \qquad \forall\, \mathbf{W} \in \mathbb{F}_2^K, \tag{2.18}$$

where

$$Z_n = E_n^f(\mathcal{S}_n), \qquad n \in [N],$$

is the function-aware and placement-dependent message transmitted by server n. The encoding and decoding operations may be non-linear in general.

Let $Z = (Z_1, \ldots, Z_N)$ denote the collection of all server transmissions. We define the total number of transmitted symbols under placement configuration $\mathcal{S}$ as

$$\tau_{\mathcal{S}} \triangleq |Z| = \sum_{n=1}^{N} |Z_n|. \tag{2.19}$$

We next present an example that illustrates how the placement configuration $\mathcal{S}$ affects $\tau_{\mathcal{S}}$ for a given $(K, N, M, \mathcal{S}, f)$ distributed computing scheme. In Section 2.4, we then establish the relationship between $\tau_{\mathcal{S}}$ and the communication cost measure used throughout this chapter.

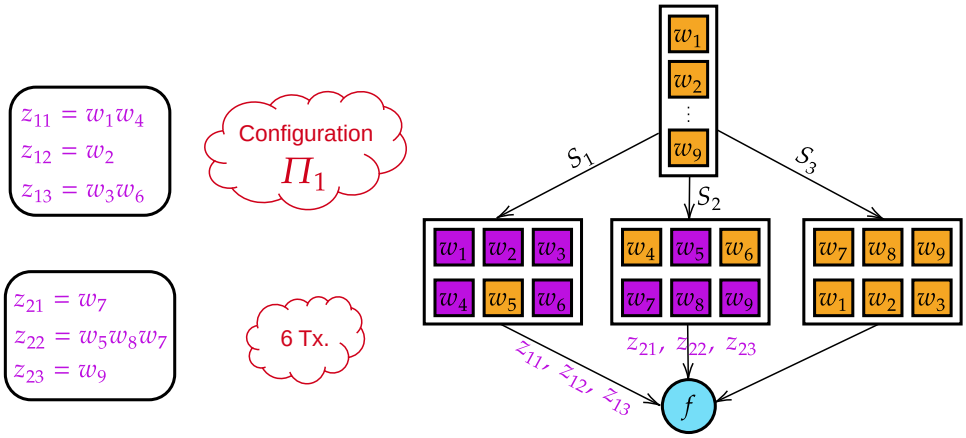

Figure 2.2: Configuration $\mathcal{S}^{(1)}$ (Π_1 in the figure): cyclic. The violet colored boxes represent the assigned datasets to the servers.

2.2.2 The Interplay between Placement and Communication Cost

In this subsection, we first present an example, with two different placement configurations, namely $\mathcal{S}^{(1)}$, which is cyclic, and $\mathcal{S}^{(2)}$, for computing a Boolean function. We next contrast the total number of transmissions needed in each configuration to demonstrate the role of placement and transmissions.

Example 1. *Consider a distributed computing system with 9 datasets, 3 servers, each with cache size 6, for evaluating*

$$f(\mathbf{W}) = W_1W_4W_7 \oplus W_2W_5W_8W_7 \oplus W_3W_6W_9 \tag{2.20}$$

with different placement configurations as follows.

(i) **Configuration $\mathcal{S}^{(1)}$:**
The dataset assignment is cyclic such that

$$\mathcal{S}_1^{(1)} = \{W_1, W_2, W_3, W_4, W_5, W_6\},$$
$$\mathcal{S}_2^{(1)} = \{W_4, W_5, W_6, W_7, W_8, W_9\},$$
$$\mathcal{S}_3^{(1)} = \{W_1, W_2, W_3, W_7, W_8, W_9\},$$

satisfying the cache size constraint with equality. To successfully compute (2.20) in this configuration, it suffices that the servers need to compute and transmit data as shown in Table 2.1 and Figure 2.2. Hence, this scenario requires $\tau_{\mathcal{S}^{(1)}} = 6$ transmissions in total.

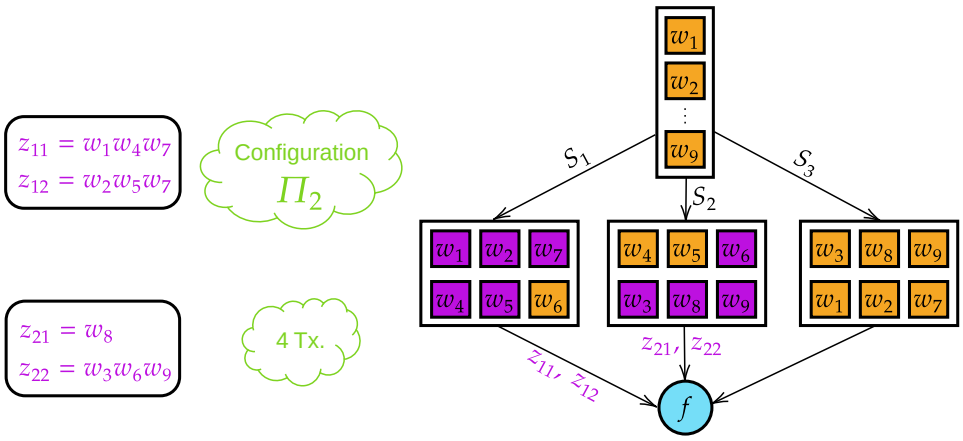

Figure 2.3: Configuration $\mathcal{S}^{(2)}$ (Π_2 in the figure): acyclic. The violet colored boxes indicate the assigned datasets to the servers.

Assigned subsets	Transmitted data
$\mathcal{S}_1^{(1)}$	$Z_{11}^{(1)} = W_1W_4,\ Z_{12}^{(1)} = W_3W_6,\ Z_{13}^{(1)} = W_2$
$\mathcal{S}_2^{(1)}$	$Z_{21}^{(1)} = W_5W_8W_7,\ Z_{22}^{(1)} = W_7,\ Z_{23}^{(1)} = W_9$
$\mathcal{S}_3^{(1)}$	No transmissions

Table 2.1: Server-transmission details for $\mathcal{S}^{(1)}$.

(ii) **Configuration $\mathcal{S}^{(2)}$:**
The dataset assignment satisfies

$$\mathcal{S}_1^{(2)} = \{W_1, W_2, W_4, W_5, W_6, W_7\}\,,$$
$$\mathcal{S}_2^{(2)} = \{W_3, W_4, W_5, W_6, W_8, W_9\}\,,$$
$$\mathcal{S}_3^{(2)} = \{W_1, W_2, W_3, W_7, W_8, W_9\}\,.$$

To successfully compute (2.20) in this setting, without the presence of stragglers, a viable transmission scheme is shown in Table 2.2 and Figure 2.3. Thanks to a better arrangement of the datasets that is sensitive to the function in $\mathcal{S}^{(2)}$ versus $\mathcal{S}^{(1)}$, the total number of transmissions is

$$\tau_{\mathcal{S}^{(2)}} = 4 < \tau_{\mathcal{S}^{(1)}} = 6\,.$$

Assigned subsets	Transmitted data
$\mathcal{S}_1^{(2)}$	$Z_{11}^{(2)} = W_1W_4W_7$, $Z_{12}^{(2)} = W_2W_5W_7$
$\mathcal{S}_2^{(2)}$	$Z_{21}^{(2)} = W_3W_6W_9$, $Z_{22}^{(2)} = W_8$
$\mathcal{S}_3^{(2)}$	No transmissions

Table 2.2: Server-transmission details for $\mathcal{S}^{(2)}$.

Example 1 is intended solely to illustrate how dataset placement affects the required number of transmissions, particularly under a cyclic placement with replicated dataset assignments. It is not restricted to the homogeneous setting $d = M$ and $K = NM$, which is utilized later for the main analytical result presented in this chapter.

We infer from Example 1 how a cleverly conducted placement phase in distributed computing settings could dramatically reduce the total number of transmissions needed for error-free recovery of the Boolean function at the user-side.

To evaluate the communication cost of the proposed $(K, N, M, \mathcal{S}, f)$ distributed computing scheme, we will next detail a novel approach that relies on the average joint sensitivity of the computation task abstracted by the Boolean function.

2.3 A Primer on Sensitivity and Influences

Definition 2. (Sensitivity [123]). *The sensitivity of a Boolean function $f(\mathbf{W})$ for a fixed input $\mathbf{W} \in \mathbb{F}_2^K$ with the k^{th} entry of $\mathbf{W}$ being flipped is given by*

$$S_k(f, \mathbf{W}) = \mathbb{1}_{f(\mathbf{W} \oplus e_k) \neq f(\mathbf{W})} \ . \tag{2.21}$$

The sensitivity of f on input $\mathbf{W}$ is then expressed as

$$S(f, \mathbf{W}) = \sum_{k=1}^{K} S_k(f, \mathbf{W}) = |\{k : f(\mathbf{W} \oplus e_k) \neq f(\mathbf{W})\}| \ . \tag{2.22}$$

Here we note that the sensitivity of $f(\mathbf{W})$ to the set $\mathbf{W}$ is the number of input variables in $\mathbf{W}$ such that a flip in a given input entry W_k, $k \in [K]$ by keeping $\mathbf{W} \backslash W_k$ fixed causes a flip in the outcome or value of f. Hence, sensitivity is a measure of the dependence of the value of f on individual entries of $\mathbf{W}$.

Definition 3. (Influence [123].) *The influence of the k^{th} variable on Boolean function $f(\mathbf{W})$ over uniformly distributed input vectors $\mathbf{W} \in \{0,1\}^K$ is expressed as*

$$\mathrm{Inf}_k(f) = \mathbb{P}[f(\mathbf{W} \oplus e_k) \neq f(\mathbf{W})] = \mathbb{E}_{\mathbf{W}}[S_k(f, \mathbf{W})] \ . \tag{2.23}$$

We note that $\mathrm{Inf}_k(f)$ captures the sensitivity of $f(\mathbf{W})$ on the number of input vectors $\mathbf{W}$ via flipping a predetermined index k. A high value of $\mathrm{Inf}_k(f)$ means that the value of f depends on the k^{th} variable on many distinct input vectors.

Theorem 1. (An information-theoretic interpretation of influence [125, Theorem 4].) *For any Boolean function f and for any input vector $\mathbf{W}$, composed of independent random variables $W_k,\ k \in [K]$, we have:*

$$\mathrm{Inf}_k(f) = \frac{H(f(\mathbf{W}) \,|\, \mathbf{W} \backslash W_k)}{H(W_k)} \ , \tag{2.24}$$

demonstrating that $\mathrm{Inf}_k(f)$ equals the amount of information needed to determine f given all the datasets except W_k, normalized by the measure of uncertainty of W_k, i.e., $H(W_k)$.

Definition 4. (Average sensitivity [123].) *The average sensitivity of $f(\mathbf{W})$ is the expected sensitivity of $f(\mathbf{W})$ over a uniform distribution of the input vector $\mathbf{W} \in \{0\ , 1\}^K$:*

$$\mathrm{as}(f) = \mathbb{E}_{\mathbf{W}}[S(f, \mathbf{W})] = \sum_{k=1}^{K} \mathbb{E}_{\mathbf{W}}[S_k(f, \mathbf{W})] = \sum_{k=1}^{K} \mathrm{Inf}_k(f) \ . \tag{2.25}$$

2.4 Main Results

In this section, to determine the role of the placement configuration on the communication cost, we first provide a primer on the sensitivity of a Boolean function on its input, and the influence of a set of input variables on the outcome of the function. We then present our main results.

2.4.1 Joint Sensitivity and Influences

Building on the classical notions of sensitivity, influence, and average sensitivity tailored for capturing the sensitivity of a Boolean function by modifying one dataset at each time [122], [123], we will exploit the joint behavior of datasets across servers, as in [126]. A Boolean function $f(\mathbf{W})$ depends on its k^{th} input variable if there exists at least one $\mathbf{W} \in \mathbb{F}_2^K$ such that $f(\mathbf{W} \oplus e_k) \neq f(\mathbf{W})$. To that end, we next define the sensitivity of $f(\mathbf{W})$ on a set of W_k's.

Definition 5. (Joint sensitivity.) *The joint sensitivity of $f(\mathbf{W})$ to the set $\mathcal{S}$ of subsets of datasets is defined as*

$$\mathrm{Sen}_{\mathcal{S}}(f, \mathbf{W}) = \sum_{n=1}^{|\mathcal{S}|} \mathrm{Sen}_{\mathcal{S}_n}(f, \mathbf{W}) \ , \tag{2.26}$$

where $e_{\mathcal{S}_n} \triangleq \bigoplus_{\{k: w_k \in \mathcal{S}_n\}} e_k$ describes the multi-dataset flipped vector, the measure $\mathrm{Sen}_{\mathcal{S}_n}(f, \mathbf{W}) = \mathbb{1}_{f(\mathbf{W} \oplus e_{\mathcal{S}_n}) \neq f(\mathbf{W})}$ captures the sensitivity of $f(\mathbf{W})$ on input $\mathbf{W}$ with the jointly flipped entry datasets with indices k such that $W_k \in \mathcal{S}_n \subseteq \mathcal{S}$.

Definition 6. (Joint influence.) *The joint influence of the datasets of $\mathcal{S}_n$ on the function f is defined as*

$$\mathrm{Inf}_{\mathcal{S}_n}(f) = \mathbb{P}[f(\mathbf{W} \oplus e_{\mathcal{S}_n}) \neq f(\mathbf{W})] = \mathbb{E}_{\mathbf{W}}[\mathrm{Sen}_{\mathcal{S}_n}(f, \mathbf{W})] \ .$$

Definitions 5-6 allow us to introduce our next key metric.

Definition 7. (Average joint sensitivity.) *The average joint sensitivity of $f(\mathbf{W})$ to the set $\mathcal{S} = \{\mathcal{S}_n : n \in [N]\}$ of all possible datasets specified by ρ_n given a cache size constraint M is given as follows:*

$$\mathrm{as}_{\mathcal{S}}(f) = \mathbb{E}_{\mathbf{W}}[\mathrm{Sen}_{\mathcal{S}}(f, \mathbf{W})] = \sum_{n=1}^{|\mathcal{S}|} \mathrm{Inf}_{\mathcal{S}_n}(f) \ . \tag{2.27}$$

Utilizing Definitions 5-7, we can next present our approach for formulating an optimal dataset placement configuration from a communication cost perspective.

2.4.2 The Communication-Optimal Configuration

To evaluate the tradeoff between placement and communication cost for computing a Boolean function f, we first present a Lemma that focuses on each product subfunction f_n given in (2.10) to obtain the joint influence of datasets on f_n.

Lemma 1. (Joint influence on a product subfunction.) *Let $f_n(\mathbf{W}) = \prod_{k \in \mathcal{P}_n} W_k$ be a product subfunction of degree d. For any $\mathcal{S}_n \subseteq \mathbf{W}$ satisfying $\{k : W_k \in \mathcal{S}_n\} \cap \mathcal{P}_n \neq \emptyset$, the joint influence of multiple datasets in a subset $\mathcal{S}_n$ on a product subfunction f_n of degree d equals the influence of each dataset on f_n, i.e., we have*

$$\mathrm{Inf}_{\mathcal{S}_n}(f_n) = \mathrm{Inf}_k(f_n) = \frac{1}{2^{d-1}}, \qquad k \in \mathcal{P}_n.$$

Proof. Let $\mathcal{A} \triangleq \{k : W_k \in \mathcal{S}_n\} \cap \mathcal{P}_n \neq \emptyset$ denote the set of indices of the variables in $\mathcal{S}_n$ that also appear in the product subfunction.

The event $f_n(\mathbf{W} \oplus e_{\mathcal{S}_n}) \neq f_n(\mathbf{W})$ can occur only if flipping the variables indexed by $\mathcal{A}$ changes the value of this product. Since $f_n(\mathbf{W}) = 1$ if and only if all the d variables in the product are equal to one, there are two possible ways in which the value of the product can change after jointly flipping the variables in $\mathcal{A}$. First, the product can change from 1 to 0. This happens only when all variables in $\mathcal{P}_n$ are initially equal to one. In this case, flipping at least one variable in $\mathcal{A}$ changes at least one factor from 1 to 0, and hence the product becomes zero. Second, the product can change from 0 to 1. This happens only when all variables in $\mathcal{A}$ are initially equal to zero and all variables in $\mathcal{P}_n \setminus \mathcal{A}$ are initially equal to one. After flipping the variables in $\mathcal{A}$, all d variables become equal to one, and hence the product becomes one. Consequently, among the 2^d possible assignments of the d variables involved in the product, the only two assignments that lead to a change of the product under the considered joint flip are

$$(1, 1, \ldots, 1) \quad \text{and} \quad \Big(W_k = 0 \text{ for } k \in \mathcal{A},\ W_k = 1 \text{ for } k \in \mathcal{P}_n \setminus \mathcal{A}\Big).$$

Therefore,

$$\mathrm{Inf}_{S_n}(f_n) = \mathbb{P}[f_n(\mathbf{W} \oplus e_{S_n}) \neq f_n(\mathbf{W})] = \frac{2}{2^d} = \frac{1}{2^{d-1}}.$$

Similarly, for an individual variable W_k appearing in f_n, the influence is

$$\mathrm{Inf}_k(f_n) = \mathbb{P}\left[\prod_{i \in \mathcal{P}_n \setminus \{k\}} W_i \neq 0\right] = \left(\frac{1}{2}\right)^{d-1},$$

since all the remaining $d - 1$ variables must be equal to one. Hence,

$$\mathrm{Inf}_{S_n}(f_n) = \mathrm{Inf}_k(f_n) = \frac{1}{2^{d-1}},$$

which proves the claim.

□

Towards determining the joint influence of datasets on f in (2.10), we next present another Lemma that contrasts the joint influence of datasets for the summation of two product subfunctions, namely f_n and $f_{n'}$ where $n' \neq n$, for different dataset placement configurations. To that end, we denote the set of variables included in the subfunction f_n as

$$\mathcal{I}_{f_n} \triangleq \{W_k : k \in \mathcal{P}_n\}.$$

Lemma 2 (Increase in joint influence due to mixing product subfunctions)**.** *Let $f(\mathbf{W}) = \bigoplus_{j=1}^{N} f_j(\mathbf{W})$ be the Boolean function demanded by a user, where the product subfunctions have disjoint supports and degree $d \geq 2$. Consider two distinct subfunctions f_n and $f_{n'}$, where $n \neq n'$, and let*

$$\mathcal{I}_{f_n} \triangleq \{W_k : k \in \mathcal{P}_n\}, \qquad \mathcal{I}_{f_{n'}} \triangleq \{W_k : k \in \mathcal{P}_{n'}\},$$

where f_n and $f_{n'}$ are product subfunctions of degree $d \geq 2$.

Consider the reference subset $\mathcal{S}_1 = \mathcal{I}_{f_n}$, and a mixed subset obtained by replacing one variable of $\mathcal{I}_{f_n}$ by one variable of $\mathcal{I}_{f_{n'}}$, namely

$$\mathcal{S}_1' = \Big(\mathcal{I}_{f_n} \setminus \{W_a\}\Big) \cup \{W_b\}, \qquad W_a \in \mathcal{I}_{f_n},\ W_b \in \mathcal{I}_{f_{n'}}.$$

We therefore have

$$\mathrm{Inf}_{\mathcal{S}_1'}(f) \geq \mathrm{Inf}_{\mathcal{S}_1}(f).$$

More generally, the same inequality holds for any subset obtained from $\mathcal{I}_{f_n}$ by replacing one or more of its variables with variables belonging to other product subfunctions.

Proof. By Lemma 1, grouping the variables that appear in the same product subfunction yields

$$\mathrm{Inf}_{\mathcal{S}_1}(f) = \frac{1}{2^{d-1}}.$$

We now consider the mixed subset $\mathcal{S}_1'$, which differs from $\mathcal{S}_1$ by one variable. Without loss of generality, write

$$\mathcal{S}_1' = \Big(\mathcal{I}_{f_n} \setminus \{W_a\}\Big) \cup \{W_b\}, \qquad W_a \in \mathcal{I}_{f_n},\ W_b \in \mathcal{I}_{f_{n'}}.$$

Let

$$f_j = \prod_{W_k \in \mathcal{I}_{f_j}} W_k, \qquad j \in \{n, n'\}.$$

Then, flipping the variables in $\mathcal{S}_1'$ affects f_n through the $d-1$ variables of $\mathcal{I}_{f_n} \setminus \{W_a\}$, while it affects $f_{n'}$ through the variable W_b.

Consequently,

$$\mathrm{Inf}_{\mathcal{S}_1'}(f) = \mathbb{P}\Big[f(\mathbf{W} \oplus e_{\mathcal{S}_1'}) \neq f(\mathbf{W})\Big].$$

Expanding the two product terms gives

$$\mathrm{Inf}_{\mathcal{S}_1'}(f) = 2\left(\frac{1}{2}\right)^{d-1}\left[1 - \left(\frac{1}{2}\right)^{d-1}\right].$$

Since

$$2\left(\frac{1}{2}\right)^{d-1}\left[1-\left(\frac{1}{2}\right)^{d-1}\right] \geq \left(\frac{1}{2}\right)^{d-1},$$

we obtain

$$\mathrm{Inf}_{\mathcal{S}_1'}(f) \geq \mathrm{Inf}_{\mathcal{S}_1}(f).$$

More generally, consider a subset $\mathcal{S}$ that intersects $t \geq 1$ distinct product subfunctions. Let these subfunctions be indexed by $j_1, \ldots, j_t$, and define

$$\mathcal{A}_i \triangleq \{k : W_k \in \mathcal{S}\} \cap \mathcal{P}_{j_i}, \qquad i \in [t],$$

where $\mathcal{A}_i \neq \emptyset$ for every $i \in [t]$. Since the supports $\{\mathcal{P}_j\}_{j\in[N]}$ are disjoint, flipping the variables in $\mathcal{S}$ affects each product subfunction f_{j_i} only through the variables indexed by $\mathcal{A}_i$.

By Lemma 1, for every $i \in [t]$, the probability that f_{j_i} changes under this joint flip is

$$p \triangleq \frac{1}{2^{d-1}} \tag{2.28}$$

Importantly, this probability does not depend on the cardinality of $\mathcal{A}_i$, provided that $\mathcal{A}_i$ is nonempty. Moreover, because the product supports are disjoint and the datasets are independent, the change events associated with $f_{j_1}, \ldots, f_{j_t}$ are independent.

Since the overall function is the XOR of the component product subfunctions, its value changes if and only if an odd number of these t product subfunctions change. Therefore,

$$\begin{aligned}\mathrm{Inf}_{\mathcal{S}}(f) &= \sum_{\substack{m=1\\ m:\,\mathrm{odd}}}^{t} \binom{t}{m} p^m (1-p)^{t-m} \\ &= \frac{1-(1-2p)^t}{2}. \end{aligned} \tag{2.29}$$

To compare this quantity with the influence obtained when all flipped variables belong to a single product subfunction, recall (2.28). Consequently, for $d \geq 2$, we have $d-1 \geq 1$, and thus $2^{d-1} \geq 2$, which implies $0 < p = \frac{1}{2^{d-1}} \leq \frac{1}{2}$. Consequently, we have $0 \leq 1-2p < 1$. Since $t \geq 1$, raising a number in $[0,1]$ to the power t cannot increase it, and hence $(1-2p)^t \leq 1-2p$. It follows that

$$\mathrm{Inf}_{\mathcal{S}}(f) = \frac{1-(1-2p)^t}{2} \geq \frac{1-(1-2p)}{2} = p = \frac{1}{2^{d-1}}.$$

Therefore, a subset that mixes variables from several product subfunctions cannot have smaller joint influence than a subset whose variables belong to a single product subfunction. This completes the proof. □

From Lemma 2, the subsets with the lowest joint influence include datasets from the same f_n. We next derive a lower bound on the average joint sensitivity for the proposed setting using Lemmas 1-2, and present the communication-optimal placement configuration in Theorem 2.

Theorem 2 (Joint influence-optimal and communication-optimal placement configuration)**.** *Consider the partitioned placement setting described above, with $d = M \geq 2$, $K = NM$, and*

$$f(\mathbf{W}) = \bigoplus_{n=1}^{N} f_n(\mathbf{W}), \qquad f_n(\mathbf{W}) = \prod_{W_k \in \mathcal{I}_{f_n}} W_k,$$

where the sets $\{\mathcal{I}_{f_n}\}_{n\in[N]}$ form a partition of $\{W_1, \ldots, W_K\}$. Define

$$\mathcal{S}^\star \triangleq \{\mathcal{S}_n^\star : n \in [N]\}, \quad \textit{where} \quad \mathcal{S}_n^\star = \mathcal{I}_{f_n}.$$

Then, for every admissible partitioned placement $\mathcal{S} = \{\mathcal{S}_n : n \in [N]\}$,

$$\mathrm{as}_{\mathcal{S}}(f) \geq \mathrm{as}_{\mathcal{S}^\star}(f) = \frac{N}{2^{M-1}}, \tag{2.30}$$

$$\tau_{\mathcal{S}} \geq \tau_{\mathcal{S}^\star} = N. \tag{2.31}$$

Proof. We establish the two claims separately.

Optimality of the average joint sensitivity. Under $\mathcal{S}^\star$, every server stores exactly the variables appearing in one product subfunction. By Lemma 1,

$$\mathrm{Inf}_{\mathcal{S}_n^\star}(f) = \frac{1}{2^{M-1}}, \qquad n \in [N].$$

Therefore,

$$\begin{aligned} \mathrm{as}_{\mathcal{S}^\star}(f) &= \sum_{n=1}^{N} \mathrm{Inf}_{\mathcal{S}_n^\star}(f) \\ &= \frac{N}{2^{M-1}}. \end{aligned} \tag{2.32}$$

Moreover, Lemma 2 shows that replacing variables from one product subfunction by variables belonging to other product subfunctions cannot decrease the corresponding joint influence. Consequently, every placement subset $\mathcal{S}_n$ intersects at least one product support and satisfies

$$\mathrm{Inf}_{\mathcal{S}_n}(f) \geq \frac{1}{2^{M-1}}.$$

Summing over all $n \in [N]$ therefore gives

$$\mathrm{as}_{\mathcal{S}}(f) = \sum_{n=1}^{N} \mathrm{Inf}_{\mathcal{S}_n}(f) \geq \frac{N}{2^{M-1}} = \mathrm{as}_{\mathcal{S}^\star}(f).$$

Achievability of N transmissions. Under $\mathcal{S}^\star$, server n can compute and transmit the single binary symbol

$$Z_n^\star = f_n(\mathbf{W}) = \prod_{W_k \in \mathcal{S}_n^\star} W_k.$$

The user then recovers

$$f(\mathbf{W}) = \bigoplus_{n=1}^{N} Z_n^\star.$$

Hence,

$$\tau_{\mathcal{S}^\star} = \sum_{n=1}^{N} |Z_n^\star| = N.$$

Communication converse. Consider an arbitrary admissible partitioned placement $\mathcal{S}$, and fix a server $n \in [N]$. Choose any dataset $W_k \in \mathcal{S}_n$.

Since the function supports $\{\mathcal{I}_{f_j}\}_{j\in[N]}$ partition $\{W_1, \ldots, W_K\}$, the variable W_k appears in exactly one product subfunction, say f_j.

We can choose two global dataset realizations $\mathbf{W}$ and $\mathbf{W}'$ that differ only in their k-th entry and satisfy $f(\mathbf{W}) \neq f(\mathbf{W}')$. For example, set all other variables appearing in f_j equal to one, and force every other product subfunction to zero by setting at least one of its variables equal to zero. Flipping only W_k then changes the value of f_j, and hence changes the value of the Boolean function f.

Because the placement sets form a partition, the local observations of all servers $n' \neq n$ are identical under $\mathbf{W}$ and $\mathbf{W}'$. Their transmitted messages are therefore also identical. To permit zero-error recovery, server n's message must distinguish these two local realizations. Consequently, its encoding function cannot be constant, and hence

$$|Z_n| \geq 1.$$

Since this argument holds for every $n \in [N]$,

$$\tau_{\mathcal{S}} = \sum_{n=1}^{N} |Z_n| \geq N\,.$$

Together with the achievable construction under $\mathcal{S}^\star$, this proves

$$\tau_{\mathcal{S}} \geq \tau_{\mathcal{S}^\star} = N\,. \qquad \square$$

Remark 1. *The average joint sensitivity and the total communication cost are optimized by the same placement configuration $\mathcal{S}^\star$ in the considered partitioned*

setting. However, the result does not imply a general one-to-one or monotonic relationship between the individual message length $|Z_n|$ *and the joint influence* $\mathrm{Inf}_{\mathcal{S}_n}(f)$*. The optimality of the average joint sensitivity follows from Lemmas 1 and 2, whereas the communication lower bound follows from the zero-error distinguishability argument in the proof of Theorem 2.*

2.5 Conclusions

In this chapter, we used the notions of sensitivity and influence to study the relationship between data placement and communication cost in the distributed computation of Boolean functions. In particular, we showed that the placement configuration plays a central role in determining how much information the servers must transmit for the user to recover the requested function. For a class of linearly-separable Boolean functions, we characterized a placement strategy that minimizes the communication cost by exploiting the structure of the function and its sensitivity to groups of the underlying datasets. The proposed approach groups datasets so as to minimize the aggregate joint influence within each group, thereby assigning strongly interacting datasets to the same server whenever beneficial. This sensitivity-based perspective provides a systematic way to connect the algebraic structure of a Boolean function with the design of communication-efficient placement configurations.

Possible future research directions include extending this framework to non-linear Boolean functions of arbitrary degree and to probabilistic models in which the datasets are non-uniformly distributed or statistically correlated. In such settings, the influence of each dataset depends not only on the algebraic structure of the requested function but also on the underlying source distribution, potentially leading to new placement–communication tradeoffs. Another relevant direction is to consider heterogeneous servers with different storage/computational capacities and communication constraints.

Chapter 3

Sparse Tensor Factorization Approach

This chapter focuses on multi-user distributed computing systems in which users request the evaluation of non-linearly separable real-valued functions. Unlike linearly decomposable settings, these functions exhibit complicated multi-variable interactions that cannot be captured through simple additive or separable structures, thereby introducing significant challenges in both computation and communication design.

To address this complexity, we reformulate the problem through a tensor-theoretic lens, where user demands are represented as high-dimensional tensors encoding the interactions among multiple input variables. Within this framework, the computation task is framed as a sparse tensor factorization problem, where the goal is to decompose the demand tensor into structured components that can be efficiently distributed across servers and computed.

The proposed formulation explicitly incorporates system-level constraints, including limited computational capabilities at each server and restricted communication bandwidth to the users. These constraints naturally translate into sparsity and structural restrictions on the factorization, governing both the number of active dimensions processed by each server and the number of users it can simultaneously serve. By leveraging this structured sparsity, we design efficient achievable schemes that significantly reduce the required computational resources compared to conventional approaches, while preserving the exact computation of the desired functions.

Relation to author's prior publications. This chapter is based on [101], [102] and their extended version in [103]. The presentation has been substantially revised and expanded for this thesis, with unified notation, a reformulated system model, additional explanations, and updated conclusions.

Unless otherwise indicated, the technical statements, theorem formulations, proofs, examples, and figures in this chapter are adapted from [103].

3.1 Introduction

The increasing scale and complexity of modern computational workloads have made distributed computing an indispensable paradigm [127], particularly in the context of large-scale ML systems. General-purpose frameworks such as MapReduce [33] and Spark [39] offer parallel execution across multiple servers and have enabled a broad range of distributed computation applications; see, for example, [4], [69], [74], [80], [88], [105], [128], [129], [130], [131], [132], [133], [134], [135]. Nevertheless, the finite computational capacity of the servers and the limited communication resources of the underlying network remain key constraints in the design of efficient distributed computing schemes.

Motivated by the complementary limitations of computation and communication, as well as the growing complexity of distributed workloads, we investigate multi-user distributed computation for *non-linearly separable functions*. The system consists of N servers with constrained computation and communication resources and K users, each requesting a polynomial function of L real-valued, potentially non-linear, basis subfunctions. A coordinator distributes the required computation tasks among the servers so that every user can recover its requested function without error. Our objective is to reduce the number of required servers N, or equivalently, to maximize the *achievable system rate* K/N. Within this framework, we develop task-assignment and communication strategies and derive bounds on the achievable system rate.

Before introducing the general formulation, we begin with a simple example that highlights the main system constraints and performance metrics.

Example 2. *We first examine the structure of the demanded functions through this example. Consider a system setting with $L = 4$ basis subfunctions*

$$f_1(\cdot) \triangleq e^{x_1}\,, \quad f_2(\cdot) \triangleq \log(x_2)\,, \quad f_3(\cdot) \triangleq \sqrt{x_2}\,, \quad f_4(\cdot) \triangleq \cos(x_3)\,,$$

which operate on fixed input vectors $x_1, x_2, x_3 \in \mathbb{R}^B$ and generate the output files [1] $\{W_\ell = f_\ell(\cdot)\}_{\ell\in[4]}$. Let us consider the demanded function

$$F_1(\mathbf{W}) = 7\,W_1^2W_2^3 + 8\,W_1W_3^2W_4 + 6\,W_3W_4^4 + 4\,W_1^4W_4^2 \tag{3.1}$$

by a first user, which is obviously non-linearly separable over the variables in

$$\mathbf{W} = (W_1, W_2, W_3, W_4)\,.$$

[1] All operations are naturally component-wise here.

Our construction is characterized by the number of basis subfunctions L, with $L = 4$ in this example, and by the per-output exponent bounds $\{P_\ell\}_{\ell\in[L]}$. Specifically, the exponent of W_ℓ is restricted to

$$(p_\ell - 1) \in \{0, 1, \ldots, P_\ell - 1\}, \qquad \ell \in [L].$$

Equivalently, $(P_\ell - 1)$ denotes the maximum degree of W_ℓ in $\mathbf{W}$ that may appear in any demanded polynomial as a computational task.

In Example 2, the maximum degrees of the basis subfunctions in (3.1) *are*

$$P_1 - 1 = 4, \qquad P_2 - 1 = 3, \qquad P_3 - 1 = 2, \qquad P_4 - 1 = 4, \tag{3.2}$$

or, equivalently,

$$(P_1, P_2, P_3, P_4) = (5, 4, 3, 5).$$

As an illustration, $P_3 = 3$ means that the exponent of W_3 cannot exceed 2. This maximum exponent appears in the monomial $W_1W_3^2W_4$, which is the second additive term of the requested function $F_1(\mathbf{W})$ in (3.1).

Next, suppose that a second user requests a function of the form

$$F_2(\mathbf{W}) = 3\,W_2W_3^3 + 2\,W_1^3W_3 + 11\,W_1^2W_2 + 13\,W_2^2W_4^3\,. \tag{3.3}$$

We observe that the specified degree limits $4, 3, 2, 4$, which must be satisfied by all requested functions, are violated by the second polynomial in (3.3). *Indeed, its maximum degrees are*

$$P_1' - 1 = 3, \qquad P_2' - 1 = 2, \qquad P_3' - 1 = 3, \qquad P_4' - 1 = 3,$$

corresponding to

$$(P_1', P_2', P_3', P_4') = (4, 3, 4, 4).$$

In particular, the degree of W_3 exceeds the declared maximum of 2 in (3.2).

Consider $K = 2$ users requesting $F_1(\mathbf{W})$ and $F_2(\mathbf{W})$, respectively, and $N = 3$ distributed servers. The network topology, illustrated in Figure 3.1, allows each server to communicate with at most $\Delta \leq K$ users. In this example, we set $\Delta = 1$. Each user must then recover its requested function by linearly combining the signals received from the connected servers.

A second system parameter is the computation budget $\Gamma \leq L$, which limits the number of basis subfunctions that each server can evaluate. Setting $\Gamma = 2$ would make the exact computation of $F_1(\mathbf{W})$ impossible, since its second term, $8W_1W_3^2W_4$, depends jointly on three basis subfunctions and therefore could not be generated at any server. Hence, this example requires $\Gamma = 3$.

Finally, we impose an additional computation constraint through the exponent ranges Λ_ℓ associated with each output file W_ℓ, for $\ell \in [L]$. These

parameters restrict the powers of the output file that can be locally generated at a server and, consequently, determine the number of multiplications required to produce the desired exponents.

The main challenging procedure in this chapter is to determine which basis subfunctions, together with which range of their powers, should be assigned to each server, while also designing the corresponding linear encoding and decoding operations at the servers and users. One possible alternative is to "linearize" the requested polynomials and cast the problem as a linearly-separable computation task [2] of the type studied in [105]. Instead, we develop a more general formulation in which the original problem is represented as a sparse tensor decomposition. This tensor-based approach can provide substantial improvements over direct linearization. The gain is illustrated in Example 4 of Section 3.4 and is further demonstrated through the numerical results in Section 3.7 over a range of system parameters.

3.1.1 Related Works

Earlier works in the literature have extensively studied the fundamental limits of distributed computation for linearly-separable functions. These studies cover gradient computation [74], [88], [129], [130], distributed matrix multiplication [80], [133], [134], [136], and polynomial computation [100], [105], [137]. Among these works, the framework in [105] most closely relates to our setting. For linearly-separable functions, the authors connect distributed computation under support-sparsity constraints to a sparse matrix factorization problem and develop tessellation-based constructions that satisfy these constraints. Their approach partitions the demand matrix into submatrices of suitable dimensions and uses the sparsity pattern and rank of each submatrix to determine the required computational resources [3]. However, a matrix-factorization approach treats each distinct monomial as a separate effective basis subfunction. This representation enlarges the underlying space and may require excessive resources when the requested functions contain non-linear interactions among the original basis subfunctions. We therefore develop a new framework that directly captures the structure of non-linearly separable functions. Emerging ML applications [132], [138], [139], particularly large language models, further motivate this framework because their distributed

[2] For the example above, linearization would introduce $L' = 8$ effective subfunctions. The first two would be $W_1^2 W_2^3$ and $W_1 W_3^2 W_4$, respectively, with the remaining monomials treated in the same manner.

[3] For an $A \times B$ submatrix, the maximum possible rank equals $\min(A, B)$, where A and B denote the numbers of rows and columns, respectively.

execution requires many non-linear operations. For example, activation functions [140] and attention mechanisms [141] may involve higher-order terms and products of the underlying basis subfunctions [142].

Many existing methods, including the matrix factorization-based scheme in [105] and coded computing schemes such as [70], [143], [144], [145], assume that the users' requested functions are linearly-separable. Non-linearly separable functions, however, require a richer representation that records the degree of each basis subfunction within every monomial. These additional indices naturally lead to a tensor representation, which directly captures the non-linear interactions among the basis subfunctions. From a practical point of view, the communication system models have also used tensors to describe high-dimensional signal interactions and access patterns, particularly in massive random access [146].

Some of the existing approaches, such as the tessellated distributed computing (TDC) framework [105], also impose structural or disjoint support conditions on the users' demands. Under these assumptions, the relevant supports occupy separate regions of the demand matrix. Such separation does not generally hold for non-linearly separable functions, whose monomials can produce overlapping supports and overlapping submatrices. Applying rigid matrix partitions in this setting may therefore lead to inefficient decompositions and prevent the construction of low-rank representations. In particular, the ranks of the resulting submatrices can grow rapidly with the combinatorial structure of the demand tensor, or of its matricized form [4]. These difficulties also reflect the computational hardness of the underlying decomposition problems. Minimizing the rank of sparse or structured matrices is NP-hard [147], [148], while tensor rank minimization remains NP-hard even to approximate [149]. Consequently, existing methods generally resort to heuristic constructions or simplified decomposition strategies.

Therefore, the distinction from TDC is also representational: while TDC operates on a matrix factorization-based formulation after expressing the demand through effective linearly-separable subfunctions, the proposed tensor-based formulation in this chapter retains the original polynomial interactions as modes of a higher-order tensor and subsequently exploits this multi-way structure through the sparse tiling and combinatorial assignment procedures developed in Section 3.6.1 and Section 3.6.2, respectively.

[4]Matricization, also known as unfolding or flattening, reshapes a tensor into a matrix by grouping a subset of its modes into the row index and the remaining modes into the column index.

3.1.2 Our Contributions

In this work, we develop a framework for the lossless distributed computation of non-linearly separable functions. We summarize our main contributions below.

- **Tensor representation of user demands:** We represent the users' demands through a full-rank tensor $\bar{\mathcal{F}}$ that preserves their underlying polynomial structure. By assigning the exponent range of each basis subfunction to a separate tensor mode, this representation captures a broad class of non-linearly separable functions.

- **Sparse tensor factorization framework:** We formulate the distributed computing problem through the sparse factorization $\bar{\mathcal{F}} = \bar{\mathcal{E}} \times_1 \mathbf{D}$, where Section 3.4 defines the mode-1 product $\times_1$. For each server n, the encoding tensor $\bar{\mathcal{E}}$ specifies the subset of basis subfunctions and the corresponding power terms that the server computes. The decoding matrix $\mathbf{D}$ determines the set of users, of cardinality at most Δ, to which each server transmits. The support constraints on $\bar{\mathcal{E}}$ and $\mathbf{D}$ therefore capture the computation and communication costs, respectively.

- **Achievable scheme and system-rate bound:** We develop a lossless sparse tensor factorization scheme in Section 3.6.1. The scheme partitions $\bar{\mathcal{F}}$ into carefully selected subtensors and factorizes each of them into compatible components of $\bar{\mathcal{E}}$ and $\mathbf{D}$. Using tools from fixed-support sparse matrix factorization and multilinear singular value decomposition [150], [151], we derive a lower bound on the achievable system rate K/N in Theorem 3.

  A direct alternative would first matricize the demand tensor. However, this transformation produces a very large matrix whose effective subfunction space grows exponentially with the number of basis subfunctions and combinatorially with their degree bounds. It therefore introduces substantial rank and computation overhead. In comparison with the sparse matrix factorization method of [105], our tensor-based construction achieves an exponential reduction in the total computation cost, as shown in Proposition 1, while also reducing the required number of servers and communication resources, as illustrated in Cases I and II of Example 4.

- **Assignment-based combinatorial reduction:** We introduce a tile-assignment framework in Algorithm 1, described in Section 3.6.2, to

resolve overlaps among the subtensors, or tile closures. Rather than assigning the same index tuple, and therefore the same demand coefficient, to several tiles, the algorithm assigns each tuple to exactly one feasible tile while respecting the corresponding capacity constraints.

This disjoint assignment reduces the sum rank of the resulting subtensors and, consequently, the number of required servers, as established in Theorem 4. The method removes redundant contributions created by overlapping supports without imposing disjoint-support assumptions or solving a general rank-minimization problem. By combining structured tensor factorization with a global tuple-to-tile assignment, the proposed scheme uses the available computational resources more efficiently than the default tensor decomposition of Theorem 3. Section 3.7 further demonstrates the gains obtained by this overlap-resolution step.

We also extend both single-shot constructions to a multi-shot setting with T shots in Section 3.8 (see Theorems 5 and 6).

- **Numerical evaluation of the assignment algorithm:** In Section 3.7, we evaluate the exact number of servers N_{alg} produced by Algorithm 1 under several system configurations. The results show that the assignment-based construction reduces N compared with both the TDC scheme of [105] and the default tensor factorization scheme with overlapping subtensors given in Theorem 3.

Notation. For a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, we define its binary support matrix as $\text{Supp}(\mathbf{X})(i,j) \triangleq \mathbb{1}\{\mathbf{X}(i,j) \neq 0\}$, $(i,j) \in [m] \times [n]$. Hence, $\text{Supp}(\mathbf{X}) \in \{0,1\}^{m\times n}$ indicates the locations of the nonzero entries of $\mathbf{X}$. These support and indexing conventions extend naturally to tensors. In particular, for an order-d tensor $\bar{\mathcal{X}} \in \mathbb{R}^{I_1\times\cdots\times I_d}$, the notation $\bar{\mathcal{X}}(i_1,\ldots,i_d)$ denotes its entry indexed by $(i_1,\ldots,i_d)$, while a colon represents all indices along the corresponding mode. For example, $\bar{\mathcal{X}}(i_1,:,\ldots,:)$ denotes the subtensor obtained by fixing the first-mode index to i_1. Similarly, $\text{Supp}(\bar{\mathcal{X}})$ denotes the binary support tensor indicating the locations of the nonzero entries of $\bar{\mathcal{X}}$. Finally, $\vee$ denotes the logical "OR" operation.

3.2 System Model

Motivated by practical distributed systems like MapReduce [33], we consider a distributed computing system with N servers and K users. Each user requests the exact evaluation of an arbitrary multivariate polynomial constructed from L real-valued, potentially non-linear basis subfunctions $\{f_\ell(\cdot)\}_{\ell\in[L]}$.

The user demands in this chapter are arbitrary real-valued multivariate polynomial functions of L basis subfunctions, subject to prescribed degree limits $\{P_\ell - 1\}_{\ell\in[L]}$. These degree limits are part of the system model and define the size of the demand tensor. The computation constraint Γ limits the number of active basis subfunctions that can be handled by a server, while Λ_ℓ limits the range of powers of W_ℓ assigned to a server. The proposed construction is lossless and assumes that the demand coefficients are known to the coordinator when assigning computation and communication tasks.

A coordinator node orchestrates the distributed computation of the demanded functions through three phases, described next.

a) **Demand Phase**.

The demand phase is the initial phase in which each user $k \in [K]$ independently requests the evaluation of one real-valued function expressed in the following multivariate form

$$F_k(W_1, \ldots, W_L) = \sum_{\underline{\mathbf{p}}\in\prod_{\ell\in[L]}[P_\ell]} c_{k,\underline{\mathbf{p}}} \prod_{\ell\in[L]} W_\ell^{p_\ell-1} \tag{3.4}$$

where

$$W_\ell \triangleq f_\ell(x), \qquad \ell \in [L],$$

denotes the real-valued output of the basis subfunction $f_\ell(\cdot)$ evaluated at a common input vector $x \in \mathbb{R}^d$. We refer to $\{W_\ell\}_{\ell\in[L]}$ as the ℓ-th *output file*. The vector x represents the underlying data instance, such as a feature vector or signal, on which all basis subfunctions are evaluated. Here, $c_{k,\underline{\mathbf{p}}} \in \mathbb{R}$ denotes the *basis coefficient* associated with the monomial

$$\prod_{\ell\in[L]} W_\ell^{p_\ell-1},$$

where

$$\underline{\mathbf{p}} \triangleq (p_1, \ldots, p_L)$$

indexes the exponent vector $(p_1 - 1, \ldots, p_L - 1)$. Defining

$$W'(\underline{\mathbf{p}}) \triangleq \prod_{\ell\in[L]} W_\ell^{p_\ell-1},$$

allows us to rewrite (3.4) as

$$F_k = \sum_{\underline{\mathbf{p}}\in\prod_{\ell\in[L]}[P_\ell]} c_{k,\underline{\mathbf{p}}}\, W'(\underline{\mathbf{p}}). \tag{3.5}$$

This representation casts the problem as a classical multi-user gradient coding problem by treating each monomial $W'(\underline{\mathbf{p}})$ as a separate basis subfunction. In contrast, our formulation works directly with the high-dimensional structure in (3.4) and explicitly incorporates the multiplication cost required to generate the different powers of the output files. Accounting for these costs introduces additional algebraic challenges, which we describe in the following phases.

b) **Non-Linear Computing Phase**.

The coordinator then assigns each server n a subset of basis subfunctions $\mathcal{S}_n \subseteq [L]$ for local evaluation. Accordingly, server n computes

$$\{W_\ell = f_\ell(x)\}_{\ell \in \mathcal{S}_n}.$$

We measure the resulting local workload through the computation cost

$$\Gamma \triangleq \max_{n \in [N]} |\mathcal{S}_n| \tag{3.6}$$

where Γ denotes the maximum number of basis subfunctions that any server can evaluate locally. We therefore impose the per-server computation constraint

$$|\mathcal{S}_n| \leq \Gamma \leq L, \qquad n \in [N].$$

Consistent with this constraint, we restrict each nonzero monomial in a user's demand to involve at most Γ basis subfunctions. More precisely,

$$c_{k,\underline{\mathbf{p}}} = 0$$

in (3.4) whenever the support of the exponent vector $(p_1 - 1, \ldots, p_L - 1)$ has cardinality greater than Γ. Thus, every monomial with a nonzero coefficient depends on at most Γ basis outputs, in accordance with the local computation budget.

The coordinator further assigns each server $n \in [N]$ a set of exponent vectors

$$\mathcal{P}_n \subseteq \prod_{\ell \in [L]} [P_\ell].$$

For every $\underline{\mathbf{p}} \in \mathcal{P}_n$, server n computes the required power terms

$$\{W_\ell^{p_\ell - 1}\}_{\ell \in \mathcal{S}_n}$$

and multiplies them to form the corresponding monomial

$$\prod_{\ell \in \mathcal{S}_n} W_\ell^{p_\ell - 1}.$$

These local power terms and multiplication operations make the computation phase inherently non-linear.

To quantify the multiplication cost, we assume that, for each W_ℓ with $\ell \in \mathcal{S}_n$, server n computes an ordered set of exponents of cardinality [5] Λ_ℓ. Specifically, for every $\ell \in \mathcal{S}_n$, the server evaluates the ordered set

$$[[q\Lambda_\ell + 1 : (q+1)\Lambda_\ell]], \qquad q \in \mathbb{N}. \tag{3.7}$$

Equivalently, when server n generates a positive power term W_ℓ^α required by a user's demand, it expresses the exponent α as

$$\alpha \triangleq q\Lambda_\ell + r, \qquad q \in \mathbb{N}, \qquad r \in [[1 : \Lambda_\ell]]. \tag{3.8}$$

The case $\alpha = 0$ corresponds to $W_\ell^0 = 1$ and requires no multiplication. Based on the exponent range available at the server in (3.7), we rewrite the demanded power as

$$W_\ell^\alpha = W_\ell^{q\Lambda_\ell + 1} W_\ell^{r-1}.$$

Using the standard square-and-multiply method, this gives the following cost for computing W_ℓ^α:

$$\lfloor \log_2(q\Lambda_\ell + 1) \rfloor + (\mathrm{wt}_2(q\Lambda_\ell + 1) - 1) + (r - 1), \tag{3.9}$$

where $\mathrm{wt}_2(a)$ denotes the Hamming weight of the binary representation of the positive integer a, i.e., the number of its non-zero binary digits. The term $\lfloor \log_2 a \rfloor$ counts the squaring operations, while $\mathrm{wt}_2(a) - 1$ counts the additional multiplications associated with the non-leading binary digits equal to one. The subtraction of one arises because the leading binary digit initializes the computation with W_ℓ and therefore requires no multiplication. Hence, the first two terms account for computing the anchor power $W_\ell^{q\Lambda_\ell + 1}$ using the square-and-multiply method, whereas the final $r - 1$ term accounts for the successive multiplications by W_ℓ needed to reach the desired exponent within the assigned range. Consequently, the number of self-multiplications required for the output

[5] If the servers have heterogeneous computational capabilities, this cardinality may depend on both ℓ and n, and can be denoted by $\Lambda_{\ell,n}$.

of basis subfunction $f_\ell(\cdot)$ at each server scales as $\mathcal{O}(\log \alpha + \Lambda_\ell)$, whereas naive repeated multiplication incurs a cost of $\mathcal{O}(\alpha)$. When the cost of generating the assigned exponent range dominates the logarithmic anchor-computation cost, the overall complexity behaves as $\mathcal{O}(\Lambda_\ell)$.

For example, when $\alpha = 851$ and $\Lambda_\ell = 100$, we have $q = 8$, $r = 51$, and $q\Lambda_\ell + 1 = 801$. Since

$$801 = (1100100001)_2 \qquad \text{and} \qquad \mathrm{wt}_2(801) = 4,$$

the cost expression in (3.9) gives

$$\lfloor \log_2(801) \rfloor + (4-1) + (51-1) = 9 + 3 + 50 = 62$$

multiplications, compared to $\alpha - 1 = 850$ multiplications under the naive method. Moreover, in the high-degree regime of interest, we assume that the number of exponent levels of each output file is much larger than the number of basis subfunctions involved in a monomial, i.e.,

$$\Lambda_\ell \gg \Gamma, \qquad \forall\, \Gamma \le L.$$

Once the required power terms

$$\{W_\ell^{p_\ell - 1}\}_{\ell \in \mathcal{S}_n}$$

have been generated, combining them into the monomial

$$\prod_{\ell \in \mathcal{S}_n} W_\ell^{p_\ell - 1}, \qquad \underline{\mathbf{p}} \in \mathcal{P}_n,$$

requires at most $|\mathcal{S}_n| - 1 \le \Gamma - 1$ additional multiplications. This cost is therefore negligible compared with the cost of generating the individual power terms, which scales with the cardinality of the exponent ranges, described by $\{\Lambda_\ell\}_{\ell \in \mathcal{S}_n}$.

We therefore use the cardinality of the assigned exponent range as the multiplication cost parameter at each server, defined as

$$\Lambda_\ell \triangleq \max_{\underline{\mathbf{p}} \in \mathcal{P}_n} p_\ell - \min_{\underline{\mathbf{p}} \in \mathcal{P}_n} p_\ell + 1\,, \quad \ell \in \mathcal{S}_n \tag{3.10}$$

These parameters specify the exponent ranges of the output files

$$\{W_\ell\}_{\ell \in \mathcal{S}_n}$$

that server n must generate. They therefore determine the largest local multiplication cost required to evaluate any multiplicative term in (3.4). For each $\ell \in [L]$, we impose the multiplication constraint

$$\Lambda_\ell \leq P_\ell,$$

which ensures that the exponent range of W_ℓ assigned to a server does not exceed the maximum exponent range per output file each server can evaluate.

c) **Communication Phase**.

After completing its assigned local computations, each server $n \in [N]$ constructs the signals

$$z_n \triangleq \sum\nolimits_{\underline{\mathbf{p}} \in \mathcal{P}_n} e_{n,\underline{\mathbf{p}}} \prod\nolimits_{\ell \in [L]} W_\ell^{p_\ell - 1}, \quad n \in [N] \tag{3.11}$$

where the coefficients $e_{n,\underline{\mathbf{p}}} \in \mathbb{R}$, for $n \in [N]$ and $\underline{\mathbf{p}} \in \mathcal{P}_n$, determine the non-linear computation and encoding performed by server n. For every $\ell \notin \mathcal{S}_n$, we set $p_\ell = 1$, and hence $W_\ell^{p_\ell - 1} = W_\ell^0 = 1$. This convention indicates that neither the corresponding basis subfunction nor any of its nonzero power terms contributes to the multiplicative terms computed by server n.

Server n then transmits z_n to a subset of users

$$\mathcal{T}_n \subseteq [K]$$

over an error-free shared link. In the final decoding stage, each user $k \in [K]$ linearly combines the signals received from its connected servers to recover

$$F_k' \triangleq \sum\nolimits_{n \in [N]} d_{k,n} z_n \tag{3.12}$$

where the decoding coefficients $d_{k,n} \in \mathbb{R}$, for $n \in [N]$, determine how user k linearly combines the server transmissions. We set

$$d_{k,n} = 0, \qquad k \notin \mathcal{T}_n,$$

since user k does not receive the signal transmitted by server n whenever k lies outside its intended recipient set $\mathcal{T}_n$.

We measure the communication burden at each server through the cost

$$\Delta \triangleq \max_{n \in [N]} |\mathcal{T}_n| \tag{3.13}$$

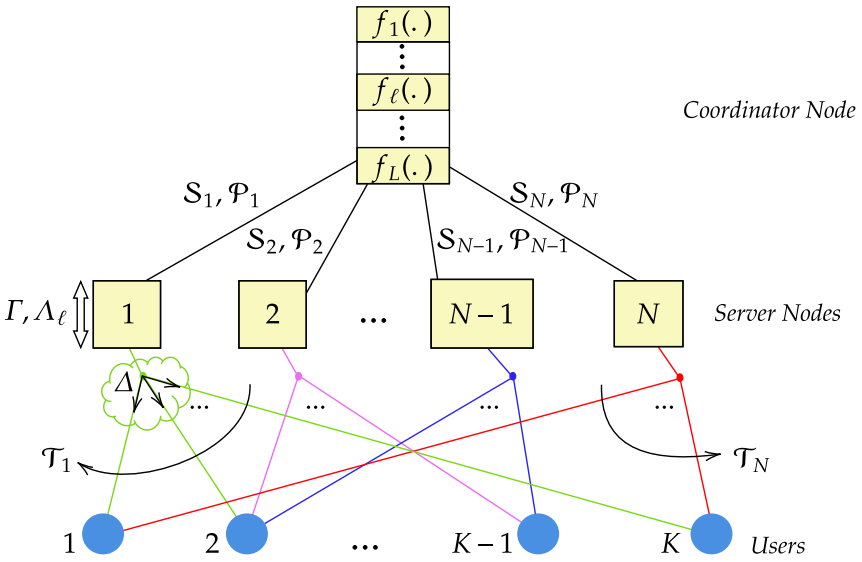

Figure 3.1: The lossless $(K, N, T, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell\in[L]})$ distributed computing setting with a coordinator node, N servers, and K users.

where Δ denotes the maximum number of users that any server can serve. Accordingly, the communication constraint satisfies

$$\Delta \leq K.$$

We emphasize that Γ, Δ, and Λ_ℓ, $\ell \in [L]$ impose strict per-instance constraints on the considered distributed system. Every valid scheme must therefore satisfy them for each problem setting. Accordingly, the costs defined in (3.6), (3.13), and (3.10) lead to a system characterized by the normalized constraints

$$\gamma = \frac{\Gamma}{L}\ , \quad \delta = \frac{\Delta}{K}\ , \quad \left\{\lambda_\ell = \frac{\Lambda_\ell}{P_\ell}\right\}_{\ell\in[L]} \tag{3.14}$$

where $\gamma, \delta, \lambda_\ell \in [0, 1]$, $\ell \in [L]$. Accordingly, for each server n, the scheme must determine the assigned basis subfunction set $\mathcal{S}_n$, the corresponding exponent vector set $\mathcal{P}_n$, and the intended user set $\mathcal{T}_n$.

Serving a large number of users with a limited number of servers increases the load on the system. We capture this burden through the *system rate*, defined as

$$R \triangleq \frac{K}{N}\ . \tag{3.15}$$

We consider the system model shown in Figure 3.1, characterized by the set of system parameters $\Big(K, N, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell\in[L]}\Big)$. Our objective is to design a scheme that enables every user to recover its requested function without error while satisfying the predetermined computation, communication, and multiplication constraints. For arbitrary values of $\{\Lambda_\ell, P_\ell\}_{\ell\in[L]}$ and any $\Gamma \geq 1$, one possible approach is to apply the matrix factorization framework of [105]. In particular, we can rewrite (3.4) in the linearly-separable form of (3.5) by treating each monomial $W'(\underline{\mathbf{p}})$ as a separate output file. Representing all multiplicative terms in (3.4) in this manner requires

$$L' \triangleq \prod_{\ell\in[L]} P_\ell \tag{3.16}$$

new basis subfunctions, one for each exponent tuple $\underline{\mathbf{p}} \in \prod_{\ell\in[L]}[P_\ell]$.

For the single-shot setting, Theorem 1 of [105] requires

$$N = \frac{K}{\Delta}\frac{L'}{\Gamma}\min(\Delta, \Gamma) \tag{3.17}$$

servers. The number of effective basis subfunctions for representing user demands in linearly-separable form (cf. (3.16)) grows exponentially with L as the exponent bounds P_ℓ are always larger than one. Consequently, the number of servers required by this linearized construction will also scale exponentially with the number of original basis subfunctions.

We next introduce the basics of tensors and review the definitions required to formulate our problem in tensor form.

3.3 Basic Tensor Definitions

We first introduce a tensor in the following definition.

Definition 8. An order-N tensor, $\bar{\mathcal{X}} \in \mathbb{R}^{I_1\times\cdots\times I_N}$, *is a multi-way array with N modes, with the n-th mode of dimensionality I_n, for $n \in [N]$. Special cases of tensors include matrices as order-2 tensors (e.g., $\mathbf{X} \in \mathbb{R}^{I_1\times I_2}$), vectors as order-1 tensors (e.g., $\boldsymbol{x} \in \mathbb{R}^{I_1}$), and scalars as order-0 tensors (e.g., $x \in \mathbb{R}$).*

We next define the matrix unfolding of a tensor.

Definition 9. *A* mode-n unfolding *of a tensor is the procedure of mapping the elements from a multidimensional array to a two-dimensional array (matrix). Conventionally, such a procedure is associated with stacking mode-n fibers*

(modal vectors) as column vectors of the resulting matrix. For instance, mode-1 unfolding of $\bar{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *is represented as* $\bar{\mathcal{X}}_{(1)} \in \mathbb{R}^{I_1 \times I_2 I_3 \cdots I_N}$*, and given by*

$$\bar{\mathcal{X}}_{(1)}\Big(i_1, \overline{i_2 i_3 \dots i_N}\Big) = \bar{\mathcal{X}}(i_1, i_2, \dots, i_N)\,. \tag{3.18}$$

Note that the overlined subscripts refer to linear indexing (or Little-Endian ordering) [152], where the linear index is given by

$$\begin{aligned}\overline{i_1 i_2 \dots i_N} &= 1 + \sum_{n=1}^{N} (i_n - 1) \prod_{k=1}^{n-1} I_k \\ &= 1 + i_1 + (i_2 - 1) I_1 + \cdots + (i_N - 1) I_1 \dots I_{N-1}.\end{aligned} \tag{3.19}$$

We proceed to define tensor folding of a vector as follows.

Definition 10. *Any given vector* $\mathbf{x} \in \mathbb{R}^{I_1 I_2 \dots I_N}$ *can be* folded *into an N-th order tensor,* $\bar{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$*, with the relation between their entries defined by*

$$\bar{\mathcal{X}}(i_1, i_2, \dots, i_N) = \mathbf{x}(i), \quad \forall i_n \in [I_n] \tag{3.20}$$

where $i = 1 + \sum_{n=1}^{N} (i_n - 1) \prod_{k=1}^{n-1} I_k$.

We next formally define the stacking operation in tensors.

Definition 11. *Consider grouping J order-N tensor samples* $\bar{\mathcal{X}}_j \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $j \in [J]$*, so as to form an order-*$(N+1)$ *data tensor,* $\bar{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times J}$*. This* stacking *operation is denoted by*

$$\bar{\mathcal{Y}} = stack_{N+1}\left(\bar{\mathcal{X}}_1, \dots, \bar{\mathcal{X}}_J\right). \tag{3.21}$$

In other words, the combined tensor samples introduce another dimension, the $(N+1)$*-th mode of* $\bar{\mathcal{Y}}$*, such that its mode-*$(N+1)$ *unfolding* $\bar{\mathcal{Y}}_{(N+1)} \in \mathbb{R}^{(I_1 I_2 \cdots I_N) \times J}$ *is*

$$\bar{\mathcal{Y}}_{(N+1)}\Big(\overline{i_1 i_2 i_3 \dots i_N}, j\Big) \;=\; \Big[\mathbf{x}_1, \dots, \mathbf{x}_J\Big] \tag{3.22}$$

where $\mathbf{x}_j \in \mathbb{R}^{I_1 I_2 \cdots I_N}$ *denotes the vectorization of the tensor* $\bar{\mathcal{X}}_j$*, obtained by stacking all its entries into a single column vector in Little-Endian order consistent with* (3.19)*.*

We next define the mode-n product for tensors.

Definition 12. *The* mode-n product *takes as input an order-N tensor,* $\bar{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$*, and a matrix* $\mathbf{A} \in \mathbb{R}^{J \times I_n}$*, to produce another tensor,* $\bar{\mathcal{Y}}$*, of the same order as the original tensor* $\bar{\mathcal{X}}$*. The operation is denoted by*

$$\bar{\mathcal{Y}} = \bar{\mathcal{X}} \times_n \mathbf{A} \tag{3.23}$$

where $\bar{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$. *The mode-n product is comprised of 3 consecutive steps:*

$$\bar{\mathcal{X}} \to \bar{\mathcal{X}}_{(n)}\,, \quad \bar{\mathcal{Y}}_{(n)} = \mathcal{A}\bar{\mathcal{X}}_{(n)}\,, \quad \bar{\mathcal{Y}}_{(n)} \to \bar{\mathcal{Y}}\ . \tag{3.24}$$

To see the transition to the sparse tensor factorization, we next introduce the tensor contraction product.

Definition 13. *([153])* Tensor Contraction Product *(TCP) is at the core of tensor decompositions, an operation similar to the mode-n product, but the arguments of which are multidimensional arrays that can be of a different order. For instance, given an N-th order tensor* $\bar{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ *and an M-th order tensor* $\bar{\mathcal{Y}} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, *with common modes* $I_n = J_m$, *then, their* (n, m)*-contraction denoted by* $\times_n^m$, *yields a third tensor* $\bar{\mathcal{Z}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times \cdots \times J_M}$ *of order* $(N + M - 2)$, $\bar{\mathcal{Z}} = \bar{\mathcal{X}} \times_n^m \bar{\mathcal{Y}}$ *with the entries*

$$\begin{aligned}\bar{\mathcal{Z}}(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N, j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_M) \\ = \sum_{i_n \in [I_n]} \bar{\mathcal{X}}(i_1, \ldots, i_{n-1}, i_n, i_{n+1}, \ldots, i_N) \\ \times \bar{\mathcal{Y}}(j_1, \ldots, j_{m-1}, i_n, j_{m+1}, \ldots, j_M)\,.\end{aligned} \tag{3.25}$$

The overwhelming indexing associated with the TCP operation in (3.25) becomes unmanageable for larger tensor networks, whereby multiple TCPs are carried out across a large number of tensors. Manipulation of such expressions is prone to errors and prohibitive to the manipulation of higher-order tensors.

We next generalize the TCP operation for the cases where there is more than one common mode between two tensors. Recall that this operation is used in (3.32).

Definition 14. Generalized TCP *is an operation similar to TCP, but it contracts a whole* ordered block of modes *(a multi-index) between two tensors. For instance, given an N-th order tensor* $\bar{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ *and an M-th order tensor* $\bar{\mathcal{Y}} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, *assume the tails match pairwise, i.e.,* $(I_n, \ldots, I_N) = (J_m, \ldots, J_M)$ *and* $N - n + 1 = M - m + 1$. *Then, their* $([[n : N]], [[m : M]])$*-contraction, denoted by* $\times_{[[n:N]]}^{[[m:M]]}$, *yields a third tensor* $\bar{\mathcal{Z}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J_1 \times \cdots \times J_{m-1}}$, *where*

$$\bar{\mathcal{Z}} \;=\; \bar{\mathcal{X}} \times_{[[n:N]]}^{[[m:M]]} \bar{\mathcal{Y}}$$

of order $(n + m - 2)$, *with entries*

$$\bar{\mathcal{Z}}(i_1,\ldots,i_{n-1},\,j_1,\ldots,j_{m-1}) = \sum_{\underline{\mathbf{i}}\in\mathcal{I}} \bar{\mathcal{X}}(i_1,\ldots,i_{n-1},\,i_n,\ldots,i_N) \times \bar{\mathcal{Y}}(j_1,\ldots,j_{m-1},\,i_n,\ldots,i_N) \quad (3.26)$$

where $\mathcal{N} \triangleq [[n:N]]$ *denotes the range of contracted modes,* $\underline{\mathbf{i}} \triangleq (i_n)_{n\in\mathcal{N}} = (i_n,\ldots,i_N)$ *refers to their corresponding indices, and* $\mathcal{I} \triangleq \prod_{n\in\mathcal{N}}[I_n]$. *The sum ranges over all* $\underline{\mathbf{i}} \in \mathcal{I}$ *and contracts* $N-n+1$ *paired modes (from* n *to* N*), resulting in an order-*$(n+m-2)$ *tensor* $\bar{\mathcal{Z}}$.

Our next step is to formulate the problem in tensor and matrix forms, which allows us to proceed to obtain the results accordingly.

3.4 Problem Formulation in Tensor Form

In the $(K,N,L,\Gamma,\Delta,\{P_\ell,\Lambda_\ell\}_{\ell\in[L]})$ framework, the desired functions in (3.4) are fully represented by a tensor $\bar{\mathcal{F}} \in \mathbb{R}^{K\times P_1\times\ldots\times P_L}$ of all function coefficients $\{c_{k,\underline{\mathbf{p}}}\}$ across all the users, which is an order-$(L+1)$ tensor, i.e., a multi-way array with $L+1$ modes. With $\bar{\mathcal{F}}$ in place, we must decide on the computation assignment (encoding) and the communication protocol (decoding). For the error-free case, from [151], this task is equivalent — directly from (3.11),(3.12)— to solving a sparse tensor factorization problem subject to the sparsity constraints Γ,Δ, and $\{\Lambda_\ell\}_{\ell=1}^{L}$, as specified in (3.6), (3.13), and (3.10), respectively. This problem takes the form

$$\bar{\mathcal{F}} = \bar{\mathcal{E}} \times_1 \mathbf{D} \quad (3.27)$$

where $\bar{\mathcal{E}} \in \mathbb{R}^{N\times P_1\times\ldots\times P_L}$ is the computing tensor, capturing the non-linear encoding tasks of servers that holds the coefficients $e_{n,\underline{\mathbf{p}}}$ from (3.11), $\mathbf{D} \in \mathbb{R}^{K\times N}$ is the communication matrix derived from the decoding coefficients $d_{k,n}$ in (3.12), capturing the communication and linear decoding task done by each user, and finally, the operation $\times_1$ denotes the mode-1 product[6] of $\bar{\mathcal{E}}$ and $\mathbf{D}$ and is comprised of three consecutive operations:

$$\bar{\mathcal{E}} \to \bar{\mathcal{E}}_{(1)}\,,\ \bar{\mathcal{F}}_{(1)} = \mathbf{D}\bar{\mathcal{E}}_{(1)}\,,\ \bar{\mathcal{F}}_{(1)} \to \bar{\mathcal{F}} \quad (3.28)$$

where the matrix $\bar{\mathcal{E}}_{(1)} \in \mathbb{R}^{N\times P_1P_2\ldots P_L}$ represents the mode-1 unfolding of the encoding tensor $\bar{\mathcal{E}} \in \mathbb{R}^{N\times P_1\times\ldots\times P_L}$.

[6]The mode-n product is defined similarly to (3.28) with respect to the mode-n unfolding of the tensor (cf. Definition 9 in Section 3.3).

To motivate the underlying idea behind the tensor construction, we next form tensors of desired function outputs, transmissions, and retrieved function outputs, as detailed below.

a) **Desired Function Outputs in Vector Form**.

We exploit the representation in (3.4), and consider the following vector of desired function outputs

$$\mathbf{f} \triangleq [F_1, F_2, \ldots, F_K]^\intercal \in \mathbb{R}^K \tag{3.29}$$

and for all $\underline{\mathbf{p}} \in \prod_{\ell\in[L]}[P_\ell]$, the tensor of basis coefficients

$$\bar{\mathcal{F}}_k(\underline{\mathbf{p}}) \triangleq c_{k,\underline{\mathbf{p}}}\,,\ \bar{\mathcal{F}}_k \in \mathbb{R}^{P_1\times P_2\times\ldots\times P_L},\ k\in[K] \tag{3.30}$$

as well as the tensor of multiplicative terms of $\{W_\ell\}_{\ell\in[L]}$

$$\bar{\mathcal{W}}(\underline{\mathbf{p}}) \triangleq \prod\nolimits_{\ell\in[L]} W_\ell^{p_\ell-1},\ \bar{\mathcal{W}} \in \mathbb{R}^{P_1\times P_2\times\ldots\times P_L}\ . \tag{3.31}$$

Next, using (3.29)-(3.31) leads us to the vector of function outputs

$$\mathbf{f} = \mathit{stack}_1(\bar{\mathcal{F}}_1,\ldots,\bar{\mathcal{F}}_K) \times_{[[2:L+1]]}^{[[1:L]]} \bar{\mathcal{W}} \tag{3.32}$$

where $\mathit{stack}_1(\bar{\mathcal{F}}_1,\ldots,\bar{\mathcal{F}}_K)$ forms an order-$(L+1)$ tensor $\bar{\mathcal{F}} \in \mathbb{R}^{K\times P_1\times\ldots\times P_L}$ composed by stacking the K tensors $\{\bar{\mathcal{F}}_k\}_{k\in[K]}$ (each of order L) along a new first mode. In the above, the operation $\times_{[[2:L+1]]}^{[[1:L]]}$ simply generalizes the tensor contraction product[7] (cf. [151]) by capturing an ordered set of common modes between two given tensors $\bar{\mathcal{F}} \in \mathbb{R}^{K\times P_1\times P_2\cdots\times P_L}$ and $\bar{\mathcal{W}} \in \mathbb{R}^{P_1\times P_2\cdots\times P_L}$. This generalized tensor contraction product yields a lower order tensor. Particularly, the $\times_{[[2:L+1]]}^{[[1:L]]}$-contraction product of $\bar{\mathcal{F}}$ and $\bar{\mathcal{W}}$ is obtained by contracting modes $[[2:L+1]]$ in $\bar{\mathcal{F}}$ and $[[1:L]]$ in $\bar{\mathcal{W}}$, corresponding to the indices $\underline{\mathbf{p}} \in \prod_{\ell\in[L]}[P_\ell]$, yielding an order-1 tensor (i.e., a vector) $\mathbf{f} \in \mathbb{R}^K$ with entries

$$F_k = \sum_{\underline{\mathbf{p}}\in\prod_{\ell\in[L]}[P_\ell]} \bar{\mathcal{F}}(k,\underline{\mathbf{p}})\,\bar{\mathcal{W}}(\underline{\mathbf{p}})\,,\quad k\in[K]\,. \tag{3.33}$$

b) **Transmissions in Vector Form**.

[7]Tensor contraction product operation extends matrix multiplication to higher-order tensors by summing over shared modes.

In the communication phase, similarly to above, from (3.11), the transmission from server n takes the following form $z_n = \bar{\mathcal{E}}_n \times_{[[1:L]]}^{[[1:L]]} \bar{\mathcal{W}} \in \mathbb{R}$, thus yielding the overall transmission vector

$$\mathbf{z} \triangleq [z_1, z_2, \ldots, z_N] = \bar{\mathcal{E}} \times_{[[2:L+1]]}^{[[1:L]]} \bar{\mathcal{W}}\,, \quad \mathbf{z} \in \mathbb{R}^N \tag{3.34}$$

where for all $\underline{\mathbf{p}} \in \prod_{\ell\in[L]}[P_\ell]$, the encoding coefficients $e_{n,\underline{\mathbf{p}}}$ from (3.11) satisfy

$$\bar{\mathcal{E}}_n(\underline{\mathbf{p}}) \triangleq e_{n,\underline{\mathbf{p}}},\ \bar{\mathcal{E}}_n \in \mathbb{R}^{P_1\times P_2\times\ldots\times P_L}\,. \tag{3.35}$$

c) **Retrieved Function Outputs in Vector Form**.

In the decoding phase, from (3.12), each retrieved function output takes the form $F'_k = \mathbf{d}_k^\intercal \mathbf{z} \in \mathbb{R}$, thus resulting in the vector of all outputs taking the form

$$\mathbf{f}' \triangleq [F'_1, F'_2, \ldots, F'_K] = [\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_K]^\intercal \mathbf{z} \in \mathbb{R}^K \tag{3.36}$$

where the decoding coefficients $d_{k,n}$ from (3.12) satisfy

$$\mathbf{d}_k \triangleq [d_{k,1}, d_{k,2}, \ldots, d_{k,N}]^\intercal \in \mathbb{R}^N,\ k \in [K]\,. \tag{3.37}$$

The tensors of basis coefficients in (3.30), the tensors of encoding coefficients (3.35), and the vector of decoding coefficients in (3.37) allow us to form the respective tensors

$$\bar{\mathcal{F}} \triangleq \mathit{stack}_1(\bar{\mathcal{F}}_1, \ldots, \bar{\mathcal{F}}_K) \in \mathbb{R}^{K\times P_1\times P_2\times\ldots\times P_L}, \tag{3.38}$$

$$\bar{\mathcal{E}} \triangleq \mathit{stack}_1(\bar{\mathcal{E}}_1, \ldots, \bar{\mathcal{E}}_N) \in \mathbb{R}^{N\times P_1\times P_2\times\ldots\times P_L}, \tag{3.39}$$

$$\mathbf{D} \triangleq [\mathbf{d}_1, \ldots, \mathbf{d}_K]^\intercal \in \mathbb{R}^{K\times N}\,. \tag{3.40}$$

We are now ready to establish the connection between the tensor forms in (3.38), (3.39), and (3.40) and the tensor factorization problem. We note that employing the definitions of $\mathbf{f}$ and $\mathbf{f}'$ from (3.32) and (3.36), respectively, and setting the recovery error to zero, i.e., $\|\mathbf{f}' - \mathbf{f}\|^2 = 0$, we see that resolving our distributed computing problem requires that $\bar{\mathcal{F}}$ be decomposed as (3.27).

In terms of the corresponding connection to the sparsity of $\mathbf{D}$ and $\bar{\mathcal{E}}$, we recall from (3.6), our metric Γ, which directly from (3.35) and (3.39), implies the computation constraint as

$$\max_{n\in[N]} \sum_{\ell\in[L]} \left| \mathbb{1}\Big(\mathrm{supp}\big(\bar{\mathcal{E}}(n, \underbrace{:, \ldots, :}_{\ell-1 \text{ terms}}, p_\ell, \underbrace{:, \ldots, :}_{L-\ell \text{ terms}}) \big) \neq \emptyset \Big) \right| \leq \Gamma\,, \quad p_\ell \in [2 : P_\ell]$$

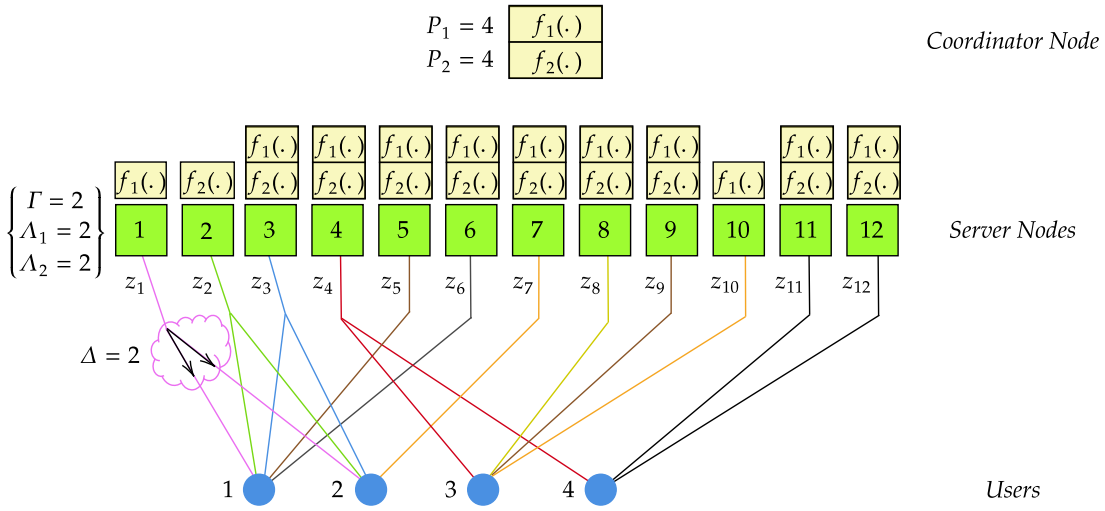

Figure 3.2: Corresponding to Example 3, the distributed computing system is illustrated for a set of system parameters, including computation cost $\Gamma = 2$, communication cost $\Delta = 2$, and multiplication costs $\Lambda_1 = \Lambda_2 = 2$ for distributed computation of multivariate polynomial demands with the maximum degree ranges $P_1 = P_2 = 4$.

Furthermore, from (3.13), we recall Δ, which from (3.37) and (3.40), implies a communication constraint

$$\max_{n\in[N]} \left|\text{supp}\Big(\mathbf{D}(:,n)\Big)\right| \leq \Delta \ .$$

Finally, from (3.10), we recall Λ_ℓ, which from (3.35) and (3.39), suggests, for all $\ell \in [L]$, $p_\ell \in [P_\ell]$, the multiplication constraints

$$\|\bar{\mathcal{E}}(n, p_1, p_2, \ldots, p_{\ell-1}, :, p_{\ell+1}, \ldots, p_L)\|_0 \leq \Lambda_\ell \ .$$

We next explain in detail a full example, motivating the proposed tensor factorization approach by the exact recovery of the user demands.

Example 3. *Let us consider the system model depicted in Figure 3.2, where we have $N = 12$ servers, $K = 4$ users, $L = 2$ basis subfunctions whose maximum range of degrees are specified by $P_1 = P_2 = 4$. Let us assume each server is able to compute up to $\Gamma = 2$ basis subfunctions, $\Lambda_1 = \Lambda_2 = 2$ number of multiplications, and transmit to up to $\Delta = 2$ users in one shot $T = 1$.*

Consider that in the demand phase, each user requests to compute a linear combination of the multiplicative terms of output files in $\mathbf{W} = (W_1, W_2)$, corresponding to

$$F_1(\mathbf{W}) = 2W_1 + 4W_2 + 2W_1W_2 + 2W_1^3W_2 + W_1^3W_2^2 \ ,$$

$$
\begin{aligned}
F_2(\mathbf{W}) &= 2W_1W_2 + W_1 + W_2 + 2W_1^2W_2^2 + 5W_1^2W_2^3 + W_1^3W_2^3\,,\\
F_3(\mathbf{W}) &= 2W_1W_2 + 4W_1W_2^2 + 6W_1^2W_2^2 + 5W_1^3\,,\\
F_4(\mathbf{W}) &= 3W_1W_2 + 2W_1W_2^2 + 5W_1^3W_2^3
\end{aligned} \tag{3.41}
$$

where each user demand can be captured by the Frobenius inner product of two matrices $\mathbf{F}_k$ *and* $\mathbf{M}$ *as*

$$
F_k(\mathbf{W}) = \langle \mathbf{F}_k, \mathbf{M} \rangle_{\mathrm{F}} = \sum_{p_1 \in [P_1]} \sum_{p_2 \in [P_2]} \mathbf{F}_k(p_2, p_1)\, \mathbf{M}(p_2, p_1),\ k \in [4]\,,
$$

where

$$
\mathbf{M}(\mathbf{W}) = \begin{bmatrix} 1 & W_1 & W_1^2 & W_1^3 \\ W_2 & W_1W_2 & W_1^2W_2 & W_1^3W_2 \\ W_2^2 & W_1W_2^2 & W_1^2W_2^2 & W_1^3W_2^2 \\ W_2^3 & W_1W_2^3 & W_1^2W_2^3 & W_1^3W_2^3 \end{bmatrix}, \quad \mathbf{M} \in \mathbb{R}^{P_2 \times P_1}
$$

captures the computational tasks, whose elements are the multiplicative terms of the output files W_1 *and* W_2*. Furthermore, the coefficients in* (3.41) *are described in matrix form as*

$$
\mathbf{F}_1 = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 4 & 2 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},\ \mathbf{F}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 5 & 1 \end{bmatrix},
$$

$$
\mathbf{F}_3 = \begin{bmatrix} 0 & 0 & 0 & 5 \\ 0 & 2 & 0 & 0 \\ 0 & 4 & 6 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\ \mathbf{F}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}
$$

where $\mathbf{F}_k = \bar{\mathcal{F}}(k, :, :) \in \mathbb{R}^{P_2 \times P_1}$ *is indeed a subtensor (here matrix) of the entire demand tensor* $\bar{\mathcal{F}} \in \mathbb{R}^{K \times P_2 \times P_1}$*, which captures all coefficients in all users' demands in* (3.41) *with the first and second dimension reflecting* W_2 *and* W_1*, respectively, as illustrated in Figure 3.3.*

In the computation phase, the coordinator allocates the set of basis subfunctions $\mathcal{S}_n$ *to server* $n = 1, \ldots, 12$*, determined as*

$$
\mathcal{S}_1 = \{1\},\ \mathcal{S}_2 = \{2\},\ \mathcal{S}_{10} = \{1\},\ \mathcal{S}_{[3:13]\setminus\{10\}} = \{1, 2\}
$$

along with the set $\mathcal{P}_n$ *of index tuples* $\underline{\mathbf{p}} = (p_1, p_2)$ *corresponding to the multiplicative terms* $W_1^{p_1-1}W_2^{p_2-1}$ *to be computed by server* $n = 1, \ldots, 12$*, which is described as*

$$
\mathcal{P}_1 = \{(2,1)\},\ \mathcal{P}_2 = \{(1,2)\},\ \mathcal{P}_3 = \mathcal{P}_4 = \{(2,2)\},\ \mathcal{P}_5 = \{(4,2)\},\ \mathcal{P}_6 = \{(4,3)\},
$$

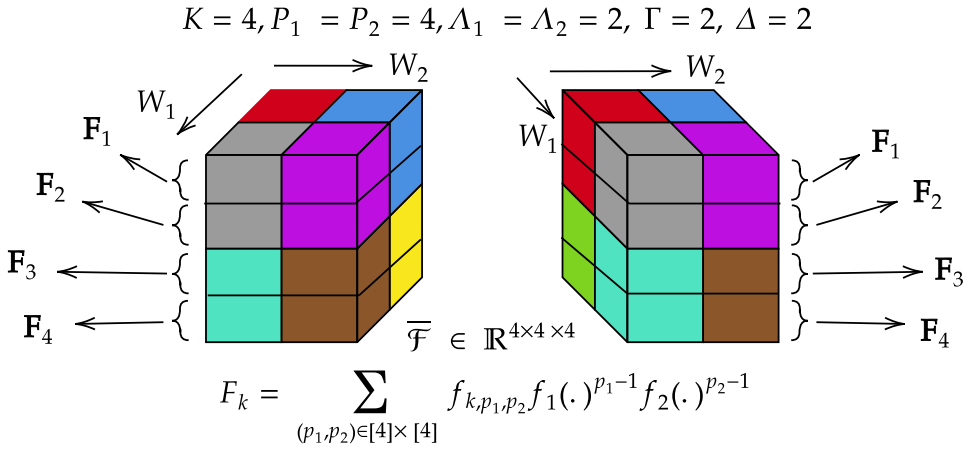

Figure 3.3: Corresponding to Example 3, the tensor $\bar{\mathcal{F}}$ forming all user demands is illustrated here.

$\mathcal{P}_7 = \{(3,3), (3,4), (4,4)\}$, $\mathcal{P}_8 = \{(2,3)\}$, $\mathcal{P}_9 = \{(3,3)\}$, $\mathcal{P}_{10} = \{(4,1)\}$, $\mathcal{P}_{11} = \{(2,3)\}$, $\mathcal{P}_{12} = \{(4,4)\}$.

Next, the coordinator node asks server $n = 1, \ldots, 12$ *to compute* z_n *according to*

$$z_1 = 2W_1\,, \quad z_2 = 4W_2\,, \quad z_3 = W_1W_2\,, \quad z_4 = W_1W_2\,, \quad z_5 = 2W_1^3W_2\,,$$
$$z_6 = W_1^3W_2^2\,, \quad z_7 = 2W_1^2W_2^2 + 5W_1^2W_2^3 + W_1^3W_2^3\,, \quad z_8 = 4W_1W_2^2\,,$$
$$z_9 = 6W_1^2W_2^2\,, \quad z_{10} = 5W_1^3\,, \quad z_{11} = 2W_1W_2^2\,, \quad z_{12} = 5W_1^3W_2^3$$

where

$$z_n = \langle \mathbf{E}_n, \mathbf{M} \rangle_{\mathrm{F}}$$

is the computational output that server n *evaluates, and the encoding coefficients above are described in the matrix form with the non-zero entries*

$$\mathbf{E}_1(1,2) = 2\,, \quad \mathbf{E}_2(2,1) = 4\,, \quad \mathbf{E}_3(2,2) = 1\,, \quad \mathbf{E}_4(2,2) = 1\,, \quad \mathbf{E}_5(2,4) = 2\,,$$
$$\mathbf{E}_6(3,4) = 1\,, \quad \mathbf{E}_7(3,3) = 2\,, \quad \mathbf{E}_7(4,3) = 5\,, \quad \mathbf{E}_7(4,4) = 1\,, \quad \mathbf{E}_8(3,2) = 4\,,$$
$$\mathbf{E}_9(3,3) = 6\,, \quad \mathbf{E}_{10}(1,4) = 5\,, \quad \mathbf{E}_{11}(3,2) = 2\,, \quad \mathbf{E}_{12}(4,4) = 5$$

where $\mathbf{E}_n = \bar{\mathcal{E}}(n,:,:) \in \mathbb{R}^{P_2\times P_1}$ *denotes the encoding subtensor (here matrix) of the entire encoding tensor* $\bar{\mathcal{E}}$*, which captures the assignment of computational tasks to all the servers* $n = 1, \ldots, 12$*, and* $\mathbf{E}_n(p_2, p_1)$ *demonstrates the* (p_2, p_1)*-th entry of the encoding matrix* $\mathbf{E}_n$*, where* $p_2 \in [P_2]$*,* $p_1 \in [P_1]$*.*

In the communication phase, the coordinator requests server n to send z_n *to the users in* $\mathcal{T}_n$*, where* $\mathcal{T}_1 = \{1\}$, $\mathcal{T}_2 = \{1\}$, $\mathcal{T}_3 = \{1, 2\}$, $\mathcal{T}_4 = \{3, 4\}$, $\mathcal{T}_5 = \{1\}$, $\mathcal{T}_6 = \{2\}, \mathcal{T}_7 = \{2\}, \mathcal{T}_8 = \{2\}, \mathcal{T}_9 = \{3\}$, $\mathcal{T}_{10} = \{3\}, \mathcal{T}_{11} = \{4\}, \mathcal{T}_{12} = \{4\}$.

Each user is then going to decode by linearly combining z_n*'s according to*

$$\begin{aligned} F_1' =& z_1 + z_2 + 2z_3 + z_5 + z_6\,, \quad F_2' = 1/2z_1 + 1/4z_2 + 2z_3 + z_7\,, \\ F_3' =& 2z_4 + z_8 + z_9 + z_{10}\,, \quad F_4' = 3z_4 + z_{11} + z_{12}\,. \end{aligned}$$

To decode, each user $k = 1, 2, 3, 4$ *utilises a linear decoding algorithm according to the elements on the* k*-th row of the decoding matrix* $\mathbf{D}$*, where*

$$\mathbf{D} \triangleq \begin{bmatrix} 1 & 1 & 2 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/4 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

It can be easily verified that $F_k = F_k'$, $k = 1, 2, 3, 4$*, confirming the lossless reconstruction of the demands of all users.*

As illustrated above, the design of the task assignment sets $\mathcal{S}_n$, $\mathcal{P}_n$ to the servers, which governs the sparsity of encoding tensor $\bar{\mathcal{E}}$, and the design of connectivity sets $\mathcal{T}_n$, which governs the sparsity of decoding matrix $\mathbf{D}$ is central to the proposed distributed computing framework. Our goal is to guarantee lossless recovery of all user demands under sparsity constraints while requiring significantly fewer servers than existing solutions [105].

Before proceeding with the achievable scheme construction, we provide a brief review of the fundamental concepts related to SVD decompositions for high-dimensional data, which will be useful for the proofs of Theorems 3 and 4, presented in Section 3.6.1 and Section 3.6.2, respectively.

3.5 A Primer on Multilinear SVD

To prove our main results (cf. Theorems 3 and 4), we need the notion of multilinear SVDs. The multilinear SVD (MLSVD) extends the concept of the matrix SVD into the multilinear domain [151]. This decomposition provides a powerful tool for analyzing tensors and obtaining low-rank multilinear approximations.

For matrices, the SVD is well-known and expressed as

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\intercal} \tag{3.42}$$

where $\mathbf{M} \in \mathbb{R}^{J_1 \times J_2}$ is an arbitrary real-valued matrix, $\mathbf{\Sigma} \in \mathbb{R}^{I_1 \times I_2}$ is a diagonal matrix with the entries $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(I_1,I_2)} \geq 0$ in descending order, and $\mathbf{U} \in \mathbb{R}^{J_1 \times I_1}$ and $\mathbf{V} \in \mathbb{R}^{J_2 \times I_2}$ are orthogonal matrices.

Using mode-n tensor-matrix products, we have:

$$\mathbf{M} = \mathbf{\Sigma} \times_1 \mathbf{U} \times_2 \mathbf{V}^\intercal . \tag{3.43}$$

The MLSVD generalizes this decomposition to higher-order tensors. In the literature, e.g., [151], it is also referred to as the higher-order SVD or Tucker decomposition, though "Tucker decomposition" has evolved into a broader term. The MLSVD of a N-th order tensor is represented as

$$\bar{\mathcal{T}} = \bar{\mathcal{S}} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \ldots \times_N \mathbf{U}^{(N)} \tag{3.44}$$

where $\bar{\mathcal{T}} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_N}$, $\bar{\mathcal{S}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, and $\mathbf{U}^{(n)} \in \mathbb{R}^{J_n \times I_n}$, $n \in [N]$. Similar to the matrix case, where $\mathbf{U}$ and $\mathbf{V}$ serve as orthonormal bases for the column and row spaces, the MLSVD computes N orthonormal mode matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times J_n}$, $n \in [N]$, each spanning the subspace of mode-n vectors. These mode matrices are directly analogous to the singular vector bases in the matrix SVD. Concretely, the MLSVD unfolds the tensor along each mode, applies the matrix SVD to the resulting unfolding, and collects the mode-n singular vectors to form the factor matrices, yielding an orthogonal decomposition of the tensor into a core tensor and its mode matrices. The subtensors $\bar{\mathcal{S}}_{i_n=\alpha}$ of $\bar{\mathcal{S}}$ are obtained by fixing the n-th index to α with the following properties:

- *All-orthogonality* [8]: Subtensors $\bar{\mathcal{S}}_{i_n=\alpha}$ and $\bar{\mathcal{S}}_{i_n=\beta}$ are orthogonal for all possible n, α, β if

$$\langle \bar{\mathcal{S}}_{i_n=\alpha}, \bar{\mathcal{S}}_{i_n=\beta} \rangle = 0 \quad \text{when} \quad \alpha \neq \beta \tag{3.45}$$

  where the scalar product of two tensors $\bar{\mathcal{T}}, \bar{\mathcal{S}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ is defined as

$$\langle \bar{\mathcal{T}}, \bar{\mathcal{S}} \rangle \triangleq \sum_{i_1} \ldots \sum_{i_N} t_{i_1 \ldots i_N} s_{i_1 \ldots i_N} \tag{3.46}$$

  where t and s represent the elements of tensors $\bar{\mathcal{T}}$ and $\bar{\mathcal{S}}$, respectively.

- *Ordering*:

$$\sigma_1^{(n)} \geq \sigma_2^{(n)} \geq \ldots \geq \sigma_{I_n}^{(n)} \geq 0 \tag{3.47}$$

[8] Arrays with a scalar product of zero are considered orthogonal.

where symbols $\sigma_i^{(n)}$ represent the n-mode singular values of $\bar{\mathcal{S}}$, and are equal to the Frobenius-norms $\|\bar{\mathcal{S}}_{i_n=i}\|$, $i \in [I_n]$, where the Frobenius-norm of tensor $\bar{\mathcal{T}}$ is described by

$$\|\bar{\mathcal{T}}\| \triangleq \langle \bar{\mathcal{T}}, \bar{\mathcal{T}} \rangle \,. \tag{3.48}$$

The mode-n rank (or n-rank) of a tensor is defined using the matrix-based methods. Specifically, it is the rank of the mode-n unfolding of the tensor, i.e., the dimension of the subspace spanned by its mode-n vectors. The mode-n vectors of $\bar{\mathcal{T}}$ are precisely the column vectors of its mode-n matrix unfolding $\bar{\mathcal{T}}_{(n)}$. We thus have:

$$\text{rank}_n(\bar{\mathcal{T}}) = \text{rank}(\bar{\mathcal{T}}_{(n)}) \,. \tag{3.49}$$

We proceed to present the main results in the next section.

3.6 Main Results

In this section, we present the main findings of this chapter, which are summarized as follows.

First in Section 3.6.1, we devise a sparse tensor factorization scheme for the proposed lossless $(K, N, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell \in [L]})$ distributed computing setting, by upper bounding the required number of servers N, which is the common dimension between $\bar{\mathcal{E}}$, $\mathbf{D}$, directly impacting our sparse tensor factorization approach (cf. Theorem 3). We then examine Theorem 3 numerically to illustrate the intuition and the performance of the proposed scheme (cf. Example 4). Next, we compare the proposed sparse tensor factorization approach with the sparse matrix factorization scheme of [105] in terms of the total computation cost (cf. Proposition 1).

We then present a tighter achievable rate result in Section 3.6.2 by leveraging a combinatorial reduction algorithm (cf. Algorithm 1 in Theorem 4). Next, we give a numerical elaboration and illustration of the proposed Algorithm 1 (cf. Example 5). Finally, we conclude this section by analyzing the complexity of the proposed Algorithm 1 (cf. Remark 5). All the results hold without any restriction on the dimensions, provided that each subtensor of $\bar{\mathcal{F}}$ has full rank, a condition that is easily justified in our real-valued function settings.

3.6.1 Achievable System Rate

We now characterize the achievable system rate for lossless distributed computation in a $\left(K, N, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell \in [L]}\right)$ distributed system. We focus on

the homogeneous setting in which

$$P_\ell = P \qquad \text{and} \qquad \Lambda_\ell = \Lambda, \qquad \forall\, \ell \in [L],$$

and assume the divisibility conditions

$$\Delta \mid K \qquad \text{and} \qquad \Lambda \mid P.$$

These assumptions simplify the construction by allowing the users and exponent indices to be partitioned into groups of equal size.

Before presenting the achievable scheme, we clarify how a bound on the required number of servers translates into an achievable rate result. Let

$$N^\star \triangleq \min \left\{ N : \begin{array}{l} \text{there exists a lossless distributed computing scheme} \\ \text{satisfying the prescribed computation, communication,} \\ \text{and multiplication constraints} \end{array} \right\}$$

denote the minimum number of servers required by the system. Since the system rate is defined as

$$R = \frac{K}{N},$$

the largest achievable rate is

$$R^\star = \frac{K}{N^\star}.$$

Suppose that a particular construction achieves lossless recovery using N_{ach} servers. This establishes the upper bound

$$N^\star \leq N_{\text{ach}}.$$

Because K/N is decreasing in N, it follows that

$$R^\star = \frac{K}{N^\star} \geq \frac{K}{N_{\text{ach}}}.$$

Therefore, an upper bound on the minimum required number of servers yields a lower bound on the maximum achievable system rate. Equivalently, the rate

$$R_{\text{ach}} \triangleq \frac{K}{N_{\text{ach}}}$$

is achievable through the proposed construction. The following result specifies such a construction and the corresponding achievable rate.

Theorem 3 (Achievable number of servers and system rate)**.** *Consider the lossless* $\left(K, N, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell\in[L]}\right)$ *distributed computing system under*

$$P_\ell = P \qquad \text{and} \qquad \Lambda_\ell = \Lambda, \qquad \forall\, \ell \in [L],$$

with $\Delta \mid K$ *and* $\Lambda \mid P$*. There exists a lossless distributed computing scheme using* N_{ach} *servers, where*

$$N^\star \leq N_{\text{ach}} \triangleq \frac{K}{\Delta}\binom{L}{\Gamma}\min\left\{\Delta, \Lambda^\Gamma\right\}\left(\frac{P}{\Lambda}\right)^\Gamma. \tag{3.50}$$

Consequently, the corresponding rate

$$R_{\text{ach}} \triangleq \frac{K}{N_{\text{ach}}} \tag{3.51}$$

is achievable. Therefore, the maximum achievable system rate $R^\star$ *satisfies*

$$\begin{aligned} R^\star \geq R_{\text{ach}} &= \frac{K}{N_{\text{ach}}} \\ &= \frac{\Delta}{\binom{L}{\Gamma}\min\left\{\Delta, \Lambda^\Gamma\right\}\left(\frac{P}{\Lambda}\right)^\Gamma}. \end{aligned} \tag{3.52}$$

Proof. This proof has two parts. We first, in Part I, define the n-th rank-one contribution support, representative rank-one support, or tile, along with their related parameters. These tiles represent equivalence classes of rank-one contribution supports, which show how any support constraint on $\mathbf{D}$ and $\bar{\mathcal{E}}$ contributes to the supports on $\bar{\mathcal{E}} \times_1 \mathbf{D}$. The correspondence above is shown via Lemma 3.

Then in Part II, to design the decoding matrix $\mathbf{D}$ and encoding tensor $\bar{\mathcal{E}}$ for lossless reconstruction of the demand tensor $\bar{\mathcal{F}}$, the following steps are involved.

1. Designing the sizes of the tiles for the product tensor $\bar{\mathcal{F}} = \bar{\mathcal{E}} \times_1 \mathbf{D}$.
2. Creating and filling the non-zero elements of the tiles in the product tensor $\bar{\mathcal{F}}$.
3. Placing the filled tiles in $\mathbf{D}$ and in $\bar{\mathcal{E}}$, accordingly.
4. The number of servers required is obtained by associating the rank of each tile with the servers and then summing the ranks over all tiles [9].

[9] Quickly recall that for a $(L+1)$- dimensional tensor $\bar{\mathcal{E}}$, then $\bar{\mathcal{E}}(n, :, \ldots, :)$ represents a L dimensional subtensor and $\mathbf{D}(:, n)$ for a matrix $\mathbf{D}$ is its n-th column, and $\text{Supp}(\mathbf{D})$ ($\text{Supp}(\bar{\mathcal{E}})$) is a binary matrix (tensor), indicating the support of $\mathbf{D}$ ($\bar{\mathcal{E}}$). Also, when we refer to a support constraint, this will be in the form of a binary matrix (tensor) that indicates the support (the position of the allowed non-zero elements) of a matrix (tensor) of interest.

Throughout the proof, the design borrows concepts and definitions from tensor theory in Section 3.3, and the blockwise SVD approach of [150], generalized to the multilinear scenario in [151] by utilizing tensors, as detailed in Section 3.5.

We now proceed with Part I of the proof for Theorem 3.

Part I (Basic Concepts and Definitions). We here present the basic concepts and preliminaries of our proposed scheme. We first introduce the concept of *rank-one contribution support*, which is essential in our tile design procedure by capturing the constitutive components of each tile.

Definition 15. *Given two support constraints* $\mathbf{I} \in \{0,1\}^{K\times N}$ *and* $\bar{\mathcal{J}} \in \{0,1\}^{N\times P_1\times P_2\times\ldots\times P_L}$*, then for any* $n \in [N]$*, we refer to*

$$\bar{\mathcal{S}}_n(\mathbf{I},\bar{\mathcal{J}}) \triangleq \bar{\mathcal{J}}(n,:,\ldots,:)\times_1 \mathbf{I}(:,n) \tag{3.53}$$

as the n-th rank-one contribution support*.*

We note that when the supports are implied, we may shorten $\bar{\mathcal{S}}_n(\mathbf{I},\bar{\mathcal{J}})$ to just $\bar{\mathcal{S}}_n$. The aforementioned $\mathbf{I}$ and $\bar{\mathcal{J}}$ will generally represent the support of $\mathbf{D}$ and $\bar{\mathcal{E}}$ respectively, while $\bar{\mathcal{S}}_n(\mathbf{I},\bar{\mathcal{J}})$ will generally capture some of the support of $\bar{\mathcal{E}}\times_1\mathbf{D}$ and thus of $\bar{\mathcal{F}}$. We have the following lemma for this.

Lemma 3. *For* $\mathbf{I} \triangleq supp(\mathbf{D})$ *and* $\bar{\mathcal{J}} \triangleq supp(\bar{\mathcal{E}})$*, then*

$$\begin{aligned}\cup_{n=1}^{N}\bar{\mathcal{S}}_n(\mathbf{I},\bar{\mathcal{J}}) &= \cup_{n=1}^{N}\bar{\mathcal{J}}(n,:,\ldots,:)\times_1\mathbf{I}(:,n)\\ &\supseteq Supp(\bar{\mathcal{E}}\times_1\mathbf{D})\,.\end{aligned} \tag{3.54}$$

Proof of Lemma 3 . The above follows from Definition 15 and

$$\bar{\mathcal{E}}\times_1\mathbf{D} = \sum_{n=1}^{N}\bar{\mathcal{E}}(n,:,\ldots,:)\times_1\mathbf{D}(:,n)\,. \qquad \square$$

We continue with the proof of Theorem 3 by introducing the concept of *equivalence classes of rank-one supports*, which we will utilize later for enumerating the number of required servers.

Definition 16. *Given two supports* $\mathbf{I}\in\{0,1\}^{K\times N}$ *and* $\bar{\mathcal{J}}\in\{0,1\}^{N\times P_1\times P_2\times\ldots\times P_L}$*, the* equivalence classes of rank-one supports *are defined by the equivalence relation* $i\sim j$ *on* $[N]$*, which holds if and only if* $\bar{\mathcal{S}}_i = \bar{\mathcal{S}}_j$*, as represented in Figure 3.4.*

The above splits the columns of $\mathbf{D}$ (and correspondingly the rows of $\bar{\mathcal{E}}$) into equivalence classes such that $i\sim j$ holds if and only if

$$\bar{\mathcal{J}}(i,:,\ldots,:)\times_1\mathbf{I}(:,i) = \bar{\mathcal{J}}(j,:,\ldots,:)\times_1\mathbf{I}(:,j)\,.$$

We next introduce a basic assumption on $\mathbf{D}$ and $\bar{\mathcal{E}}$, termed the *disjoint support assumption*, which will be used to derive the result of Theorem 3.

Definition 17. *(*Disjoint Support Assumption.*) We say that the matrix* $\mathbf{D} \in \mathbb{R}^{K\times N}$ *and the tensor* $\bar{\mathcal{E}} \in \mathbb{R}^{N\times P_1\times\ldots\times P_L}$ *accept the* disjoint support assumption *if and only if for any distinct pair of columns* $\mathbf{D}(:,i), \mathbf{D}(:,i'), i, i' \in [N]$ *of* $\mathbf{D}$ *and the respective two subtensors* $\bar{\mathcal{E}}(i, \underbrace{:,\ldots,:}_{L \text{ terms}}), \bar{\mathcal{E}}(i', \underbrace{:,\ldots,:}_{L \text{ terms}})$ *of* $\bar{\mathcal{E}}$*, then either we have* $supp(\bar{\mathcal{E}}(i,:,\ldots,:) \times_1 \mathbf{D}(:,i)) = supp(\bar{\mathcal{E}}(i',:,\ldots,:) \times_1 \mathbf{D}(:,i'))$ *or* $supp(\bar{\mathcal{E}}(i,:,\ldots,:) \times_1 \mathbf{D}(:,i)) \cap supp(\bar{\mathcal{E}}(i',:,\ldots,:) \times_1 \mathbf{D}(:,i')) = \emptyset$*.*

Next, we describe the *high-dimensional tiles* and their corresponding *component columns and rows*, which are essential in our tensor factorization procedure.

Definition 18. *For two supports* $\mathbf{I} \in \{0,1\}^{K\times N}, \bar{\mathcal{J}} \in \{0,1\}^{N\times P_1\times P_2\times\ldots\times P_L}$*, and for* $\mathcal{C}$ *being the collection of equivalence classes as in Definition 16, then for each class* $\mathcal{P} \in \mathcal{C}$*, we call* $\bar{\mathcal{S}}_{\mathcal{P}}$ *to be the* representative rank-one support *of class* $\mathcal{P}$*, which will also be called the* tile *labeled*[10] *by* $\mathcal{P}$*. Furthermore, for each tile* $\bar{\mathcal{S}}_{\mathcal{P}}$*, let* $\mathbf{c}_{\mathcal{P}} \triangleq \mathbf{I}(:,n), n \in \mathcal{P}$ *(resp.* $\mathbf{r}_{\mathcal{P}} \triangleq \bar{\mathcal{J}}(n,:,\ldots,:), n \in \mathcal{P}$*) be the corresponding* component column *(resp.* component row*) of the representative rank-one support. Finally,* $\mathcal{C}_{\mathcal{P}} \triangleq supp(\mathbf{c}_{\mathcal{P}}) \subset [K]$ *describes the set of the indices of the non-zero elements in* $\mathbf{c}_{\mathcal{P}}$*, while* $\mathcal{R}_{\mathcal{P}} \triangleq supp(\mathbf{r}_{\mathcal{P}}) \subset \prod_{\ell\in[L]}[P_\ell]$ *describes the set of indices of the non-zero elements in* $\mathbf{r}_{\mathcal{P}}$*. This is illustrated in Figure 3.5.*

We then indicate how each equivalence class of tiles covers the product tensor, composed of the union of the representative rank-one supports.

Definition 19. *For every set* $\mathcal{C}' \subseteq \mathcal{C}$ *of equivalence class, let us define the* union of the representative rank-one supports

$$\mathcal{S}_{\mathcal{C}'} \triangleq \bigcup_{\mathcal{P}\in\mathcal{C}'} \bar{\mathcal{S}}_{\mathcal{P}} \tag{3.55}$$

to be the point-wise logical OR of the corresponding $\bar{\mathcal{S}}_{\mathcal{P}}$*.*

In the above definition (cf. (3.55)), $\mathcal{S}_{\mathcal{C}'}$ is simply the area of the product tensor covered by all the tiles $\mathcal{P} \in \mathcal{C}'$.

We conclude Part I by introducing the maximum rank of a representative rank-one support of an equivalence class $\mathcal{C}$, which will be used to determine the number of servers and derive the upper bound in the proof of Theorem 3.

[10]Note that $\bar{\mathcal{S}}_n = \bar{\mathcal{S}}_{\mathcal{P}}$ for any $n \in \mathcal{P}$. Furthermore, the term *tile* will be used interchangeably to represent both $\bar{\mathcal{S}}_{\mathcal{P}}$ and $\mathcal{P}$.

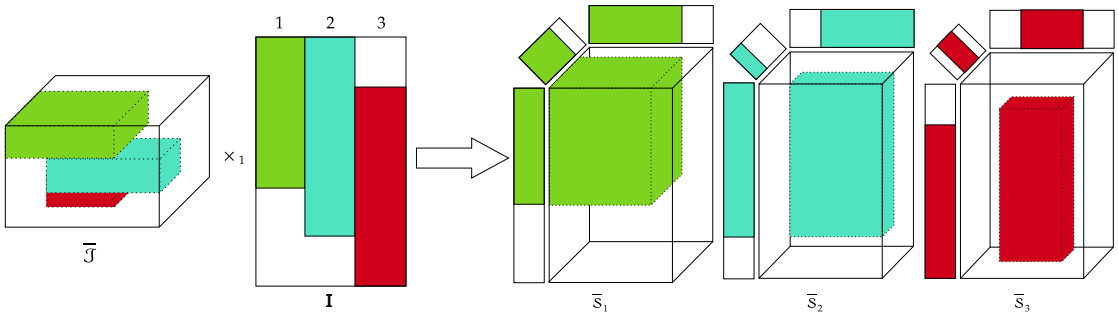

Figure 3.4: The figure on the left illustrates the support constraints $\mathbf{I}$ and $\bar{\mathcal{J}}$ on $\mathbf{D}$ and $\bar{\mathcal{E}}$ respectively. The constraints $\mathbf{I}(:,1)$ and $\bar{\mathcal{J}}(1,:,:)$ on the columns and rows of $\mathbf{D}$ and $\bar{\mathcal{E}}$ respectively are colored green, $\mathbf{I}(:,2)$ and $\bar{\mathcal{J}}(2,:,:)$ are colored cyan and $\mathbf{I}(:,3)$ and $\bar{\mathcal{J}}(3,:,:)$ are colored red. The product of a column with a row of the same color yields the corresponding rank-one contribution support $\bar{\mathcal{S}}_n(\mathbf{I},\bar{\mathcal{J}}), n = 1,2,3$, as described in Definition 15, and as illustrated on the right side of the figure.

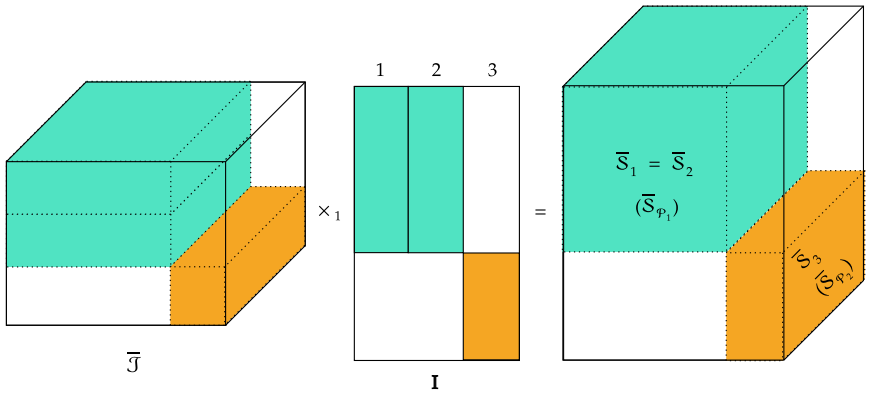

Figure 3.5: This figure illustrates three different rank-one contribution supports $\bar{\mathcal{S}}_1, \bar{\mathcal{S}}_2, \bar{\mathcal{S}}_3$, where the first two fall into the same equivalence class $\bar{\mathcal{S}}_{\mathcal{P}_1} = \bar{\mathcal{S}}_1 = \bar{\mathcal{S}}_2$, while $\bar{\mathcal{S}}_{\mathcal{P}_2} = \bar{\mathcal{S}}_3$.

Definition 20 ([150])**.** *The* maximum rank of a representative rank-one support of class $\mathcal{P} \in \mathcal{C}$ *is obtained as*

$$r_{\mathcal{P}} \triangleq \min(|\mathcal{C}_{\mathcal{P}}|, |\mathcal{R}_{\mathcal{P}}|)\,. \tag{3.56}$$

The above simply says that for the case of $\mathbf{I} = \text{supp}(\mathbf{D})$ and $\bar{\mathcal{J}} = \text{supp}(\bar{\mathcal{E}})$, then the part of tensor $\bar{\mathcal{E}} \times_1 \mathbf{D}$ covered by tile $\bar{\mathcal{S}}_{\mathcal{P}}$ can have rank which is at

most $r_{\mathcal{P}}$.

With the above in place, we proceed with Part II of the proof of Theorem 3 to describe the method used to design the matrix $\mathbf{D}$ and the tensor $\bar{\mathcal{E}}$ for our sparse tensor factorization $\bar{\mathcal{F}} = \bar{\mathcal{E}} \times_1 \mathbf{D}$.

Part II (Construction of $\mathbf{D}$, $\bar{\mathcal{E}}$). We next detail the four-step procedure for deriving the decoding matrix $\mathbf{D}$ and the encoding tensor $\bar{\mathcal{E}}$.

a) **Step 1: Designing the sizes of the tiles for the $K \times P_1 \times \ldots \times P_L$ product tensor $\bar{\mathcal{E}} \times_1 \mathbf{D}$.**

The family of all tiles is initially specified by the set of equivalence classes $\mathcal{C}$ (cf. Definition 16), defined as

$$\mathcal{C} \triangleq \left\{ \mathcal{P}_{i,\underline{\mathbf{j}},r} \subseteq [K] \times [P_1] \times \cdots \times [P_L] \right\} \tag{3.57}$$

where $\mathcal{P}_{i,\underline{\mathbf{j}},r} \triangleq \mathcal{C}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}} \times \mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}$ represents a high-dimensional *tile*, which is initially indexed by $(i, \underline{\mathbf{j}}, r)$ and identified by $\mathcal{C}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}, \mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}$, where

$$\mathcal{C}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}} \triangleq \left[1 + (i-1)\Delta : \min(i\Delta, K) \right] \subseteq [K], \quad i \in \left[\left\lceil \tfrac{K}{\Delta} \right\rceil \right] \tag{3.58}$$

represents the indices of the corresponding columns of $\mathbf{I}$ (cf. Definition 18), $\underline{\mathbf{j}} \triangleq (j_1, \ldots, j_L)$ with each j_ℓ indexing the ℓ^{th} basis subfunction in tensor $\bar{\mathcal{F}}$, and

$$\begin{aligned} \mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}} &\triangleq \prod_{\ell \in [L]} [1] \cup \left(\mathbb{1}(\ell \in \mathcal{Q}_r) \times [1 + (j_\ell - 1)\Lambda_\ell : \min(j_\ell \Lambda_\ell, P_\ell)] \right) \\ &\subseteq [P_1] \times \cdots \times [P_L], \quad \forall j_\ell \in \left[\left\lceil \tfrac{P_\ell}{\Lambda_\ell} \right\rceil \right], r \in \left[\binom{L}{\Gamma} \right] \end{aligned} \tag{3.59}$$

demonstrates the set of indices of the corresponding rows of $\bar{\mathcal{J}}$ (cf. Definition 18), wherein $\mathcal{Q}_r \triangleq \{\ell_1, \ldots, \ell_\Gamma\} \subseteq [L]$ denotes a choice of Γ active coordinates among all L dimensions, i.e., Γ basis subfunctions with the exponent terms $\{p_\ell - 1 > 0\}_{\ell \in \mathcal{Q}_r}$.

The set of equivalence classes $\mathcal{C}$ in (3.57), depending on the dimensions of $\bar{\mathcal{F}}$, include the equivalence classes $\mathcal{C}_s$, $s \in [4]$, which follow from Definition 16 for all possible scenarios where $\Delta \mid K$, $\Lambda_\ell \mid P_\ell$, $\Delta \nmid K$, and $\Lambda_\ell \nmid P_\ell$. We note that for the cases $\Delta \mid K$ and $\Lambda_\ell \mid P_\ell$, we divide each $(\Gamma+1)$-dimensional space corresponding to each $\mathcal{Q}_r$, $r \in [\binom{L}{\Gamma}]$ to $\frac{K}{\Delta} \times \prod_{\ell \in \mathcal{Q}_r} \frac{P_\ell}{\Lambda_\ell}$ tiles each with dimensions $\Delta \times \Lambda_{\ell_1} \times \ldots \times \Lambda_{\ell_\Gamma}$. A similar

procedure holds for the residual space for the general cases $\Delta \nmid K$ and $\Lambda_\ell \nmid P_\ell$.

The number of representative rank-one supports for tiles $\mathcal{P}$ in each equivalence class is thus

$$|\mathcal{C}_1| = \lfloor \frac{K}{\Delta} \rfloor \sum_{r=1}^{\binom{L}{\Gamma}} \prod_{\ell \in \mathcal{Q}_r} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \,, \quad |\mathcal{C}_2| = \sum_{r=1}^{\binom{L}{\Gamma}} \prod_{\ell \in \mathcal{Q}_r} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \,,$$

$$|\mathcal{C}_3| = \lfloor \frac{K}{\Delta} \rfloor \sum_{r=1}^{\binom{L}{\Gamma}} \prod_{\ell \in \mathcal{Q}_r:\ \Lambda_\ell | P_\ell} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \times \prod_{\ell' \in \mathcal{Q}_r:\ \Lambda_{\ell'} \nmid P_{\ell'}} \lceil \frac{P_{\ell'}}{\Lambda_{\ell'}} \rceil \,,$$

$$|\mathcal{C}_4| = \sum_{r=1}^{\binom{L}{\Gamma}} \prod_{\ell \in \mathcal{Q}_r:\ \Lambda_\ell | P_\ell} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \times \prod_{\ell' \in \mathcal{Q}_r:\ \Lambda_{\ell'} \nmid P_{\ell'}} \lceil \frac{P_{\ell'}}{\Lambda_{\ell'}} \rceil \,. \tag{3.60}$$

The above information will be essential in enumerating our equivalence classes and associating each such class with a collection of servers.

The next step is to fill the tiles in the product tensor, using the constitutive component columns and rows, as follows.

b) **Step 2: Filling tiles in $\bar{\mathcal{E}} \times_1 \mathbf{D}$ as a function of $\bar{\mathcal{F}}$.**

Recall that we have a tile $\bar{\mathcal{S}}_{\mathcal{P}}(\mathcal{R}_{\mathcal{P}}, \mathcal{C}_{\mathcal{P}})$ corresponding to the non-zero elements of $\bar{\mathcal{S}}_{\mathcal{P}}$. This tile is "empty" in the sense that all non-zero entries of $\bar{\mathcal{S}}_{\mathcal{P}}$ are equal to 1.

To avoid assigning any entry of $\bar{\mathcal{F}}$ to more than one tile, we fix a strict total ascending order $\prec$ on the tiles $\{\mathcal{P}\}$, e.g., lexicographic on $(i, \underline{\mathbf{j}}, \mathcal{Q}_r)$, and maintain a binary mask $\bar{\mathcal{M}}$ (same size as $\bar{\mathcal{F}}$), initialized to zero. We process tiles in the order $\prec$, and for each tile $\mathcal{P}$, we *zero–force* the previously owned positions as

$$\bar{\mathcal{S}}^{\circ}_{\mathcal{P}} \ \triangleq \ \bar{\mathcal{S}}_{\mathcal{P}} \odot (\bar{\mathbf{1}} - \bar{\mathcal{M}}) \tag{3.61}$$

where $\bar{\mathbf{1}}$ denotes an all-ones tensor of appropriate dimensions.

Consequently, all entries already assigned by earlier tiles are set to zero. If $\bar{\mathcal{S}}^{\circ}_{\mathcal{P}} = \bar{\mathbf{0}}$, with $\bar{\mathbf{0}}$ denoting an all-zeros tensor of appropriate dimensions, we skip $\mathcal{P}$. Otherwise, we define the induced index sets

$$\mathcal{C}^{\circ}_{\mathcal{P}} \triangleq \{k : \exists\, \underline{\mathbf{p}} \text{ s.t. } (k, \underline{\mathbf{p}}) \in \mathrm{Supp}(\bar{\mathcal{S}}^{\circ}_{\mathcal{P}})\} \,,\ \forall \underline{\mathbf{p}} \in \prod_{\ell \in [L]} [P_\ell] \,,$$

$$\mathcal{R}^{\circ}_{\mathcal{P}} \triangleq \{\underline{\mathbf{p}} : \exists\, k \text{ s.t. } (k, \underline{\mathbf{p}}) \in \mathrm{Supp}(\bar{\mathcal{S}}^{\circ}_{\mathcal{P}})\} \,,\ \forall k \in [K] \tag{3.62}$$

to form the cropped subtensor, factorized as

$$\widehat{\bar{\mathcal{F}}}_{\mathcal{P}} \;\triangleq\; \Big(\bar{\mathcal{F}} \odot \bar{\mathcal{S}}^{\circ}_{\mathcal{P}}\Big)\Big(\mathcal{R}^{\circ}_{\mathcal{P}},\, \mathcal{C}^{\circ}_{\mathcal{P}}\Big)\,. \tag{3.63}$$

To decompose the high-dimensional tiles in this problem, we consider the multilinear SVD (MLSVD) approach, detailed in Section 3.5, which is based on the matrix SVD on mode-1 unfolding of Γ-dimensional tiles $\bar{\mathcal{S}}_{\mathcal{P}}$, denoted as $\bar{\mathcal{S}}_{\mathcal{P}_{(1)}}$. We then use the rank properties of the matrix unfolding of a tensor, where we have the rank budget

$$r^{\circ}_{\mathcal{P}} = \operatorname{rank}(\bar{\mathcal{S}}^{\circ}_{\mathcal{P}}) = \operatorname{rank}(\bar{\mathcal{S}}^{\circ}_{\mathcal{P}_{(1)}}) = \min\Big(|\mathcal{C}^{\circ}_{\mathcal{P}}|,\, |\mathcal{R}^{\circ}_{\mathcal{P}}|\Big)\,. \tag{3.64}$$

The maximum rank of each representative rank-one support for such tiles, i.e., of each tile $\mathcal{P}$ of $\bar{\mathcal{E}} \times_1 \mathbf{D}$, from (3.56) and Definition 20, therefore takes the form

$$r^{\circ}_{\mathcal{P}} = \begin{cases} \min\Big(\Delta, |\mathcal{R}^{\circ}_{\mathcal{P}}|\Big)\,, & \mathcal{P} \in \mathcal{C}_1 \bigcup \mathcal{C}_3\,, \\ \min\Big(\operatorname{mod}(K, \Delta), |\mathcal{R}^{\circ}_{\mathcal{P}}|\Big)\,, & \mathcal{P} \in \mathcal{C}_2 \bigcup \mathcal{C}_4 \end{cases} \tag{3.65}$$

where

$$|\mathcal{R}^{\circ}_{\mathcal{P}}| = \begin{cases} \prod\limits_{\ell \in \mathcal{Q}_r} \Lambda_\ell\,, & \mathcal{P} \in \mathcal{C}_1 \bigcup \mathcal{C}_2\,, \\ \prod\limits_{\substack{\ell \in \mathcal{Q}_r: \\ \Lambda_\ell | P_\ell}} \Lambda_\ell \prod\limits_{\substack{\ell' \in \mathcal{Q}_r: \\ \Lambda_{\ell'} \nmid P_{\ell'}}} \operatorname{mod}(P_{\ell'}, \Lambda_{\ell'})\,, & \mathcal{P} \in \mathcal{C}_3 \bigcup \mathcal{C}_4\,. \end{cases} \tag{3.66}$$

We then compute the MLSVD of $\widehat{\bar{\mathcal{F}}}_{\mathcal{P}}$ as

$$\widehat{\bar{\mathcal{F}}}_{\mathcal{P}} \;=\; \bar{\mathcal{R}}_{\mathcal{P}} \times_1 \mathbf{L}_{\mathcal{P}} \tag{3.67}$$

with $\mathbf{L}_{\mathcal{P}} \in \mathbb{R}^{|\mathcal{C}^{\circ}_{\mathcal{P}}| \times r^{\circ}_{\mathcal{P}}}$ and $\bar{\mathcal{R}}_{\mathcal{P}} \in \mathbb{R}^{r^{\circ}_{\mathcal{P}} \times |\mathcal{R}^{\circ}_{\mathcal{P}}(1)| \times \cdots \times |\mathcal{R}^{\circ}_{\mathcal{P}}(L)|}$. The columns of $\mathbf{D}$ reserved for tile $\mathcal{P}$ (one column per rank-1 component) are filled with $\mathbf{L}_{\mathcal{P}}$ on the rows indexed by $\mathcal{C}^{\circ}_{\mathcal{P}}$ and zeros elsewhere. The matching frontal slices of $\bar{\mathcal{E}}$ are filled with $\mathcal{R}^{\circ}_{\mathcal{P}}$ on the indices $\mathcal{R}_{\mathcal{P}}$ and zeros elsewhere. Finally, we update the mask by

$$\bar{\mathcal{M}} \;\leftarrow\; \bar{\mathcal{M}} \;\vee\; \operatorname{Supp}(\bar{\mathcal{S}}^{\circ}_{\mathcal{P}})\,. \tag{3.68}$$

By construction, the owned supports $\{\operatorname{Supp}(\bar{\mathcal{S}}^{\circ}_{\mathcal{P}})\}_{\mathcal{P}}$ are pairwise disjoint, hence no entry of $\bar{\mathcal{F}}$ is assigned twice. Therefore, the later tiles automatically see zeros on intersections.

In the third step, we create and fill the non-zero tiles in the decoding matrix $\mathbf{D}$ and the encoding tensor $\bar{\mathcal{E}}$, as follows.

c) **Step 3: Creating and filling the non-zero tiles in $\mathbf{D}$ and $\bar{\mathcal{E}}$.**

This step starts with MLSVD (as described in Section 3.5) of the cropped tile $\bar{\mathcal{F}}_{\mathcal{P}}$, where this decomposition takes the form

$$\bar{\mathcal{F}}_{\mathcal{P}} = \bar{\mathcal{E}}_{\mathcal{P}} \times_1 \mathbf{D}_{\mathcal{P}} \tag{3.69}$$

where $\mathbf{D}_{\mathcal{P}} \in \mathbb{R}^{|\mathcal{C}^{\circ}_{\mathcal{P}}| \times r^{\circ}_{\mathcal{P}}}$ and $\bar{\mathcal{E}}_{\mathcal{P}} \in \mathbb{R}^{r^{\circ}_{\mathcal{P}} \times |\mathcal{R}^{\circ}_{\mathcal{P}}|}$. In particular, $\bar{\mathcal{F}}_{\mathcal{P}}, \mathbf{D}_{\mathcal{P}}$, and $\bar{\mathcal{E}}_{\mathcal{P}}$ are associated to $\bar{\mathcal{T}}$, $\mathbf{U}^{(1)}$, and $\bar{\mathcal{S}}$ in all complete SVD decomposition of (3.44). Naturally, $\operatorname{rank}(\bar{\mathcal{F}}_{\mathcal{P}}) \leq r^{\circ}_{\mathcal{P}}$.

Let

$$\bigcup_{i=1}^{4} \mathcal{C}_i \triangleq \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m\},\ m \in \mathbb{N} \tag{3.70}$$

describe the enumeration we give to each tile. Then, the position that each cropped tile takes inside $\mathbf{D}$, is given by

$$\mathcal{C}^{\circ}_{\mathcal{P}_j},\ \Big[\sum_{i=1}^{j-1} r^{\circ}_{\mathcal{P}_i} + 1 \ :\ \sum_{i=1}^{j} r^{\circ}_{\mathcal{P}_i}\Big], \quad \forall\, \mathcal{P}_j \in \bigcup_{s=1}^{4} \mathcal{C}_s \tag{3.71}$$

and the position of each cropped tile in $\bar{\mathcal{E}}$ is given by

$$\Big[\sum_{i=1}^{j-1} r^{\circ}_{\mathcal{P}_i} + 1 \ :\ \sum_{i=1}^{j} r^{\circ}_{\mathcal{P}_i}\Big],\ \mathcal{R}^{\circ}_{\mathcal{P}_j}, \quad \forall\, \mathcal{P}_j \in \bigcup_{s=1}^{4} \mathcal{C}_s\,. \tag{3.72}$$

In particular, these yield

$$\mathbf{D}\left(\mathcal{C}^{\circ}_{\mathcal{P}_j}, \Big[\sum_{i=1}^{j-1} r^{\circ}_{\mathcal{P}_i} + 1, \sum_{i=1}^{j} r^{\circ}_{\mathcal{P}_i}\Big]\right) = \mathbf{D}_{\mathcal{P}_j} \tag{3.73}$$

and

$$\bar{\mathcal{E}}\left(\Big[\sum_{i=1}^{j-1} r^{\circ}_{\mathcal{P}_i} + 1, \sum_{i=1}^{j} r^{\circ}_{\mathcal{P}_i}\Big], \mathcal{R}^{\circ}_{\mathcal{P}_j}\right) = \bar{\mathcal{E}}_{\mathcal{P}_j} \tag{3.74}$$

while naturally the remaining non-assigned elements of $\mathbf{D}$ and $\bar{\mathcal{E}}$ are zero.

We are now ready to upper bound N.

d) **Step 4: An upper bound to the number of servers**.

We now proceed to bound the number of rank-one contribution supports (corresponding to the number of servers), and we do so under our previously stated assumption of disjoint supports (cf. Definition 17), which is equivalent to the disjoint-tiles assumption via the following lemma.

Lemma 4. *For matrix* $\mathbf{D}$ *and tensor* $\bar{\mathcal{E}}$*, the representative supports* $\{\bar{\mathcal{S}}_{\mathcal{P}_i}\}_{i=1}^m$ *of* $\bar{\mathcal{E}} \times_1 \mathbf{D}$ *are disjoint (i.e.,* $\bar{\mathcal{S}}_{\mathcal{P}_i} \cap \bar{\mathcal{S}}_{\mathcal{P}_j} = \bar{\mathbf{0}},\ j \neq i$*) if and only if* $\mathbf{D}$ *and* $\bar{\mathcal{E}}$ *accept the* disjoint support assumption.

Proof of Lemma 4 .

- Disjoint support assumption $\implies$ disjoint representative supports: Assume that $\mathbf{D} \in \mathbb{R}^{K \times N}$ and $\bar{\mathcal{E}} \in \mathbb{R}^{N \times P_1 \times \ldots \times P_L}$ satisfy the *disjoint support assumption* of Definition 17. Then, for any $i, i' \in [N]$, either the supports coincide, i.e.,

$$\begin{aligned} \operatorname{supp}(\mathbf{D}(:,i)) &= \operatorname{supp}(\mathbf{D}(:,i')) \ \text{ and} \\ \operatorname{supp}(\bar{\mathcal{E}}(i,:,\ldots,:)) &= \operatorname{supp}(\bar{\mathcal{E}}(i',:,\ldots,:)) \end{aligned}$$

  or they are disjoint. Consider now the induced supports of the mode-1 product $\bar{\mathcal{E}} \times_1 \mathbf{D}$. By construction, either

$$\operatorname{supp}(\bar{\mathcal{E}}(i,:,\ldots,:) \times_1 \mathbf{D}(:,i)) = \operatorname{supp}(\bar{\mathcal{E}}(i',:,\ldots,:) \times_1 \mathbf{D}(:,i')) \tag{3.75}$$

  or the two supports are disjoint. Therefore, the representative supports $\{\bar{\mathcal{S}}_{\mathcal{P}_i}\}_{i=1}^m$ are pairwise disjoint, which yields the disjoint representative support equivalence classes in Definition 16.

- Disjoint representative supports $\implies$ disjoint support assumption: Now assume that $\bar{\mathcal{E}} \times_1 \mathbf{D}$ satisfies the disjoint representative support assumption. Let $\mathcal{C} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ denote the induced equivalence classes. By the definition of these classes, for any two representatives $\mathcal{P}, \mathcal{P}' \in \mathcal{C}$, their supports are either identical or disjoint. This property propagates back to the column supports of $\mathbf{D}$ and the row supports of subtensors of $\bar{\mathcal{E}}$. Hence, for all $i, i' \in [N]$, we have that

$$\begin{aligned} &\operatorname{supp}(\mathbf{D}(:,i)) = \operatorname{supp}(\mathbf{D}(:,i')) \ \text{ or} \\ &\operatorname{supp}(\mathbf{D}(:,i)) \cap \operatorname{supp}(\mathbf{D}(:,i')) = \emptyset\,, \end{aligned} \tag{3.76}$$

  and similarly,

$$\begin{aligned} &\operatorname{supp}(\bar{\mathcal{E}}(i,:,\ldots,:)) = \operatorname{supp}(\bar{\mathcal{E}}(i',:,\ldots,:)) \ \text{ or} \\ &\operatorname{supp}(\bar{\mathcal{E}}(i,:,\ldots,:)) \cap \operatorname{supp}(\bar{\mathcal{E}}(i',:,\ldots,:)) = \emptyset\,. \end{aligned} \tag{3.77}$$

  Combining these observations in (3.75), (3.76), and (3.77) shows that the induced supports of the mode-1 products, described as

$$\operatorname{supp}(\bar{\mathcal{E}}(i,:,\ldots,:) \times_1 \mathbf{D}(:,i))\,,$$

are either identical or disjoint for any distinct pair i, i', which is precisely the disjoint support assumption of Definition 17.
This completes the proof of Lemma 4. □

Completing the proof of Lemma 4, we next use the equivalence of the disjoint support assumption and the disjoint representative supports to finalize the proof of Theorem 3. To that end, we recall from Section 3.4 (see also (3.39)–(3.40)) that each server $n \in [N]$ corresponds to one column of the decoding matrix $\mathbf{D}$ and the associated row of the encoding tensor $\bar{\mathcal{E}}$. To express N in terms of the scheme parameters, consider the tiles $\mathcal{P} \in \mathcal{C}$ and the associated submatrices $\mathbf{D}_{\mathcal{P}}$ defined in (3.69). From (3.73), these submatrices collectively span $\sum_{i=1}^{m} r^{\circ}_{\mathcal{P}_i}$ columns.

The total number of required servers is then obtained as $N^{\star} \leq \sum_{\mathcal{P}} r^{\circ}_{\mathcal{P}}$ while the design constraints $(\Gamma, \Delta, \{\Lambda_\ell\}_{\ell \in [L]})$ are satisfied at each server. Finally, as one can readily verify, the above design corresponds to

$$
\begin{aligned}
N^{\star} &\leq \sum_{s \in [4]} \sum_{\mathcal{P} \in \mathcal{C}_s} r^{\circ}_{\mathcal{P}} \qquad (3.78)\\
&= \lfloor \frac{K}{\Delta} \rfloor \sum_{r=1}^{\binom{L}{\Gamma}} \min\Big(\Delta, \prod_{\ell \in \mathcal{Q}_r} \Lambda_\ell\Big) \times \prod_{\ell \in \mathcal{Q}_r} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \\
&+ \sum_{r=1}^{\binom{L}{\Gamma}} \min\Big(\mathrm{mod}(K, \Delta), \prod_{\ell \in \mathcal{Q}_r} \Lambda_\ell\Big) \times \prod_{\ell \in \mathcal{Q}_r} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \\
&+ \lfloor \frac{K}{\Delta} \rfloor \sum_{r=1}^{\binom{L}{\Gamma}} \min\Big(\Delta, \prod_{\substack{\ell \\ \ell, \ell' \in \mathcal{Q}_r:\ \Lambda_\ell | P_\ell,\ \Lambda_{\ell'} \nmid P_{\ell'}}} \Lambda_\ell \prod_{\ell'} \mathrm{mod}(P_{\ell'}, \Lambda_{\ell'})\Big) \\
&\times \prod_{\ell \in \mathcal{Q}_r:\ \Lambda_\ell | P_\ell} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \times \prod_{\ell' \in \mathcal{Q}_r:\ \Lambda_{\ell'} \nmid P_{\ell'}} \lceil \frac{P_{\ell'}}{\Lambda_{\ell'}} \rceil \\
&+ \sum_{r=1}^{\binom{L}{\Gamma}} \min\Big(\mathrm{mod}(K, \Delta), \prod_{\substack{\ell \\ \ell, \ell' \in \mathcal{Q}_r:\ \Lambda_\ell | P_\ell,\ \Lambda_{\ell'} \nmid P_{\ell'}}} \Lambda_\ell \prod_{\ell'} \mathrm{mod}(P_{\ell'}, \Lambda_{\ell'})\Big) \\
&\times \prod_{\ell \in \mathcal{Q}_r:\ \Lambda_\ell | P_\ell} \lfloor \frac{P_\ell}{\Lambda_\ell} \rfloor \times \prod_{\ell' \in \mathcal{Q}_r:\ \Lambda_{\ell'} \nmid P_{\ell'}} \lceil \frac{P_{\ell'}}{\Lambda_{\ell'}} \rceil .
\end{aligned}
$$

Letting $\Delta \mid K$, $\Lambda_\ell = \Lambda$, $P_\ell = P$, $\ell \in [L]$, and $\Lambda \mid P$, the upper bound on $N^{\star}$ in (3.78) is simplified to (3.50), which completes the proof of Theorem 3. □

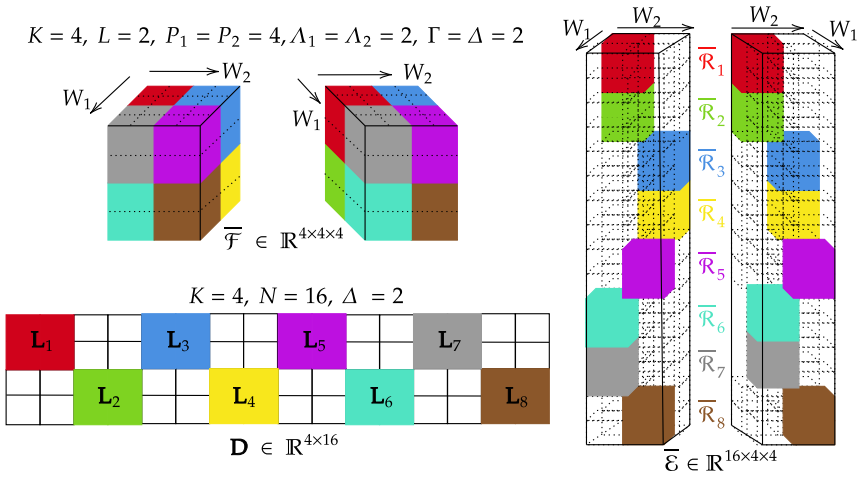

Figure 3.6: Corresponding to Example 4 (Case I), this figure illustrates the partitioning of $\bar{\mathcal{F}}$ into 8 tiles of size $(2 \times 2 \times 2)$, and the sparse tiling of $\mathbf{D}$ and $\bar{\mathcal{E}}$ with tiles $\mathbf{L}_j$ and $\bar{\mathcal{R}}_j$, respectively, resulting in the full tiling of $\bar{\mathcal{F}} = \bar{\mathcal{E}} \times_1 \mathbf{D}$.

We next present basic examples of multi-user non-linearly decomposable problems and illustrate the interplay among system parameters based on Theorem 3.

Example 4. *Consider the* $(K = 4, N, L = 2, \Gamma = 2, \{P_\ell = 4, \Lambda_\ell = 2\}_{\ell\in[L]})$ *setting, with* N *servers tasked with computing non-linear functions of* $L = 2$ *basis subfunctions*

$$F_k(.) = \sum_{(p_1,p_2)\in[4]\times[4]} c_{k,\underline{\mathbf{p}}}\, W_1^{p_1-1} W_2^{p_2-1}, \quad k \in [4]\,. \tag{3.79}$$

The coefficients $c_{k,\underline{\mathbf{p}}}$ *are described by tensor* $\bar{\mathcal{F}}$*, as demonstrated in Figure 3.6. Given the design constraints, we aim to guarantee lossless reconstruction of (3.79).*

Case I*:* $\Delta = 2$ **.** *we directly conclude from* (3.50) *that we need* $N \leq 16$ *servers. To tackle this challenge of reconstruction, we need to construct*

1. *The* $(N \times P_1 \times P_2) = (16 \times 4 \times 4)$ *computing tensor* $\bar{\mathcal{E}}$*, specifying the computational tasks of each server.*

2. *The* $(K \times N) = (4 \times 16)$ *communication matrix* $\mathbf{D}$*, determining the server-user connections.*

These originate from the decomposition of $(K \times P_1 \times P_2) = (4 \times 4 \times 4)$ *tensor* $\bar{\mathcal{F}}$ *(cf.* (3.38)*) as* $\bar{\mathcal{F}} = \bar{\mathcal{E}} \times_1 \mathbf{D}$*, representing the requested functions. The solution is then as follows.*

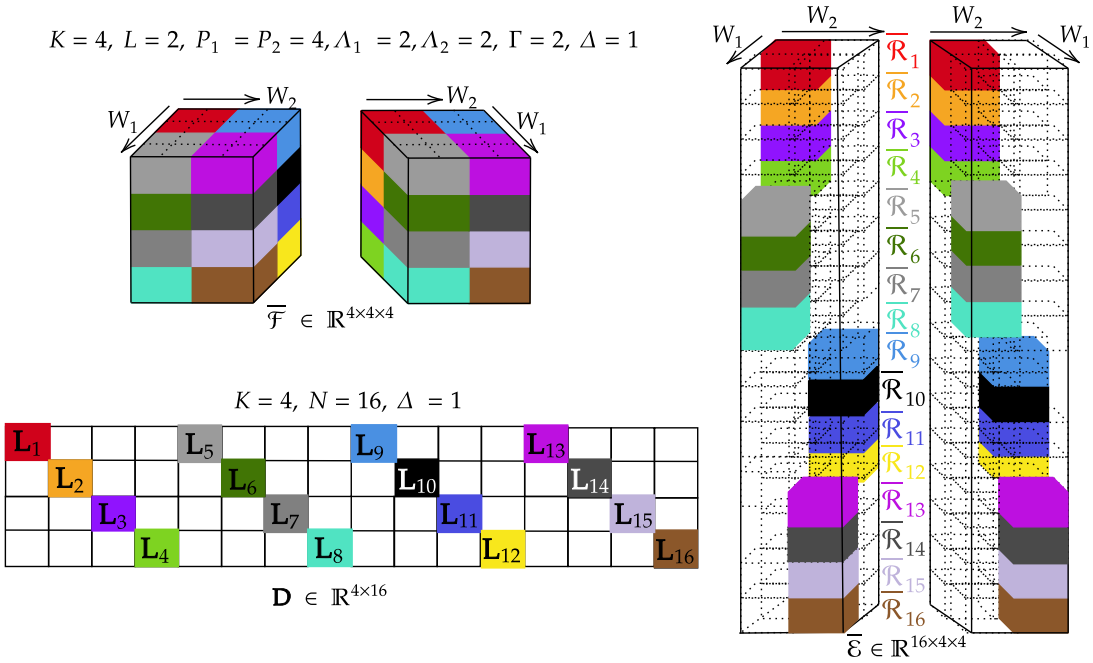

Figure 3.7: Pertaining to Example 4 (Case II) with $N = 16$ servers, the new tessellation pattern allows for a reduced $\Delta = 1$ reflecting a reduction from $\delta = 1/2$ in Case I to $\delta = 1/4$.

1. *Initially, we partition* $\bar{\mathcal{F}}$ *into* $\frac{K}{\Delta}\frac{P_1}{\Lambda_1}\frac{P_2}{\Lambda_2} = 2 \cdot 2 \cdot 2 = 8$ *disjoint* $\Delta \times \Lambda_1 \times \Lambda_2$ *subtensors* $\bar{\mathcal{S}}_j \in \mathbb{R}^{2\times 2\times 2}$, $j \in [8]$, *as depicted in Figure 3.6.*

2. *Next, using the standard tensor decomposition form (cf. Section 3.5), we decompose each* $\bar{\mathcal{S}}_j$ *as* $\bar{\mathcal{S}}_j = \bar{\mathcal{R}}_j \times_1 \mathbf{L}_j$, *where* $\bar{\mathcal{R}}_j \in \mathbb{R}^{2\times 2\times 2}, \mathbf{L}_j \in \mathbb{R}^{2\times 2}$ *for all* $j \in [8]$, *noting that such full decomposition is feasible since the maximum rank of each* $\bar{\mathcal{S}}_j$ *is* $\min(\Delta, \Lambda_1\Lambda_2) = 2$.

3. *Finally, we construct* $\mathbf{D} \in \mathbb{R}^{4\times 16}$ *and* $\bar{\mathcal{E}} \in \mathbb{R}^{16\times 4\times 4}$ *by tiling them with* $\mathbf{L}_j$ *and* $\bar{\mathcal{R}}_j$, *respectively, as in Figure 3.6.*

Example 4 provides a glimpse of the general principle behind creating our scheme. In brief, we begin by splitting our $K \times P_1 \times P_2$ tensor $\bar{\mathcal{F}}$ into $\frac{K}{\Delta}\frac{P_1}{\Lambda_1}\frac{P_2}{\Lambda_2}$ subtensors of size $\Delta \times \Lambda_1 \times \Lambda_2$. We decompose these subtensors into submatrices $\{\mathbf{L}_j\}_{j\in[8]}$ and into subtensors $\{\bar{\mathcal{R}}_j\}_{j\in[8]}$ that form tiles of $\mathbf{D}$ and $\bar{\mathcal{E}}$, respectively. The tile placement must respect the sparsity constraints $(\Gamma, \Delta, \Lambda_1, \Lambda_2)$ and must yield $\bar{\mathcal{E}} \times_1 \mathbf{D} = \bar{\mathcal{F}}$. Regarding the required number of servers, the general rule is that N is the number of subtensors, multiplied by the rank of each subtensor. Since we had 8 subtensors, each of rank 2,

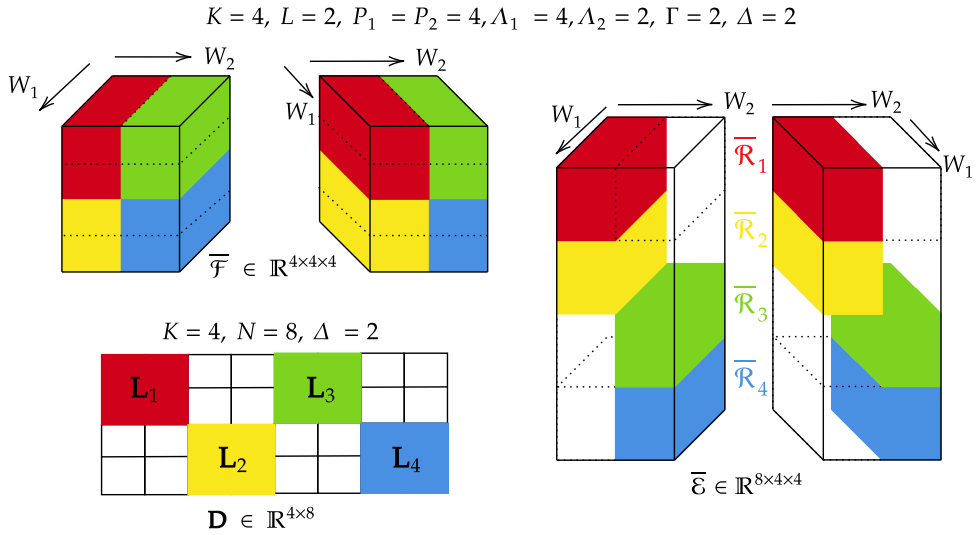

Figure 3.8: Corresponding to Example 4 with $\Lambda_1 = 4$, this figure illustrates the partitioning of $\bar{\mathcal{F}}$ into 4 tiles of size $(\Delta \times \Lambda_1 \times \Lambda_2) = (2 \times 4 \times 2)$, and also illustrates the sparse tiling of $\mathbf{D}$ and $\bar{\mathcal{E}}$ with tiles $\mathbf{L}_j$ and $\bar{\mathcal{R}}_j$ respectively, resulting in the full tiling of $\bar{\mathcal{F}} = \bar{\mathcal{E}} \times_1 \mathbf{D}$.

the proposed construction requires $N_{\text{ach}} = 16$ servers, and therefore $N^\star \leq 16$. In contrast, it is easy to show that the linearized approach, employing sparse matrix factorization of [105], would entail $L_M = \prod_{\ell \in [L]} P_\ell = 16$ basis subfunctions and approximately double the number of servers, for the same Γ and Δ.

At the same time, some reductions in γ, δ may arise naturally. In particular, multiple tessellation patterns can yield the same required value of N, but variations in tile size and placement may result in different values of Γ and Δ. To illustrate this, consider the following case.

Case II: $\Delta = 1$ **.** In the same lossless computing setting, we consider again $N = 16$ servers, $K = 4$ users, $L = 2$ subfunctions. We also maintain a computational cost of $\Gamma = 2$, but now we see that the tessellation pattern in Figure 3.7 allows for a reduced $\Delta = 1$ corresponding to $\delta = 1/4$.

On the other hand, once we reduce the computation and communication capabilities of each server, more servers may be required. Considering different multiplicative terms for the subfunctions could also significantly change the number of required servers. For example, for the same setting as in Example 4, again for $K = 4, L = 2, P_1 = P_2 = 4$, and again for $\Delta = 2$ (corresponding to $\delta = \frac{1}{2}$) and $\Gamma = 2$ (corresponding to $\gamma = \frac{1}{2}$), if we aim to evaluate different degrees of subfunctions corresponding to $\Lambda_1 = 4, \Lambda_2 = 2$, then the required

number of servers N is now 8 (cf. (3.50)), and the corresponding tessellation pattern is presented in Figure 3.8, where we see 4 subtensors of rank 2.

We next show in Proposition 1 that, in the matrix factorization approach of TDC [105], using a potentially large dimension L to accommodate more basis subfunctions increases the total computational load per server. In contrast, our proposed model improves computational efficiency by using fewer basis subfunctions, each with a bounded range of degrees, captured by the multiplication cost. Our comparison assumes that the time to compute a basis subfunction is the same as the time to compute one power of its output.

Proposition 1. *Consider a lossless* $\big(K, N, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell\in[L]}\big)$ *distributed computing system with* $P_\ell = P$ *and* $\Lambda_\ell = \Lambda$ *for all* $\ell \in L$*, where* $P > 2$*. Under the tensor factorization approach proposed in this work, with computation cost* $\Gamma_T = L$ *and multiplication cost* Λ*, the total computation cost is upper bounded by*

$$NL(\Lambda + 1)\,, \tag{3.80}$$

which scales linearly with L*. In contrast, under the matrix factorization approach in [105], with per-server computation cost* Γ_M*, the total computation cost is upper bounded by*

$$N\left(\Gamma_M + \left(\tfrac{P(P-1)}{2}\right)^L\right), \tag{3.81}$$

which grows exponentially with L*.*

Proof. The overall computational cost for the non-linearly separable distributed computing setting proposed in this work can be upper bounded as

$$N\Big(\Gamma_T + \Lambda_1 + \ldots + \Lambda_L\Big) \overset{(a)}{\leq} N\Big(L(\Lambda + 1)\Big)$$

because we have N servers each computing a basis subfunction with a computation cost Γ_T as well as the powers of all L basis subfunctions each with a multiplication cost Λ_ℓ and (a) holds because of the assumptions $\Gamma_T = L$ and $\Lambda_\ell = \Lambda$, $\ell \in [L]$.

The total computation cost of the matrix factorization approach in [105] is upper bounded as

$$\begin{aligned} N\Big(\Gamma_M + \sum_{\underline{\mathbf{p}}\in[2:P_1]\times\ldots\times[2:P_L]} p_1 \times \ldots \times p_L\Big) &\overset{(b)}{=} N\Big(\Gamma_M + \sum_{p_1\in[2:P]} p_1 \times \ldots \times \sum_{p_L\in[2:P]} p_L\Big) \\ &\overset{(c)}{=} N\Big(\Gamma_M + \big(\frac{P(P-1)}{2}\big)^L\Big) \end{aligned}$$

because we have N servers each computing a basis subfunction with a computation cost Γ_M as well as the number of multiplications for powers of each basis subfunction $\ell \in [L]$, ranging from 2 to P_ℓ, (b) follows from some algebraic manipulations, and (c) follows from $1 + \ldots + n = \frac{n(n+1)}{2}$, assuming $n = P - 1$. □

In the multi-user non-linearly separable distributed computing setting, the coordinator node must split the global computational demand tensor into smaller tensor blocks (subtensors) that can be assigned to different servers. These tensor partitions arise naturally from the need to distribute the computation across the servers while respecting the support constraints imposed by the exponent vectors of the requested basis subfunctions. However, if the partitioning is not carefully designed, the resulting subtensors may inherit large combinatorial overlaps across multiple dimensions, which in turn leads to substantial rank inflation and consequently requires more servers to compute all subtasks without error. Therefore, designing tensor partitions that minimize the resulting *sum rank* is central, as the rank of each subtensor directly determines the computation and communication loads at each server. We will address this by introducing a novel combinatorial reduction algorithm based on matching in a bipartite graph.

Next, we address the overlap of the induced subtensors (tile closures), which leads to ambiguity in the assignment of admissible index tuples and may result in inefficient designs. To resolve this issue, we introduce a capacitated tuple-to-tile assignment algorithm that enforces a disjoint decomposition by assigning each tuple to exactly one feasible tile. This construction yields a valid achievable scheme and improves the achievable rate of the proposed approach in Theorem 3.

3.6.2 Refined System Rate via a Combinatorial Reduction Algorithm

We here present a tighter achievable rate result for the proposed lossless distributed computing setting of $(K, N, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell \in [L]})$, focusing on the simplified case where $\delta, \lambda_\ell \in \mathbb{N}$. In particular, we upper bound the required number of servers N, which is the common dimension between $\bar{\mathcal{E}}$, $\mathbf{D}$, and directly impacts our sparse tensor factorization approach. Particularly, we consider a mode-1 matrix unfolding of tensor $\bar{\mathcal{F}}$ and use its rank properties to elaborate the number of required servers as the sum of the total number of tiles with different maximum representative rank-one supports. Since tiles may have combinatorial intersections, we introduce a combinatorial reduction approach based on a bipartite graph matching and a recursive assignment

maps each intersecting index tuple to a tile cell. This procedure results in tiles with a lower sum rank compared to the tensor factorization approach (cf. Theorem 3).

Theorem 4 (Achievable scheme via capacitated tile assignment)**.** *Consider the lossless distributed computing system with the set of parameters* $\big(K, N, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell\in[L]}\big)$ *under the homogeneous setting* $P_\ell = P$ *and* $\Lambda_\ell = \Lambda,\ \forall\, \ell \in [L]$*, with* $\Delta \mid K$ *and* $\Lambda \mid P$*. Algorithm 1 constructs a feasible assignment of the admissible tuples to the candidate tiles. The resulting disjoint tile assignment induces a lossless distributed computing scheme using*

$$N_{\text{alg}} \triangleq \frac{K}{\Delta} \sum_{\beta\in[n_{\mathcal{P}}]} \min\Big\{\Delta, \big|\mathcal{R}^{\star}_{\mathcal{P}_\beta}\big|\Big\} \tag{3.82}$$

servers, where $\mathcal{R}^{\star}_{\mathcal{P}_\beta}$ *denotes the disjoint row-index set assigned to tile* $\mathcal{P}_\beta$ *by Algorithm 1. Consequently, the rate*

$$R_{\text{alg}} \triangleq \frac{K}{N_{\text{alg}}} \tag{3.83}$$

is achievable. Therefore, the minimum required number of servers $N^\star$ *and the maximum achievable system rate* $R^\star$ *satisfy*

$$N^\star \le N_{\text{alg}}, \tag{3.84}$$

$$R^\star = \frac{K}{N^\star} \ge R_{\text{alg}} = \frac{K}{N_{\text{alg}}}. \tag{3.85}$$

Proof. The proof follows from the same parts and steps as the proof of Theorem 3. However, we have further details in Part II (Construction of $\mathbf{D}$, $\bar{\mathcal{E}}$), which we will explain here.

We next detail the four-step procedure for deriving $\mathbf{D}$ and $\bar{\mathcal{E}}$.

a) **Step 1: Designing the sizes of the tiles for the** $K \times P_1 \times \ldots \times P_L$ **product tensor** $\bar{\mathcal{E}} \times_1 \mathbf{D}$.

The family of all tiles is initially specified by the same set of equivalence classes $\mathcal{C}$ as in (3.57).

Despite having L basis subfunctions and therefore a L-dimensional space for each user's demand, indexed by the tuple $\underline{\mathbf{p}}$, the computation cost Γ restricts each server to compute at most Γ basis subfunctions and their multiplicative combinations. This induces $(\Gamma+1)$-dimensional tiles, denoted by $\mathcal{P}_{i,\underline{\mathbf{j}},r}$, corresponding to Γ active exponent dimensions and one user dimension.

Each tile $\mathcal{P}_{i,\underline{\mathbf{j}},r}$ is geometrically composed of Λ^Γ unit cells, corresponding to exponent tuples with exactly Γ active coordinates. However, for the purpose of assignment, each tile is associated with a larger *row-index set* (closure), denoted by $\mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}$, which includes all tuples whose active support is contained in the tile support and whose active values lie within the selected windows.

The row-index set of each initial tile has a cardinality $\left|\mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}\right| = \prod_{\ell \in \mathcal{Q}_r} \Lambda_\ell$, describing the number of unit cells of size 1×1 in each initial tile. For simplicity, we assume $\{\Lambda_\ell = \Lambda, P_\ell = P\}_{\ell \in [L]}$ and $\Lambda \mid P$, resulting in $\left|\mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}\right| = \Lambda^\Gamma$. The total number of initial tiles of various shapes and sizes, whose construction is detailed in the proof of Theorem 3, Part II, a) Step 1, corresponding to each user demand [11] within the set of equivalence classes $\mathcal{C}_1$, is given by

$$n_{\mathcal{P}} \triangleq \binom{L}{\Gamma} \left(\frac{P}{\Lambda}\right)^\Gamma \tag{3.86}$$

which follows from the fact that each tile $\mathcal{P}_{i,\underline{\mathbf{j}},r}$ is indexed by $(i, \underline{\mathbf{j}}, r)$, where i labels the column index, $\underline{\mathbf{j}}$ denotes the L-dimensional tuple of row index set, and $\mathcal{Q}_r$ specifies the r^{th} tile pattern or configuration of active dimensions.

Tile closures $\mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}$ may overlap due to two distinct origins, detailed below.

1. Combinatorial overlaps: different Γ-subsets $\mathcal{Q}_r \subseteq [L]$ may contain the same lower-dimensional support, resulting in combinatorial overlaps across different support classes $\mathcal{Q}_r$,
2. Geometric overlaps: within a fixed support class, different window selections may also contain the same tuple due to inactive coordinates taking value 1.

Consequently, a given tuple $\underline{\mathbf{p}}$ may belong to multiple row-index sets $\mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}$, and thus must be assigned uniquely to one tile to ensure a disjoint support structure for the MLSVD decomposition. Because of these two forms of overlap, naive layered counting leads to overcounting. The feasibility graph and the induced unique assignment are therefore essential for obtaining a valid disjoint tile decomposition.

[11] Considering all K user demands, $\frac{K}{\Delta} n_{\mathcal{P}} \min(\Delta, \Lambda^\Gamma)$ recovers the result in Theorem 3.

The maximum rank of each representative support, based on the matrix unfolding of the tiles, from (3.56) and Definition 20 (assuming $\Delta \mid K$), is given by

$$r_{\mathcal{P}} = \min\left(\Delta, |\mathcal{R}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}|\right). \tag{3.87}$$

We aim to assign the L-dimensional index tuples to the Γ-dimensional tiles in each user's demand tensor through a combinatorial reduction. The resulting assigned row-index sets are denoted by $\mathcal{R}^{\star}_{\mathcal{P}_{i,\underline{\mathbf{j}},r}}$, and the induced number of required servers N is given by the corresponding sum rank of the resulting subtensors.

To obtain an upper bound on N, we construct feasible tessellation patterns by formulating the assignment as a combinatorial problem based on bipartite graph matching with capacities.

We next introduce the set of admissible index tuples, which is required in the bipartite graph construction in Definition 21.

Let $\mathcal{U}$ denote the set of all admissible index tuples $\underline{\mathbf{p}}$ with at most Γ active coordinates, i.e.,

$$\mathcal{U} \triangleq \left\{ \underline{\mathbf{p}} \in [P]^L \;\middle|\; \left|\mathrm{supp}\Big(\mathbf{I}(\underline{\mathbf{p}})\Big)\right| \leq \Gamma \right\},$$

where $\mathbf{I}(\underline{\mathbf{p}}) \triangleq \Big[\mathbb{1}(p_1 > 1) \;\ldots\; \mathbb{1}(p_L > 1)\Big]$. The total number of admissible tuples is

$$n_{\mathcal{U}} = |\mathcal{U}| = \sum_{d=0}^{\Gamma} \binom{L}{d} (P-1)^d. \tag{3.88}$$

Definition 21 (Tuple–tile feasibility graph with capacities)**.** *Let $\{\mathcal{P}_\beta\}_{\beta\in[n_{\mathcal{P}}]}$ denote the set of candidate tiles, each associated with a row-index (closure) set $\mathcal{R}_{\mathcal{P}_\beta}$.*

We define the bipartite graph $G(\mathcal{U} \cup \mathcal{V}, \mathcal{E})$ as follows:

- ***Left vertex set:*** *$\mathcal{U} = \{u_\alpha\}_{\alpha\in[n_{\mathcal{U}}]}$, where each u_α corresponds to an admissible tuple $\underline{\mathbf{p}}_\alpha$.*
- ***Right vertex set:*** *$\mathcal{V} = \{\mathcal{V}_\beta\}_{\beta\in[n_{\mathcal{P}}]}$, where each supernode $\mathcal{V}_\beta$ corresponds to a tile $\mathcal{P}_\beta$.*
- ***Edge set:*** *An edge exists if and only if*

$$(u_\alpha, \mathcal{V}_\beta) \in \mathcal{E} \quad \Longleftrightarrow \quad \underline{\mathbf{p}}_\alpha \in \mathcal{R}_{\mathcal{P}_\beta}.$$

- ***Capacity:*** *Each tile $\mathcal{V}_\beta$ has capacity*

$$\kappa_\beta \triangleq |\mathcal{R}_{\mathcal{P}_\beta}|,$$

representing the maximum number of index tuples that can be assigned to that tile.

We seek a disjoint assignment mapping each tuple $u_\alpha \in \mathcal{U}$ to exactly one feasible tile $\mathcal{V}_\beta$ such that the capacity constraints are satisfied.

Let $\mathcal{M}$ denote the resulting assignment. The induced assigned row-index set of tile $\mathcal{P}_\beta$ is

$$\mathcal{R}^\star_{\mathcal{P}_\beta} \triangleq \left\{ \underline{\mathbf{p}}_\alpha \mid (u_\alpha, \mathcal{V}_\beta) \in \mathcal{M} \right\},$$

and the corresponding rank contribution is

$$r^\star_{\mathcal{P}_\beta} = \min\left(\Delta, |\mathcal{R}^\star_{\mathcal{P}_\beta}|\right).$$

Thus, the total number of servers induced by an assignment $\mathcal{M}$ is

$$N^\star \leq N_{\text{alg}} = \sum_\beta r^\star_{\mathcal{P}_\beta}$$

which coincides with the sum rank of the resulting tile decomposition. While minimizing the number of servers is equivalent to minimizing this sum rank over all feasible assignments, the proposed algorithm focuses on constructing a feasible assignment and then evaluating the induced server count. Accordingly, it provides an achievable scheme rather than minimizing the sum rank objective, which is computationally challenging.

This problem naturally corresponds to a capacitated bipartite assignment (or flow) problem. At a high level, we iteratively construct a feasible assignment by combining maximum flow computations with a simple greedy fallback step.

We adopt an iterative algorithm (Algorithm 1) that alternates between global matching and local assignment, as summarized below.

We next describe how Algorithm 1 attains global feasibility.

Remark 2 (Resolution of infeasibility via local assignment)**.** *When a global feasible assignment does not exist, the corresponding capacitated bipartite graph exhibits an imbalance between a subset of tuples and the available capacity of their neighboring tiles. This can be interpreted as*

Algorithm 1: Capacitated Flow-Based Tuple-to-Tile Assignment

Input: Graph $G(\mathcal{U}, \mathcal{V}, \mathcal{E})$ with capacities κ_β
Initialize: $\mathcal{M} \leftarrow \emptyset$, $\mathcal{U}_{\text{rem}} \leftarrow \mathcal{U}$, remaining capacities κ_β
while $\mathcal{U}_{\text{rem}} \neq \emptyset$ **do**
  Compute a maximum feasible assignment (flow) on the residual graph
  if all remaining tuples are assigned **then**
    update $\mathcal{M}$ and **break**
  Select a tuple $u_\alpha \in \mathcal{U}_{\text{rem}}$
  Assign u_α to any feasible tile $\mathcal{V}_\beta$ with remaining capacity
  Update $\mathcal{M}$, $\mathcal{U}_{\text{rem}}$, and capacities
Output: Assignment $\mathcal{M}$

a generalized violation of Hall-type conditions[154] in the capacitated setting.

The local assignment step resolves this issue by assigning at least one tuple to a feasible tile, thereby reducing the size of the remaining instance and modifying the residual capacity structure. Thus, Algorithm 1 progressively eliminates infeasible configurations and eventually restores global feasibility.

We next address the feasibility-first nature of the assignment provided by Algorithm 1.

Remark 3 (Feasibility-first nature of the algorithm)**.** *Algorithm 1 provides a feasible assignment. Its objective is to map every admissible tuple to exactly one tile while respecting capacity constraints. The final number of servers is then computed from the induced tile loads via*

$$r^\star_{\mathcal{P}_\beta} = \min(\Delta, |\mathcal{R}^\star_{\mathcal{P}_\beta}|). \tag{3.89}$$

Next, we discuss the effect of tile capacity and threshold Δ on the performance of Algorithm 1.

Remark 4 (Effect of tile capacity and threshold Δ)**.** *The relation between the tile capacity κ_β and the threshold Δ influences the resulting server count.*

When κ_β is large relative to Δ, a tile can absorb many tuples while contributing at most Δ to the sum rank. In this regime, consolidating assignments within fewer tiles can significantly reduce the number of servers.

In contrast, when $\kappa_\beta < \Delta$, the contribution of a tile grows approximately linearly with the number of assigned tuples, and the benefits of consolidation are reduced.

These observations highlight the role of tile geometry in the achievable performance, despite Algorithm 1 is primarily designed for feasibility rather than explicit sum rank minimization.

The above information provided in Remarks 3 and 4 will be essential in enumerating our equivalence class of tiles and associating each such class with a collection of servers.

After presenting Algorithm 1, our next step is to fill the tiles in the product tensor, using the constitutive component columns and rows, as follows.

b) **Step 2: Filling tiles in $\bar{\mathcal{E}} \times_1 \mathbf{D}$ as a function of $\bar{\mathcal{F}}$.**

Having established the set of equivalence classes and the corresponding (position of the) tiles of $\bar{\mathcal{F}}$ by establishing Algorithm 1, we proceed to fill (and crop) the tiles in $\bar{\mathcal{F}}$ as follows:

$$\bar{\mathcal{F}}_{\mathcal{P}} \triangleq (\bar{\mathcal{F}} \odot \bar{\mathcal{S}}_{\mathcal{P}})(\mathcal{R}^\star_{\mathcal{P}}, \mathcal{C}_{\mathcal{P}}), \quad \forall \mathcal{P} \in \mathcal{C}. \tag{3.90}$$

The first step $\bar{\mathcal{F}} \odot \bar{\mathcal{S}}_{\mathcal{P}}$ simply fills up $\bar{\mathcal{S}}_{\mathcal{P}}$ with the corresponding entries of $\bar{\mathcal{F}}$, and the second step $(\bar{\mathcal{F}} \odot \bar{\mathcal{S}}_{\mathcal{P}})(\mathcal{R}^\star_{\mathcal{P}}, \mathcal{C}_{\mathcal{P}})$ crops these[12].

c) **Step 3: Creating and filling the non-zero tiles in $\mathbf{D}$ and $\bar{\mathcal{E}}$.**

This step starts with the MLSVD decomposition (described in Section 3.5) of the cropped tile $\bar{\mathcal{F}}_{\mathcal{P}}$ where this decomposition takes the form

$$\bar{\mathcal{F}}_{\mathcal{P}} = \bar{\mathcal{E}}_{\mathcal{P}} \times_1 \mathbf{D}_{\mathcal{P}} \tag{3.91}$$

where $\mathbf{D}_{\mathcal{P}} \in \mathbb{R}^{|\mathcal{C}_{\mathcal{P}}| \times r^\star_{\mathcal{P}}}, \bar{\mathcal{E}}_{\mathcal{P}} \in \mathbb{R}^{r^\star_{\mathcal{P}} \times |\mathcal{R}^\star_{\mathcal{P}}|}$. In particular, $\bar{\mathcal{F}}, \mathbf{D}$, and $\bar{\mathcal{E}}$ are associated to $\bar{\mathcal{T}}$, $\mathbf{U}^{(1)}$, and $\bar{\mathcal{S}}$ in all complete MLSVD decomposition of (3.44). Naturally, $\operatorname{rank}(\bar{\mathcal{F}}_{\mathcal{P}}) \leq r^\star_{\mathcal{P}}$. Let

$$\mathcal{C} \triangleq \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m\},\ m \in \mathbb{N} \tag{3.92}$$

[12]We here see a small distinction between the previously uncropped tiles, and the tiles that are cropped here. These cropped tiles will be the outcomes of an MLSVD decomposition of subtensors of $\bar{\mathcal{F}}$, yielding our SVD-based factorization.

describe the enumeration we give to each tile. Then the position that each cropped tile takes inside $\mathbf{D}$, is given by

$$\mathcal{C}_{\mathcal{P}_j},\, [\sum_{i=1}^{j-1} r^\star_{\mathcal{P}_i} + 1 : \sum_{i=1}^{j} r^\star_{\mathcal{P}_i}]\,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.93}$$

and the position of each cropped tile in $\bar{\mathcal{E}}$ is given by

$$[\sum_{i=1}^{j-1} r^\star_{\mathcal{P}_i} + 1 : \sum_{i=1}^{j} r^\star_{\mathcal{P}_i}],\, \mathcal{R}^\star_{\mathcal{P}_j}\,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.94}$$

This yields

$$\mathbf{D}\Big(\mathcal{C}_{\mathcal{P}_j},\, [\sum_{i=1}^{j-1} r^\star_{\mathcal{P}_i} + 1 : \sum_{i=1}^{j} r^\star_{\mathcal{P}_i}]\Big) = \mathbf{D}_{\mathcal{P}_j}\,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.95}$$

and

$$\bar{\mathcal{E}}\Big([\sum_{i=1}^{j-1} r^\star_{\mathcal{P}_i} + 1 : \sum_{i=1}^{j} r^\star_{\mathcal{P}_i}],\, \mathcal{R}^\star_{\mathcal{P}_j}\Big) = \bar{\mathcal{E}}_{\mathcal{P}_j}\,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.96}$$

while naturally the remaining non-assigned elements of $\mathbf{D}$ and $\bar{\mathcal{E}}$ are zero.

d) **Step 4: An upper bound to the number of servers**.

We now proceed to bound the number of rank-one contribution supports (corresponding to the number of servers), and we do so under our previously stated assumption of disjoint supports (cf. Definition 17), which is equivalent to the disjoint-tiles assumption (see Lemma 4).

We recall from Section 3.4 (see also (3.39)–(3.40)) that each server $n \in [N]$ corresponds to one column of the decoding matrix $\mathbf{D}$ and the associated row of the encoding tensor $\bar{\mathcal{E}}$. To express N in terms of the system parameters, consider the tiles $\mathcal{P} \in \mathcal{C}$ and the associated submatrices $\mathbf{D}_{\mathcal{P}}$ defined in (3.91). From (3.95), these submatrices collectively span $\sum_{i=1}^{m} r^\star_{\mathcal{P}_i}$ columns.

Therefore, from (3.89), the upper bound on the number of servers is obtained as

$$N^\star \leq \sum_{\mathcal{P}\in\mathcal{C}} r^*_{\mathcal{P}} = \frac{K}{\Delta} \sum_{\beta\in[n_{\mathcal{P}}]} \min\Big(\Delta, |\mathcal{R}^\star_{\mathcal{P}_\beta}|\Big) \tag{3.97}$$

which corresponds to (3.84). □

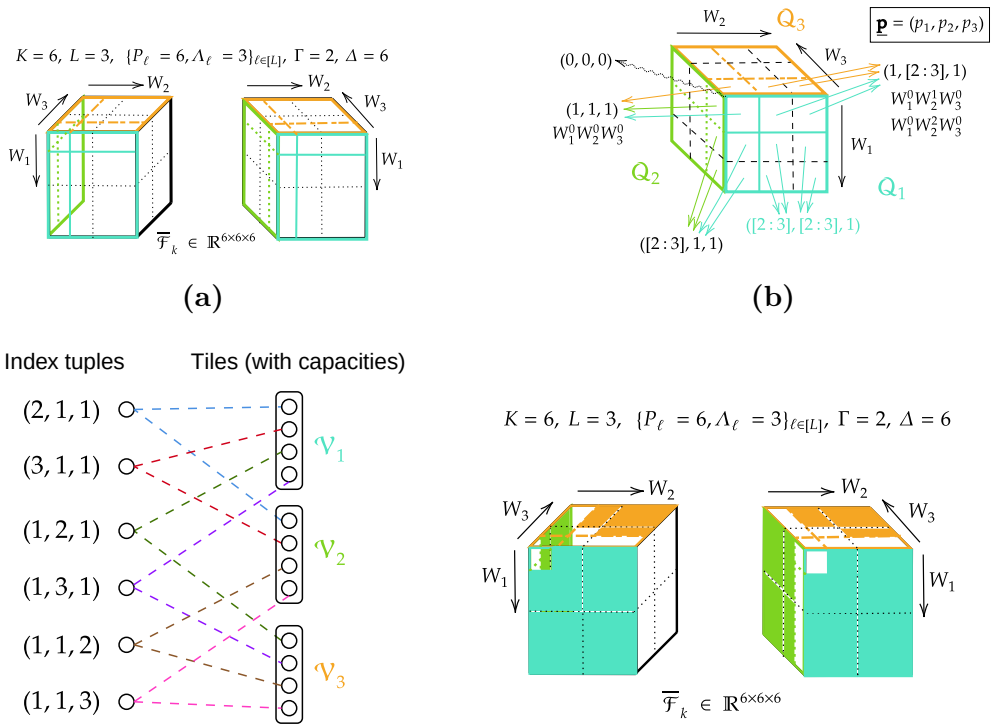

Figure 3.9: Illustration of Example 5. (a) Initial construction of Γ-dimensional tiles in $\bar{\mathcal{F}}_1$ and their combinatorial overlaps. (b) Corresponding index-tuple representation, where a single admissible tuple may belong to multiple tile closures. (c) Bipartite feasibility graph between admissible tuples and candidate tiles; each tile node represents a tile with capacity (illustrated by stacked slots corresponding to its closure size), and edges indicate feasible assignments based on closure membership. (d) Final disjoint tile assignment after applying Algorithm 1. The assignment shown is illustrative and represents one feasible outcome of the algorithm. The assignment induces disjoint sets $\mathcal{R}^{\star}_{\mathcal{P}}$, enabling the MLSVD construction. Tiles with identical colors correspond to the same active subset $\mathcal{Q}_r \subseteq [L]$ of cardinality Γ.

We next provide a comprehensive example to illustrate how the system parameters determine the tile structure, how overlaps arise among admissible tuples, and how Algorithm 1 resolves these overlaps through a unique tuple-to-tile assignment.

Example 5 (Detailed tuple-to-tile assignment example)**.** *Consider a dis-*

tributed computing setting with the parameters $(K = 6, L = 3, \{P_\ell = 6, \Lambda_\ell = 3\}_{\ell\in[L]}, \Gamma = 2, \Delta = 6)$. *To illustrate the representative supports, the induced overlaps, and the role of Algorithm 1, we focus on the demanded tensor of one user, e.g.,* 1, *denoted by* $\bar{\mathcal{F}}_1$, *shown in Figure 3.9.*

Following the protocol in the proof of Theorem 4, the Γ*-subsets of basis subfunctions are*

$$\mathcal{Q}_1 = \{1, 2\}, \qquad \mathcal{Q}_2 = \{1, 3\}, \qquad \mathcal{Q}_3 = \{2, 3\}.$$

Since $\Gamma = 2$, *every candidate tile is associated with exactly one of these three support classes.*

We detail the solution in the following steps.

a) ***Step 1: Admissible tuples****.*

Each index tuple has the form

$$\underline{\mathbf{p}} = (p_1, p_2, p_3) \in [6]^3 ,$$

and must satisfy the sparsity constraint

$$\left|supp(\mathbf{I}(\underline{\mathbf{p}}))\right| \leq 2 ,$$

that is, at most two coordinates may exceed 1. *Hence, the admissible tuple set is*

$$\mathcal{U} = \left\{\underline{\mathbf{p}} \in [6]^3 : \left|supp(\mathbf{I}(\underline{\mathbf{p}}))\right| \leq 2\right\}$$

with cardinality $|\mathcal{U}| = \sum_{d=0}^{2} \binom{3}{d}(6-1)^d = 1 + 3 \cdot 5 + 3 \cdot 25 = 91$.

b) ***Step 2: Window structure****.*

Since $P_\ell = 6$ *and* $\Lambda_\ell = 3$, *the active values* $\{2, 3, 4, 5, 6\}$ *are grouped into the windows*

$$\mathcal{W}^{(1)} = \{2, 3, 4\} , \qquad \mathcal{W}^{(2)} = \{5, 6\} .$$

Thus, for each support class $\mathcal{Q}_r$, *the active exponent coordinates are partitioned into window choices, which define the candidate tiles.*

c) ***Step 3: Tile cells versus closures****.*

Consider the support class $\mathcal{Q}_1 = \{1, 2\}$ *and the window pair* $(\mathcal{W}^{(1)}, \mathcal{W}^{(2)})$. *The corresponding geometric tile cells are the tuples with exactly two active coordinates,*

$$\mathcal{P}(\mathcal{Q}_1, \mathcal{W}^{(1)}, \mathcal{W}^{(2)}) = \left\{(p_1, p_2, p_3) \in [6]^3 : p_1 \in \mathcal{W}^{(1)},\ p_2 \in \mathcal{W}^{(2)},\ p_3 = 1\right\}.$$

These are the atomic $\Gamma = 2$*-dimensional cells displayed geometrically in Figure 3.9 (a).*

However, for assignment purposes, each tile is associated with its larger row-index set, or closure,

$$\mathcal{R}_{\mathcal{P}(\mathcal{Q}_1,\mathcal{W}^{(1)},\mathcal{W}^{(2)})} = \Big\{(p_1,p_2,p_3) \in [6]^3 : p_1 \in \{1\} \cup \mathcal{W}^{(1)},\ p_2 \in \{1\} \cup \mathcal{W}^{(2)},\ p_3 = 1\Big\}.$$

Therefore,

$$\mathcal{R}_{\mathcal{P}} = \{1,2,3,4\} \times \{1,5,6\} \times \{1\}\,,$$

and

$$|\mathcal{R}_{\mathcal{P}}| = (1+3)(1+2) = 12\,.$$

This distinction is crucial: the geometric tile contains only fully active tuples, whereas the closure contains all admissible tuples whose support is contained in the tile support and whose active entries lie in the selected windows.

d) ***Step 4: Why overlaps arise****.*

As shown in Figure 3.9 (b), overlaps arise for two reasons.

First, a lower-support tuple may be contained in multiple support classes. For instance, the tuple $(2,1,1)$ *has support* $\{1\}$*, and hence belongs to closures associated with both* $\mathcal{Q}_1 = \{1,2\}$ *and* $\mathcal{Q}_2 = \{1,3\}$*.*

Second, even within a fixed support class, an inactive coordinate equal to 1 *is compatible with several window choices. For example,*

$$(2,1,1) \in \mathcal{R}_{\mathcal{P}(\mathcal{Q}_1,\mathcal{W}^{(1)},\mathcal{W}^{(1)})} \cap \mathcal{R}_{\mathcal{P}(\mathcal{Q}_1,\mathcal{W}^{(1)},\mathcal{W}^{(2)})}\,.$$

Therefore, lower-support tuples may overlap both across support classes and across window selections. In contrast, fully active tuples are rigid. For example, the tuple $(2,5,1)$ *has support* $\{1,2\}$*, with* $2 \in \mathcal{W}^{(1)},\ 5 \in \mathcal{W}^{(2)}$*, so both its support class and its window pair are uniquely determined. Hence, it belongs to exactly one candidate tile.*

e) ***Step 5: Feasibility graph and capacities****.*

To resolve the ambiguity, we construct the tuple–tile feasibility graph of Definition 21. The left vertices correspond to admissible tuples in $\mathcal{U}$*, while the right vertices correspond to candidate tiles. An edge is*

drawn whenever the tuple belongs to the closure of the tile. Figure 3.9 (c) illustrates the resulting bipartite graph for the intersecting tiles.

Each tile node is equipped with a capacity equal to the size of its closure. In the present homogeneous setting, this capacity depends on the selected windows. For instance, for the tile above, we have $\kappa_\beta = |\mathcal{R}_{\mathcal{P}(\mathcal{Q}_1,\mathcal{W}^{(1)},\mathcal{W}^{(2)})}| = 12$.

Algorithm 1 then assigns each tuple to exactly one feasible tile while respecting these capacities.

f) ***Step 6: Induced disjoint assignment****.*

The output of Algorithm 1 is a collection of assigned sets $\mathcal{R}^\star_{\mathcal{P}_\beta} \subseteq \mathcal{R}_{\mathcal{P}_\beta}$ *such that each admissible tuple belongs to exactly one assigned set. Consequently, the resulting supports are disjoint. A feasible tessellation pattern for* $\bar{\mathcal{F}}_1$ *is illustrated in Figure 3.9 (d).*

This disjointness condition is exactly what is needed to apply the MLSVD construction described in Section 3.5.

g) ***Step 7: Induced sum rank and server count****.*

Once the assignment is fixed, the contribution of each tile is $r^\star_{\mathcal{P}_\beta} = \min\big(\Delta, |\mathcal{R}^\star_{\mathcal{P}_\beta}|\big)$*. Hence, the per-user sum rank is* $\sum_{\beta\in[n_{\mathcal{P}}]} \min\big(\Delta, |\mathcal{R}^\star_{\mathcal{P}_\beta}|\big)$*, and the total number of required servers is obtained by scaling this quantity by* K/Δ*, as in Theorem 4.*

For this example, applying Algorithm 1 yields a feasible disjoint assignment whose induced tile loads have sum rank

$$N_{\text{alg}} = \sum_{\beta} \min\big(\Delta, |\mathcal{R}^\star_{\mathcal{P}_\beta}|\big) = 66\,.$$

Since $K = \Delta = 6$*, the scaling factor* K/Δ *is equal to* 1*, and thus the total number of required servers is* $N = 66$*.*

For comparison, the default tensor factorization of Theorem 3 gives

$$N_{\text{ach}} = \frac{K}{\Delta}\binom{L}{\Gamma}\left(\frac{P}{\Lambda}\right)^{\Gamma} \min(\Delta, \Lambda^{\Gamma}) = 72\,.$$

Consequently, the assignment-based construction achieves a reduction of 6 *servers in this example.*

We next examine the computational complexity of the proposed assignment algorithm (cf. Algorithm 1) for tiling designs.

Remark 5. *The complexity of the proposed algorithm is dominated by the construction of the tuple–tile feasibility graph and the repeated solution of capacitated bipartite assignment problems via maximum flow.*

The bipartite graph consists of $n_{\mathcal{U}}$ left vertices (admissible tuples) and $n_{\mathcal{P}}$ right vertices (candidate tiles). Each edge corresponds to a feasible assignment, i.e., a tuple belonging to the closure of a tile. Hence, the total number of edges is

$$|\mathcal{E}| = \sum_{\alpha \in [n_{\mathcal{U}}]} \Big| \{\beta : \underline{\mathbf{p}}_\alpha \in \mathcal{R}_{\mathcal{P}_\beta}\} \Big|.$$

Each iteration of the algorithm solves a capacitated bipartite assignment problem, which can be formulated as a maximum flow instance. Using standard reductions (e.g., node-splitting to handle capacities [155]), this can be solved in

$$\mathcal{O}\Big(|\mathcal{E}|\sqrt{|\mathcal{U}| + |\mathcal{V}|}\Big), \quad \textit{where} \quad |\mathcal{U}| = n_{\mathcal{U}} \quad \textit{and} \quad |\mathcal{V}| = n_{\mathcal{P}}$$

by classical algorithms for bipartite matching, such as the Hopcroft–Karp algorithm [154].

In practice, the algorithm typically converges in a small number of iterations. When the global assignment is feasible, a single flow computation suffices. Otherwise, the algorithm performs a sequence of residual updates, each assigning at least one tuple and strictly reducing the number of remaining tuples. The total complexity is therefore bounded by $\mathcal{O}\Big(N_{\text{iter}} \cdot |\mathcal{E}|\sqrt{n_{\mathcal{U}} + n_{\mathcal{P}}}\Big)$, where $N_{\text{iter}} \leq n_{\mathcal{U}}$. While this bound is linear in the number of iterations in the worst case, empirical behavior is significantly more favorable, with N_{iter} typically remaining small.

In regimes where tile capacities are sufficiently large relative to the number of admissible tuples, the global assignment succeeds in a single iteration, yielding overall complexity $\mathcal{O}\Big(|\mathcal{E}|\sqrt{n_{\mathcal{U}} + n_{\mathcal{P}}}\Big)$. In contrast, in highly constrained regimes, multiple iterations may be required. However, each iteration strictly reduces the number of remaining tuples, guaranteeing termination in at most $n_{\mathcal{U}}$ steps.

Finally, in the homogeneous setting with $\{P_\ell = P, \Lambda_\ell = \Lambda\}_{\ell \in [L]}$, the number of edges satisfies $|\mathcal{E}| = \mathcal{O}(n_{\mathcal{U}} \Lambda^\Gamma)$ since each tuple is compatible with multiple tiles across support configurations and sliding windows.

3.7 Numerical Analysis and Evaluation of Algorithm 1

In this section, we provide numerical evaluations of the required number of servers for different system configurations using Algorithm 1, and compare

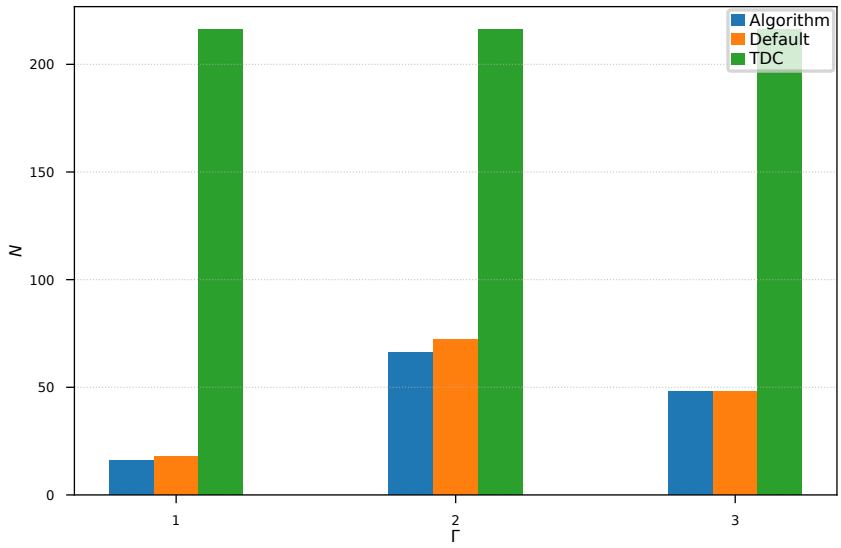

Figure 3.10: Comparison of the required number of servers N for the lossless distributed computing setting $(K = 6, L = 3, \Gamma, \Delta = 6, \{P_\ell = 6, \Lambda_\ell = 3\}_{\ell \in [L]})$. The bars compare the proposed algorithmic achievable scheme, the default tensor factorization scheme, and the TDC baseline as functions of the computation cost Γ.

the results with the default tensor factorization scheme of Theorem 3 and the sparse matrix factorization of the TDC scheme [105].

The implementation of the proposed sparse tensor factorization and assignment-based combinatorial reduction used for these numerical evaluations is publicly available in the accompanying software repository [156].

We consider homogeneous lossless systems with parameters $(K, \Delta, L, \Gamma, P, \Lambda)$, where $K = \Delta$ is considered in the first part of the simulation so that the total number of required servers coincides with the induced sum rank per block. The performance is evaluated as a function of the computation cost Γ for two representative cases $L = 3$ and $L = 5$. The corresponding results are illustrated in Figure 3.10 and Figure 3.11. In particular, the gap between the proposed scheme and the default tensor factorization increases with L, highlighting the growing importance of assignment in higher-dimensional settings.

We next discuss the results shown in Figures 3.10 and 3.11, which illustrate the required number of servers N as a function of the computation cost Γ under the lossless setting. In both figures, we consider homogeneous parameters with $P_\ell = P$ and $\Lambda_\ell = \Lambda$ for all $\ell \in [L]$, and we compare the proposed assignment-based scheme (Algorithm 1) with the default tensor factorization scheme of

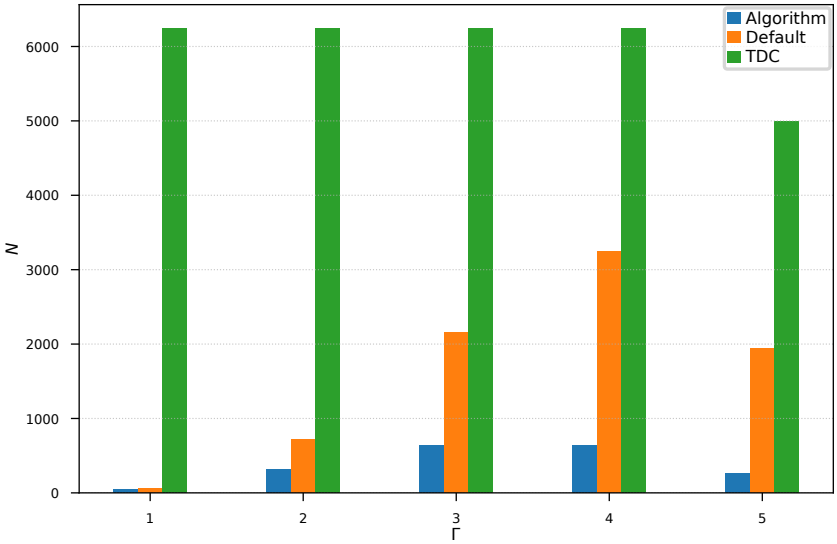

Figure 3.11: Comparison of the required number of servers N for the lossless distributed computing setting $(K = 8, L = 5, \Gamma, \Delta = 4, \{P_\ell = 5, \Lambda_\ell = 2\}_{\ell \in [L]})$. The bars compare the proposed algorithmic achievable scheme, the default tensor factorization scheme, and the TDC baseline with respect to the computation cost Γ.

Theorem 3 and the TDC baseline [105]. We next detail the comparisons.

Moderate dimensionality $(L = 3)$. In this regime, as shown in Figure 3.10, the number of admissible tuples grows relatively slowly with Γ, and all schemes exhibit a moderate increase in N as Γ increases. However, the proposed assignment-based scheme consistently outperforms the default tensor factorization scheme of Theorem 3 by eliminating overlap among subtensors. This gain is particularly visible for intermediate values of Γ, where multiple tiles share common tuples. By enforcing a disjoint assignment, Algorithm 1 avoids redundant counting and achieves a smaller sum rank of the resulting subtensors. In contrast, the TDC scheme [105] yields significantly larger values of N across all Γ since it does not exploit the multiplicative structure of the underlying functions and instead treats the problem with much larger dimensionality to capture all multiplicative terms, leading to inefficient use of computational resources.

Higher dimensionality $(L = 5)$. The impact of the assignment step becomes more pronounced as the dimension increases, as illustrated in Figure 3.11. In this setting, the number of admissible tuples grows combinatorially with Γ, which leads to a rapid increase in the required number of servers for the default tensor factorization scheme of Theorem 3. This growth is due

to the increasing overlap between high-dimensional subtensors, which are treated independently in the default construction in Theorem 3. The proposed assignment-based scheme mitigates this effect by consolidating overlapping tuples into a reduced set of tiles. As a result, the growth of N is significantly limited compared to the default scheme, especially for $\Gamma = 3$ and $\Gamma = 4$, where overlap is most pronounced. Interestingly, for $\Gamma = L$, a slight reduction in N can be observed. This is due to the fact that higher-dimensional tiles allow more efficient packing of tuples, reducing the number of active tiles required. Such behavior is not captured by the default scheme of Theorem 3, which does not perform any global assignment. Furthermore, similarly as in Figure 3.10, the TDC baseline [105] remains significantly above both tensor-based schemes and shows limited sensitivity to Γ, reflecting its failure to leverage the underlying tensor structure.

We next present all the details for the multi-shot scenario with $T > 1$ shots, which generalizes the results derived for the single-shot scenario.

3.8 Multi-Shot Scenario with $T > 1$

In this section, we study the multi-shot scenario with T shots in which each server can compute the assigned tasks and transmit to the users during T shots, allowing for a reduced number of required servers, compared to the single-shot scenario. We follow the same procedure as in the single-shot scenario and highlight the differences in the following.

3.8.1 System Model

In the multi-shot scenario with T shots, we will have $e_{n,\underline{\mathbf{p}},t}$ and $z_{n,t}$ instead of $e_{n,\underline{\mathbf{p}}}$ and z_n in (3.11), (3.12), respectively, as well as $d_{k,n,t}$ instead of $d_{k,n}$ in (3.12), allowing for utilizing the servers by operating both computation and communication phases across multiple shots $t \in [T]$. These yield

$$z_{n,t} \triangleq \sum\nolimits_{\underline{\mathbf{p}} \in \mathcal{P}_n} e_{n,\underline{\mathbf{p}},t} \prod\nolimits_{\ell \in [L]} W_\ell^{p_\ell - 1}, \quad n \in [N]\,,\, t \in [T] \tag{3.98}$$

and

$$F'_k \triangleq \sum\nolimits_{n \in [N]} d_{k,n,t} z_{n,t}\,. \tag{3.99}$$

The system will be parametrized by $(K, N, T, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell \in [L]})$ for the multi-shot scenario with T shots. Therefore, for T shots, (3.17), from [105, Theorem 1], will take the form

$$N = \frac{K}{\Delta} \frac{L'}{\Gamma} \frac{\min(\Delta, \Gamma)}{T}\,. \tag{3.100}$$

3.8.2 Problem Formulation in Tensor Form

For the multi-shot scenario with $T > 1$, we have the following differences in problem formulation details.

In the communication phase, from (3.98), the transmission from server n during time $t \in [T]$ takes the form $z_{n,t} = \bar{\boldsymbol{e}}_{n,t} \times_{[[1:L]]}^{[[1:L]]} \bar{\mathcal{W}}$, $z_{n,t} \in \mathbb{R}$, where for all $\underline{\mathbf{p}} \in \prod_{\ell \in [L]} [P_\ell]$, the encoding coefficients $e_{n,\underline{\mathbf{p}},t}$ yield a tensor of the form

$$\bar{\boldsymbol{e}}_{n,t}(\underline{\mathbf{p}}) \triangleq e_{n,\underline{\mathbf{p}},t},\ \bar{\boldsymbol{e}}_{n,t} \in \mathbb{R}^{P_1 \times \ldots \times P_L},\ n \in [N],\ t \in [T],\ \underline{\mathbf{p}} \in \mathcal{P}_n \tag{3.101}$$

thus allowing us to represent all the per-shot encoding tensors in a general tensor of the form

$$\bar{\mathcal{E}}_n(\underline{\mathbf{p}}) \triangleq stack_1(\bar{\boldsymbol{e}}_{n,1}, \ldots, \bar{\boldsymbol{e}}_{n,T}) \in \mathbb{R}^{T \times P_1 \times \ldots \times P_L},\ n \in [N] \tag{3.102}$$

where $stack_1(\bar{\boldsymbol{e}}_{n,1}, \ldots, \bar{\boldsymbol{e}}_{n,T})$ forms an order-$(L+1)$ tensor $\bar{\mathcal{E}}_n \in \mathbb{R}^{T \times P_1 \times \ldots \times P_L}$ composed by stacking the T tensors $\{\bar{\boldsymbol{e}}_{n,t}\}_{t \in [T]}$ (each of order L) along a new first mode. thus yielding the per-server transmission vector $\boldsymbol{z}_n \triangleq [z_{n,1}, \ldots, z_{n,T}]^\intercal$, obtained as

$$\boldsymbol{z}_n = \bar{\mathcal{E}}_n \times_{[[2:L+1]]}^{[[1:L]]} \bar{\mathcal{W}},\ \boldsymbol{z}_n \in \mathbb{R}^T,\ n \in [N] \tag{3.103}$$

and the overall transmitted vector $\boldsymbol{z} \triangleq [\boldsymbol{z}_1^\intercal, \boldsymbol{z}_2^\intercal, \ldots, \boldsymbol{z}_N^\intercal]^\intercal \in \mathbb{R}^{NT}$, taking the form

$$\boldsymbol{z} = [\bar{\mathcal{E}}_1, \ldots, \bar{\mathcal{E}}_N]_1 \times_{[[2:L+1]]}^{[[1:L]]} \bar{\mathcal{W}} \tag{3.104}$$

where $[\bar{\mathcal{E}}_1, \ldots, \bar{\mathcal{E}}_N]_1$ forms an order-$(L+1)$ tensor $\bar{\mathcal{E}} \in \mathbb{R}^{NT \times P_1 \times \ldots \times P_L}$ composed by concatenating the N tensors $\{\bar{\mathcal{E}}_n\}_{n \in [N]}$ (each of order $(L+1)$) along their first mode.

In the decoding phase, similarly, each retrieved function output takes the form $F'_k = \mathbf{d}_k^\intercal \mathbf{z} \in \mathbb{R}$, thus resulting in the vector of all outputs taking the form

$$\mathbf{f}' \triangleq [F'_1, F'_2, \ldots, F'_K] = [\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_K]^\intercal \mathbf{z} \in \mathbb{R}^K \tag{3.105}$$

where the decoding coefficients $d_{k,n,t}$ from (3.99) lead to

$$\mathbf{d}_{k,n} \triangleq [d_{k,n,1}, \ldots, d_{k,n,T}]^\intercal \in \mathbb{R}^T,\ k \in [K]\,,\ n \in [N]\,, \tag{3.106}$$

$$\mathbf{d}_k \triangleq [\mathbf{d}_{k,1}^\intercal, \ldots, \mathbf{d}_{k,N}^\intercal]^\intercal \in \mathbb{R}^{NT},\ k \in [K]\,. \tag{3.107}$$

The tensors of encoding coefficients in (3.102), and the vector of decoding coefficients in (3.107) allow us to form the respective tensors

$$\bar{\mathcal{E}} \triangleq [\bar{\mathcal{E}}_1, \ldots, \bar{\mathcal{E}}_N]_1 \in \mathbb{R}^{NT \times P_1 \times P_2 \times \ldots \times P_L}, \tag{3.108}$$

$$\mathbf{D} \triangleq [\mathbf{d}_1, \ldots, \mathbf{d}_K]^\intercal \in \mathbb{R}^{K \times NT} \,. \tag{3.109}$$

In terms of the corresponding connection to the sparsity of $\mathbf{D}$ and $\bar{\mathcal{E}}$, we recall from (3.6), our metric Γ, which directly from (3.102) and (3.108), implies the computation constraint as

$$\max_{n\in[N]} \sum_{\ell\in[L]} \left| \mathbb{1}\Big(\cup_{t=1}^{T} \operatorname{supp}\big(\bar{\mathcal{E}}((n-1)T+t, \underbrace{:, \ldots, :}_{\ell-1 \text{ terms}}, p_\ell, \underbrace{:, \ldots, :}_{L-\ell \text{ terms}})\big) \neq \emptyset \Big) \right| \leq \Gamma \,, \quad p_\ell \in [2 : P_\ell] \,.$$

Furthermore, from (3.13), we recall Δ, which from (3.107) and (3.109), implies a communication constraint

$$\max_{n\in[N]} \left| \cup_{t=1}^{T} \operatorname{supp}\Big(\mathbf{D}(:, (n-1)T+t)\Big) \right| \leq \Delta \,.$$

Finally from (3.10), we recall Λ_ℓ, which from (3.102) and (3.108), suggests the multiplication constraints

$$\left| \cup_{t=1}^{T} \, \| \bar{\mathcal{E}}\Big((n-1)T+t, p_1, p_2, \ldots, p_{\ell-1}, :, p_{\ell+1}, \ldots, p_L\Big) \|_0 \right| \leq \Lambda_\ell \,, \quad \ell \in [L],\, p_\ell \in [P_\ell] \,.$$

3.8.3 Main Results

Default Tensor Factorization For the multi-shot scenario with T shots, Theorem 3 will reform as follows.

Theorem 5 (Achievable number of servers and system rate in the multi-shot setting)**.** *Consider the lossless* $\big(K, N, T, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell\in[L]}\big)$ *distributed computing system under the homogeneous setting* $P_\ell = P$ *and* $\Lambda_\ell = \Lambda,\ \forall\, \ell \in [L]$*, with* $\Delta \mid K$ *and* $\Lambda \mid P$*. There exists a lossless* T*-shot distributed computing scheme using*

$$N_{\text{ach}} \triangleq \frac{K}{\Delta} \binom{L}{\Gamma} \left\lceil \frac{\min\{\Delta, \Lambda^\Gamma\}}{T} \right\rceil \left(\frac{P}{\Lambda}\right)^\Gamma \tag{3.110}$$

servers. Consequently, if $N^\star$ *denotes the minimum number of servers required in the* T*-shot setting, then*

$$N^\star \leq N_{\text{ach}}. \tag{3.111}$$

The corresponding rate

$$R_{\text{ach}} \triangleq \frac{K}{N_{\text{ach}}} \tag{3.112}$$

is therefore achievable. Hence, the maximum achievable T-shot system rate $R^\star$ satisfies

$$R^\star = \frac{K}{N^\star} \geq R_{\text{ach}} = \frac{\Delta}{\binom{L}{\Gamma}\left\lceil \frac{\min\{\Delta, \Lambda^\Gamma\}}{T} \right\rceil \left(\frac{P}{\Lambda}\right)^\Gamma}. \tag{3.113}$$

Proof. The proof follows the same construction as in the proof of Theorem 3. In the T-shot setting, a single server can contribute at most T rank dimensions to a given tile. Therefore, a tile whose mode-1 unfolding has rank $r_{\mathcal{P}}$, during T-shots, requires $\left\lceil \frac{r_{\mathcal{P}}}{T} \right\rceil$ servers. Substituting

$$r_{\mathcal{P}} = \min\{\Delta, \Lambda^\Gamma\}$$

and summing over all tiles yields (3.110). □

Assignment-based Algorithmic Tensor Factorization For the multi-shot scenario with T shots, instead of Theorem 4, we will have the following theorem.

Theorem 6 (Achievable scheme via capacitated tile assignment in the multi--shot setting)**.** *Consider the lossless $\left(K, N, T, L, \Gamma, \Delta, \{P_\ell, \Lambda_\ell\}_{\ell\in[L]}\right)$ distributed computing system under the homogeneous setting $P_\ell = P$ and $\Lambda_\ell = \Lambda,\ \forall\, \ell \in [L]$, with $\Delta \mid K$ and $\Lambda \mid P$. Algorithm 1 constructs a feasible assignment of the admissible tuples to the candidate tiles. The resulting disjoint tile assignment induces a lossless T-shot distributed computing scheme using*

$$N_{\text{alg}} \triangleq \frac{K}{\Delta} \sum_{\beta\in[n_{\mathcal{P}}]} \left\lceil \frac{\min\left\{\Delta, \left|\mathcal{R}^\star_{\mathcal{P}_\beta}\right|\right\}}{T} \right\rceil \tag{3.114}$$

servers, where $\mathcal{R}^\star_{\mathcal{P}_\beta}$ denotes the disjoint row-index set assigned to tile $\mathcal{P}_\beta$ by Algorithm 1. Consequently, if $N^\star$ denotes the minimum number of servers required in the T-shot setting, then

$$N^\star \leq N_{\text{alg}}. \tag{3.115}$$

The corresponding rate

$$R_{\text{alg}} \triangleq \frac{K}{N_{\text{alg}}} \tag{3.116}$$

is therefore achievable. Hence, the maximum achievable T-shot system rate $R^\star$ satisfies

$$R^\star = \frac{K}{N^\star} \geq R_{\text{alg}} = \frac{K}{N_{\text{alg}}}. \tag{3.117}$$

Proof. The proof follows from the same parts and steps as the proof of Theorem 4 of the main document. However, in Part I, the notion of rank-one contribution support and the related concepts are extended to the NT server-shot components, indexed by $(n-1)T+t$, for $n \in [N]$ and $t \in [T]$. Each physical server thus contributes T such components, corresponding to the T shots.

Part I and Steps a) and b) in Part II, including Algorithm 1 will remain the same, but Steps c) and d) in Part II will incorporate the number of shots T to design tiles in $\mathbf{D}$ and $\bar{\mathcal{E}}$ and enumerating the number of required servers, which we detail next.

c) **Step 3: Creating and filling the non-zero tiles in D and $\bar{\mathcal{E}}$.**

For $T = 1$, each tile $\mathcal{P}_i$ of rank $r^\star_{\mathcal{P}_i}$ is assigned to $r^\star_{\mathcal{P}_i}$ distinct servers, each contributing one rank dimension. For $T > 1$, a server may contribute up to T rank dimensions within the same tile. Since $r^\star_{\mathcal{P}_i}$ may not be divisible by T, at most one server per tile may be partially utilized. Associating such a server with additional tiles may violate the communication, computation, or multiplication constraints. Therefore, we allocate $\left\lceil \frac{r^\star_{\mathcal{P}}}{T} \right\rceil$ servers to each equivalence class of tiles. Therefore, the position that each cropped tile takes inside $\mathbf{D}$, is given by

$$\mathcal{C}_{\mathcal{P}_j}, \, [\sum_{i=1}^{j-1} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil + 1 : \sum_{i=1}^{j} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil] \,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.118}$$

and the position of each cropped tile in $\bar{\mathcal{E}}$ is given by

$$[\sum_{i=1}^{j-1} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil + 1 : \sum_{i=1}^{j} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil], \, \mathcal{R}^\star_{\mathcal{P}_j} \,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.119}$$

This yields

$$\mathbf{D}\Big(\mathcal{C}_{\mathcal{P}_j}, \, [\sum_{i=1}^{j-1} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil + 1 : \sum_{i=1}^{j} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil]\Big) = \mathbf{D}_{\mathcal{P}_j} \,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.120}$$

and

$$\bar{\mathcal{E}}\Big([\sum_{i=1}^{j-1} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil + 1 : \sum_{i=1}^{j} T\lceil \frac{r^\star_{\mathcal{P}_i}}{T} \rceil], \, \mathcal{R}^\star_{\mathcal{P}_j}\Big) = \bar{\mathcal{E}}_{\mathcal{P}_j} \,, \quad \forall \mathcal{P}_j \in \mathcal{C} \tag{3.121}$$

while naturally the remaining non-assigned elements of $\mathbf{D}$ and $\bar{\mathcal{E}}$ are zero.

d) **Step 4: An upper bound to the number of servers**.

We now proceed to bound the number of rank-one contribution supports in T shots (corresponding to the number of servers), and we do so under our previously stated assumption of disjoint supports by substituting N with NT in Definition 17, which is equivalent to the disjoint-tiles assumption again by substituting N with NT in Lemma 4. We then recall from Section 3.8.2 (see also (3.108)–(3.109)) that each server–shot pair $(n, t) \in [N] \times [T]$ corresponds to one column of the decoding matrix $\mathbf{D}$ and the associated row of the encoding tensor $\bar{\mathcal{E}}$. To express N in terms of the scheme parameters, consider the tiles $\mathcal{P} \in \mathcal{C}$ and the associated submatrices $\mathbf{D}_{\mathcal{P}}$ defined in (3.91). From (3.118)–(3.121), each tile $\mathcal{P}_i$ occupies $T \left\lceil r^{\star}_{\mathcal{P}_i}/T \right\rceil$ server-shot components, corresponding to $\left\lceil r^{\star}_{\mathcal{P}_i}/T \right\rceil$ physical servers. Hence, summing over the non-empty tiles gives

$$N^{\star} \leq \sum_{\mathcal{P} \in \mathcal{C}} \left\lceil \frac{r^{\star}_{\mathcal{P}}}{T} \right\rceil . \tag{3.122}$$

Finally, from (3.89) and considering the multi-shot scenario with T transmissions, the upper bound on the number of servers is expressed as

$$N^{\star} \leq N_{\text{alg}} = \sum_{\mathcal{P} \in \mathcal{C}} \lceil \frac{r^{*}_{\mathcal{P}}}{T} \rceil = \frac{K}{\Delta} \sum_{\beta \in [n_{\mathcal{P}}]} \left\lceil \frac{\min\left(\Delta, |\mathcal{R}^{\star}_{\mathcal{P}_\beta}|\right)}{T} \right\rceil \tag{3.123}$$

which completes the proof for Theorem 6. □

3.9 Discussion and Conclusions

In this chapter, we studied lossless multi-user distributed computation of real-valued multivariate polynomials in a system with K users and N servers. We represented the users' demands through structured tensors and reformulated the computation problem as a sparse tensor factorization. Based on this formulation, we developed a new achievable scheme and characterized the achievable system rate $R_{\text{ach}} = \frac{K}{N_{\text{ach}}}$, as stated in Theorem 3. The resulting characterization reveals how the computation budget Γ, communication budget Δ, and multiplication budget Λ jointly determine the system performance. By exploiting the interaction among these constraints, the proposed scheme reduces the number of servers required for lossless recovery of the user demands. We further introduced the combinatorial assignment procedure in

Algorithm 1 to eliminate redundant computations imposed by overlapping subtensors. Theorem 4 establishes the resulting improved achievable rate, which is tighter than that of Theorem 3.

From a performance viewpoint, the proposed scheme improves on the existing alternative solutions in two main aspects. First, unlike the TDC framework of [105], it directly exploits the multiplicative structure of the requested functions rather than treating each monomial as an independent effective subfunction. Second, compared to the default tensor factorization construction, it resolves overlaps among the tensor components through a consistent assignment rule and thereby avoids redundant computations. The numerical results show that these structural gains translate into a significant reduction in the required number of servers.

Several questions remain open for future investigation. A natural extension is to study the lossy setting, in which users recover approximations of their requested functions. This setting would introduce new tradeoffs between the recovery accuracy and the computation, communication, and multiplication constraints, represented by Γ, Δ, and Λ, respectively. Another important direction is to derive converse bounds that characterize the fundamental performance limits of the proposed distributed computing model.

Chapter 4

Straggler-Resilient Coded Distributed Matrix Multiplication

In this chapter, we investigate coding strategies for straggler-resilient DMM. Our objective is to mitigate the latency caused by slow or unresponsive worker nodes while enabling efficient recovery of the desired matrix product. In large-scale distributed systems, the overall computation time is often determined by the slowest workers, making straggler mitigation a central challenge.

We consider a two-source–workers–receiver architecture in which the two source nodes separately observe the input matrices and construct coded descriptions of their respective data. These descriptions are distributed among multiple workers, which independently perform their assigned computations and return the resulting intermediate values.

The proposed coding scheme allows the receiver to recover the desired matrix product from a sufficiently large subset of worker responses, without waiting for all workers to complete their computations. Its construction exploits the algebraic structure of matrix multiplication while controlling the computation assigned to each worker, the amount of communicated data, and the achievable recovery threshold.

Relation to the author's prior publications and to the underlying structured coding framework. This chapter is mainly based on the author's joint work in [108], together with the preliminary works in [106], [107]. The structured source coding mechanisms employed in this chapter build on the prior distributed structured matrix computation frameworks developed in [109], [110].

The main contribution considered in this chapter is the DSPolyDot con-

struction developed in [108], which combines these structured source coding techniques with PolyDot-style polynomial redundancy across worker nodes and studies the resulting straggler-resilient recovery mechanism. The work in [108] was originally motivated by cooperative coded matrix multiplication in vehicular networks. For the purposes of this thesis, the presentation is reformulated as a general two-source–workers–receiver DMM problem, with emphasis on the recovery threshold, worker storage, communication cost, computation cost, and the structural assumptions required by the construction.

4.1 Introduction

Modern distributed computing systems increasingly rely on computationally intensive workloads, arising in edge and cloud computing [31], [157], [158], large-scale estimation and learning [16], [87], [129], [159], [160], and data-intensive signal processing. Many of these applications reduce to large matrix multiplications that must be completed under stringent delays. To accelerate these computations, DMM partitions the overall task among multiple worker nodes operating in parallel. A central master node encodes and distributes submatrices to the workers, which perform the assigned computations and return intermediate results. The master then combines the returned results to recover the desired matrix product. Although parallelization can substantially reduce the computation time, its performance is often limited by stragglers, namely workers that respond significantly more slowly than the others or fail to return their results. This motivates the development of coding schemes that introduce structured redundancy and enable recovery without waiting for every assigned computation.

In DMM, a master node partitions a large matrix product into smaller subcomputations and assigns them to multiple worker nodes for parallel execution. However, differences in processing speed, communication delay, and node availability generally lead to nonuniform response times. Some workers may therefore respond considerably later than others or fail to return their results altogether, thereby acting as stragglers. In a conventional uncoded implementation, the master may need to wait for all assigned subcomputations before reconstructing the desired matrix product. Consequently, even a small number of slow workers can dominate the overall completion time and substantially reduce the benefits of parallel processing.

Coded distributed computing mitigates this bottleneck by introducing structured redundancy into the assigned computations. The master first encodes the input matrices and distributes coded submatrices among the workers. Each worker independently computes an intermediate matrix product

and returns its result to the master. By appropriately designing the encoding and decoding procedures, the master can recover the desired matrix product from only a sufficiently large subset of worker responses, without waiting for the slowest workers. The recovery threshold, defined as the minimum number of worker results required for successful decoding, therefore becomes a key measure of the straggler resilience and latency performance of the system.

Several coded computation frameworks have been developed to reduce the latency caused by straggling workers, including convolutional coding approaches [92] and polynomial-based constructions for DMM [70], [71], [95], [111], [112], [161]. Prominent examples include polynomial (Poly) codes [95], [111], [162] and PolyDot codes [112]. These schemes encode the matrix partitions through linear combinations whose coefficients are generated using polynomial evaluations, thereby enabling the master to reconstruct the desired product from a subset of the returned computations.

Recent studies have extended coded DMM along several complementary directions. To accommodate large worker populations over small finite fields, algebraic function fields have been used to generalize polynomial and MatDot codes, yielding asymptotically near-optimal recovery thresholds while preserving the per-worker computation order of the classical constructions [163]. Local expansions of algebraic functions have further enabled algebraic-geometry variants of polynomial, MatDot, and PolyDot codes, including an algebraic-geometry-based PolyDot construction with improved recovery thresholds over earlier algebraic-geometry approaches and comparable communication costs [164]. Coded schemes for both single and batch matrix multiplication have also been developed over Galois rings using reverse multiplication-friendly embeddings, extending coded computation to small finite fields and residue rings relevant to finite-precision implementations [165].

From a partitioning and system-design perspective, grouped systematic MatDot codes combine systematic encoding with three-dimensional matrix partitioning to reduce completion time and memory consumption while retaining exact recovery [166]. In parallel, one-sided slicing provides a flexible implementation framework that supports different combinations of matrix partitioning and replication without requiring a distinct multiplication algorithm for each data layout [167]. Beyond the multiplication of two matrices, multivariate polynomial codes have been proposed for distributed matrix-chain multiplication, revealing tradeoffs between worker-side computation and the storage overhead associated with univariate extensions [168]. Related work on distributed batch matrix multiplication has also characterized the tradeoffs among download rate, encoder randomness, and controlled information leakage when one input matrix must remain private from colluding workers [169]. Collectively, these studies broaden coded matrix multiplica-

tion through richer algebraic fields, partitioning methods, implementation frameworks, and computation models.

Classical coded DMM constructions, including polynomial, MatDot, and PolyDot codes, introduce redundancy across worker computations through coded matrix partitions and polynomial interpolation, with different partitioning choices leading to different tradeoffs among recovery threshold, storage, communication, and computation costs [95], [112]. More recent extensions broaden this coded computation framework through richer algebraic fields, alternative partitioning and implementation strategies, matrix-chain and batch computation models, and privacy constraints [98], [136], [165]. These works are discussed here to position the worker-side polynomial redundancy used in this chapter within the broader coded-DMM literature.

The source-side structured coding component used in this chapter, on the other hand, builds on the prior structured matrix computation frameworks developed in [109], [110]. In particular, these works exploit structured source-side transformations for the distributed computation of matrix products. Building on this prior framework, Malak subsequently studied structured polynomial (StPolyDot) codes in [170], where structured linear encoding is combined with the polynomial-code framework in a source–workers–receiver setting, and the resulting memory and communication performance is characterized.

While closely related through their common structured coding ingredients, the present chapter is based on the joint work in [108] and considers the DSPolyDot construction developed therein. This construction combines the structured source-coding stage with PolyDot-style polynomial redundancy across workers and is studied here in the context of straggler-resilient DMM, with emphasis on its recovery threshold and the associated communication and computation costs. Accordingly, the contribution considered in this chapter lies in the particular DSPolyDot construction and its straggler-resilient implementation and analysis, rather than in the underlying structured source coding principle.

Our main contributions are summarized as follows.

- *Distributed integration of prior structured source coding with PolyDot redundancy:* We consider distributed computation of a sequence of matrix products generated by two source nodes. The first source observes the length-n matrix sequence $\{\mathbf{A}_i\}_{i\in[n]}$, while the second source observes $\{\mathbf{B}_i\}_{i\in[n]}$, where $[n] \triangleq \{1,\dots,n\}$. The matrix pairs $\{(\mathbf{A}_i,\mathbf{B}_i)\}_{i\in[n]}$ are independently and identically distributed (i.i.d.) over a finite field according to a joint distribution. For each $i \in [n]$, the matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ may be statistically dependent, whereas matrix pairs corresponding to different indices are independent.

Building on the prior structured source coding frameworks for distributed matrix computation in [109], [110], which apply separate nonlinear transformations to the two source matrices, and combining these ideas with the PolyDot-code construction [112], we consider the DSPolyDot code construction for straggler-resilient DMM. The construction applies to source matrix pairs satisfying the blockwise cross-product symmetry condition specified in Section 4.4, which is required for the worker-side post-processing operation to recover valid evaluations of the desired matrix product. Under this condition, the proposed scheme preserves the distributed nature of the two sources while introducing coded redundancy across the worker computations, thereby providing resilience against straggling workers.

- *Straggler-resilient recovery of matrix products:* The proposed DSPolyDot codes enable the receiver to reconstruct each desired product $\mathbf{A}_i^\intercal \mathbf{B}_i$ from a sufficiently large subset of the worker responses. Consequently, the receiver does not need to wait for every assigned subcomputation and can complete the decoding process despite slow or unavailable workers. We characterize the recovery threshold requirement of the proposed construction and evaluate the resulting tradeoffs among recovery threshold, storage load, computation load, and communication cost. This analysis shows how the structured encoding stage can be combined with polynomial coding to provide an alternative straggler mitigation mechanism for DMM.

- *Distributed implementation with bounded worker storage:* The proposed construction inherits the distributed structure of the linear coding method in [50]. The two source nodes separately transform and encode their respective matrix sequences without exchanging the uncoded source matrices. The coded matrix blocks are then assigned to worker nodes, each subject to a bounded storage and computation capability. This design supports practical two-source–worker–receiver implementations in which the source-side encoding, worker-side matrix multiplication, and receiver-side decoding are carried out as separate stages.

Organization. The remainder of this chapter is organized as follows. Section 4.2 introduces the DMM model considered in this chapter. Section 4.3 provides a detailed review of the Körner–Marton encoding scheme [50]. Section 4.4 describes the construction of the proposed DSPolyDot codes. Section 4.5 presents the main theoretical results. Section 4.6 provides the numerical evaluations, and Section 4.7 concludes the chapter.

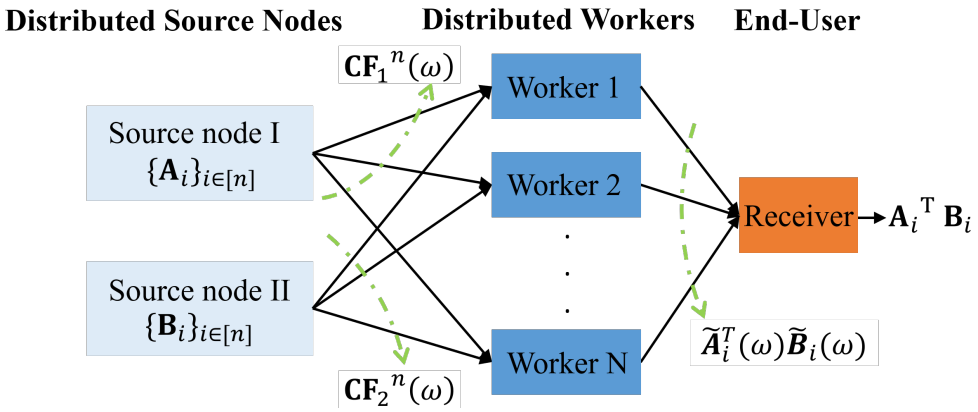

Figure 4.1: DMM framework considered in this chapter. For each worker $\omega \in \Omega$, the source nodes encode the length-n vectors $F_1^n(\omega)$ and $F_2^n(\omega)$ using a common linear encoding matrix $\mathbf{C}$, following the Körner–Marton coding principle [50]. Worker ω then applies the corresponding Körner–Marton decoding operation, constructs the prescribed polynomial $\mathbf{p}_i(\omega)$, and computes its assigned output subfunction. The worker transmits the resulting intermediate value to the receiver, which recovers the desired product $\mathbf{A}_i^\intercal \mathbf{B}_i$ for realization i through polynomial interpolation.

4.2 System Model

We consider the distributed computing system illustrated in Figure 4.1, consisting of two source nodes, N worker nodes, and a receiver. Source nodes I and II respectively store the length-n matrix sequences

$$\{\mathbf{A}_i\}_{i\in[n]} \qquad \text{and} \qquad \{\mathbf{B}_i\}_{i\in[n]},$$

where

$$\mathbf{A}_i, \mathbf{B}_i \in \mathbb{F}_q^{m_A \times m}, \qquad q \geq 2,$$

and $i \in [n]$ indexes the i-th realization. For each realization, the source nodes first partition and preprocess their respective matrices. They then separately construct coded descriptions of the resulting submatrices using the Körner–Marton encoding principle [50]. These coded quantities are distributed among the N workers. Each worker constructs and evaluates a specific polynomial function of its assigned Körner–Marton-encoded inputs and transmits the resulting intermediate computation to the receiver. Using the responses returned by a sufficiently large subset of workers, the receiver aims to recover

$$\mathbf{A}_i^\intercal \mathbf{B}_i, \qquad i \in [n],$$

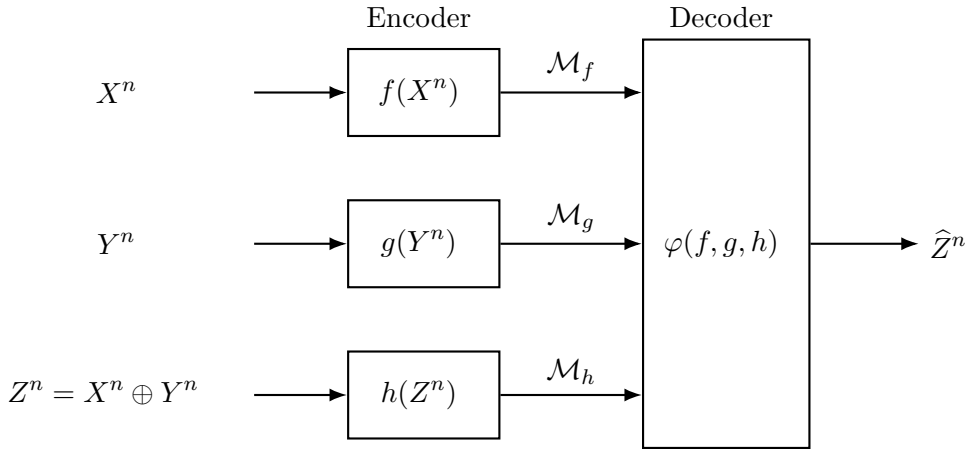

Figure 4.2: Two-help-one source network underlying the Körner–Marton formulation [50]. The three component sources are separately encoded using f, g, and h, and the decoder uses the resulting messages to reconstruct $Z^n = X^n \oplus Y^n$. In the Körner–Marton operating point, $h(Z^n) = h_0$ is a constant mapping, so that M_h carries no source-dependent information.

where $\mathbf{A}_i^\intercal$ denotes the transpose of $\mathbf{A}_i$. The detailed preprocessing, encoding, worker computation, and decoding procedures are presented later in this chapter in Section 4.4.

4.3 Structured Source Coding Preliminaries

Körner–Marton encoding is a structured distributed source coding method originally developed for computing the modulo-two sum of two correlated binary sources [50]. Its main insight is that when a decoder is interested only in a function of separately observed sources, reconstructing the individual sources may be unnecessary and communication-inefficient. Instead, the separate encoders can exploit the algebraic structure of the desired function and directly provide the decoder with a compressed description of that function, which is a summation here. The corresponding two-help-one source network is illustrated in Figure 4.2.

4.3.1 General two-help-one source network

Consider a three-component discrete memoryless source

$$\Big\{(X_i, Y_i, Z_i)\Big\}_{i=1}^{\infty},$$

where the triples (X_i, Y_i, Z_i) are independently and identically distributed according to a joint distribution P_{XYZ}. The random variables X_i, Y_i, and Z_i take values in the finite alphabets $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$, respectively. We denote the first n realizations of the component sources by

$$X^n = (X_1, \ldots, X_n), \qquad Y^n = (Y_1, \ldots, Y_n), \qquad Z^n = (Z_1, \ldots, Z_n).$$

In the two-help-one source network introduced in [50], the three component sources are encoded separately. An (n, ϵ) coding scheme consists of three encoding functions

$$f : \mathcal{X}^n \longrightarrow \mathcal{M}_f, \tag{4.1}$$

$$g : \mathcal{Y}^n \longrightarrow \mathcal{M}_g, \tag{4.2}$$

$$h : \mathcal{Z}^n \longrightarrow \mathcal{M}_h, \tag{4.3}$$

together with a decoding function

$$\varphi : \mathcal{M}_f \times \mathcal{M}_g \times \mathcal{M}_h \longrightarrow \mathcal{Z}^n. \tag{4.4}$$

In the Körner–Marton construction, the third encoder is inactive, namely,

$$h(Z^n) = h_0, \qquad R_Z = 0,$$

while the first two encoders use the same linear transformation,

$$f(X^n) = \mathbf{C}X^n, \qquad g(Y^n) = \mathbf{C}Y^n.$$

Based on the three encoded messages, the decoder forms

$$\widehat{Z}^n \triangleq \varphi\Big(f(X^n), g(Y^n), h(Z^n)\Big). \tag{4.5}$$

The coding scheme is required to satisfy

$$\Pr\Big\{\widehat{Z}^n \neq Z^n\Big\} < \epsilon. \tag{4.6}$$

The corresponding rate triple is defined as

$$\left(\frac{1}{n}\log_2\left|\mathcal{M}_X^{(n)}\right|, \frac{1}{n}\log_2\left|\mathcal{M}_Y^{(n)}\right|, \frac{1}{n}\log_2\left|\mathcal{M}_Z^{(n)}\right|\right). \tag{4.7}$$

A nonnegative rate triple (R_X, R_Y, R_Z) is achievable if, for every $\epsilon > 0$ and every $\delta > 0$, there exists a sufficiently large block length n and an (n, ϵ) coding scheme whose rate triple is componentwise no larger than

$$(R_X + \delta, R_Y + \delta, R_Z + \delta).$$

The term "two-help-one" reflects the fact that the encoded descriptions of X^n and Y^n help the decoder recover Z^n. The general form of the network allows the third source to send its own description through $h(Z^n)$. The classical Körner–Marton construction corresponds to the important special case in which the third encoder sends no information and the decoder computes Z^n using only separately encoded descriptions of X^n and Y^n.

4.3.2 The modulo-two adder

Suppose that X and Y are binary random variables taking values in binary field $\mathbb{F}_2 = \{0, 1\}$, and define

$$Z \triangleq X \oplus Y, \tag{4.8}$$

where $\oplus$ denotes addition over $\mathbb{F}_2$. Equivalently,

$$Z = \begin{cases} 0, & X = Y, \\ 1, & X \neq Y. \end{cases}$$

The resulting source network is referred to as the *modulo-two adder*.

The original Körner–Marton result in [50] considers a symmetric joint distribution of X and Y. For some $0 < p < 1$,

$$\Pr\{X = 0, Y = 0\} = \Pr\{X = 1, Y = 1\} = \frac{1-p}{2}, \tag{4.9}$$

$$\Pr\{X = 0, Y = 1\} = \Pr\{X = 1, Y = 0\} = \frac{p}{2}. \tag{4.10}$$

Hence,

$$\Pr\{Z = 1\} = \Pr\{X \neq Y\} = p,$$

and

$$H(Z) = h_2(p), \tag{4.11}$$

where

$$h_2(p) \triangleq -p \log_2 p - (1-p) \log_2 (1-p) \tag{4.12}$$

denotes the binary entropy function.

For this symmetric modulo-two adder, the achievable rate region is given by [50]

$$\mathcal{R}_{\mathrm{KM}} = \Big\{ (R_X, R_Y, R_Z) : R_X + R_Z \geq H(Z),\ R_Y + R_Z \geq H(Z) \Big\}. \tag{4.13}$$

In particular, when the third source does not transmit any information, i.e.,

$$R_Z = 0,$$

the rate region requires

$$R_X \geq H(Z), \qquad R_Y \geq H(Z). \tag{4.14}$$

The rate pair

$$(R_X, R_Y) = \Big(H(Z), H(Z) \Big) \tag{4.15}$$

is therefore achievable. This is the operating point commonly associated with Körner–Marton encoding.

4.3.3 Common linear encoding construction

The achievability of (4.15) follows from the existence of suitable linear codes for binary symmetric channels [50]. More precisely, for every $\epsilon > 0$ and every $\delta > 0$, and for a sufficiently large block length n, there exists a binary matrix

$$\mathbf{C} \in \mathbb{F}_2^{m \times n} \tag{4.16}$$

and a decoding function

$$\psi : \mathbb{F}_2^m \longrightarrow \mathbb{F}_2^n \tag{4.17}$$

such that

$$m < n\Big(H(Z) + \delta\Big) \tag{4.18}$$

and

$$\Pr\Big\{\psi\Big(\mathbf{C}Z^n\Big) \neq Z^n\Big\} < \epsilon. \tag{4.19}$$

The matrix $\mathbf{C}$ can be interpreted as the parity-check matrix of a linear channel code. The vector $\mathbf{C}Z^n$ is then the syndrome of Z^n, and ψ acts as a syndrome decoder that recovers the most likely source sequence associated with the observed syndrome.

The two separate source encoders use the same linear transformation:

$$f(X^n) = \mathbf{C}X^n, \tag{4.20}$$
$$g(Y^n) = \mathbf{C}Y^n. \tag{4.21}$$

The third encoder is chosen to be a constant mapping,

$$h(Z^n) = h_0, \tag{4.22}$$

and therefore communicates no source-dependent information.

The decoder first adds the two received descriptions over $\mathbb{F}_2$. By the linearity of $\mathbf{C}$,

$$\begin{aligned} f(X^n) \oplus g(Y^n) &= \mathbf{C}X^n \oplus \mathbf{C}Y^n \\ &= \mathbf{C}\Big(X^n \oplus Y^n\Big) \\ &= \mathbf{C}Z^n. \end{aligned} \tag{4.23}$$

It then applies the source decoder and forms

$$\widehat{Z}^n = \psi\Big(f(X^n) \oplus g(Y^n)\Big). \tag{4.24}$$

Combining (4.19) and (4.23) yields

$$\Pr\left\{\widehat{Z}^n \neq Z^n\right\} < \epsilon. \tag{4.25}$$

Since the number of rows of $\mathbf{C}$ satisfies (4.18), the rate of each encoder approaches $H(Z)$. Consequently,

$$\Big(H(Z), H(Z), 0\Big)$$

is an achievable rate triple.

Another immediate achievable point is

$$\Big(0, 0, H(Z)\Big),$$

which corresponds to directly compressing Z^n at the third source. Time sharing between these two operating points produces

$$R_X = (1-\alpha)H(Z), \tag{4.26}$$
$$R_Y = (1-\alpha)H(Z), \tag{4.27}$$
$$R_Z = \alpha H(Z), \qquad 0 \leq \alpha \leq 1. \tag{4.28}$$

Together with rate enlargement, these points generate the complete region in (4.13).

4.3.4 Gains from computing instead of reconstructing

A conventional distributed source coding strategy could first reconstruct X^n and Y^n and subsequently calculate their modulo-two sum. Under the symmetric distribution in (4.9)–(4.10), the minimum sum rate required to reconstruct both sources is

$$\begin{aligned} H(X,Y) &= H(X) + H(Y \mid X) \\ &= 1 + h_2(p). \end{aligned} \tag{4.29}$$

In contrast, when $R_Z = 0$, Körner–Marton encoding directly computes Z^n using the sum rate

$$R_X + R_Y = 2h_2(p). \tag{4.30}$$

For $0 < p < \frac{1}{2}$, $h_2(p) < 1$, and thus

$$2h_2(p) < 1 + h_2(p). \tag{4.31}$$

Therefore, direct function computation requires less communication than separate reconstruction of the two sources. The saving becomes larger as the correlation between X and Y increases, since the difference sequence Z^n then has substantially lower entropy than the individual source sequences.

The gain results from two forms of structure. First, the decoder seeks only the function $X^n \oplus Y^n$, rather than the individual source sequences. Second, the two encoders use the same linear transformation, so their separately generated messages remain algebraically aligned. The sum of the two messages is therefore a valid compressed representation of the desired function.

4.3.5 Algebraic interpretation and finite-field formulation

Although the original Körner–Marton construction concerns binary sources and modulo-two addition [50], Han and Kobayashi extended the structured encoding principle to the computation of sums of q-ary correlated sources over the finite field $\mathbb{F}_q$ [51]. Related refinements of the Körner–Marton approach for more general binary source distributions were also developed by Ahlswede and Han [171]. Here, we use the finite-field formulation underlying the former extension. Let U and V be correlated random variables over a finite field $\mathbb{F}_q$, and suppose that the desired function is

$$W = U + V, \tag{4.32}$$

where the addition is performed over $\mathbb{F}_q$. For the corresponding length-n sequences U^n, V^n, and W^n, the two source nodes can apply a common linear transformation

$$\mathbf{C} \in \mathbb{F}_q^{m_n \times n}$$

and transmit

$$f(U^n) = \mathbf{C}U^n, \tag{4.33}$$
$$g(V^n) = \mathbf{C}V^n. \tag{4.34}$$

The decoder then obtains

$$\begin{aligned} f(U^n) + g(V^n) &= \mathbf{C}U^n + \mathbf{C}V^n \\ &= \mathbf{C}\big(U^n + V^n\big) \\ &= \mathbf{C}W^n. \end{aligned} \tag{4.35}$$

A source decoder matched to the distribution of W can subsequently recover W^n from its compressed linear description.

The essential property is therefore not limited to the binary field: the common encoder preserves the algebraic operation that defines the desired function. For more general functions, the source nodes may first apply local transformations that produce an additive representation. Specifically, suppose that separate mappings F_1 and F_2 satisfy

$$G(A, B) = F_1(A) + F_2(B), \tag{4.36}$$

or that the desired function can be reconstructed from $G(A, B)$. Körner–Marton encoding can then be applied to the transformed source sequences rather than directly to the original observations. This principle allows structured linear source coding to support broader non-linear and bilinear computations [109].

4.3.6 Application to the proposed DSPolyDot codes

In the distributed matrix multiplication framework considered in this chapter, the two source nodes separately observe the matrix sequences

$$\{\mathbf{A}_i\}_{i\in[n]} \qquad \text{and} \qquad \{\mathbf{B}_i\}_{i\in[n]}.$$

For every worker $\omega \in \Omega$, the source nodes construct local sequences

$$F_1^n(\omega) = \Big(F_{1,1}(\omega), \ldots, F_{1,n}(\omega)\Big), \tag{4.37}$$
$$F_2^n(\omega) = \Big(F_{2,1}(\omega), \ldots, F_{2,n}(\omega)\Big). \tag{4.38}$$

The local transformations are designed so that their finite-field sum contains the structured quantities needed by worker ω:

$$G^n(\omega) \triangleq F_1^n(\omega) + F_2^n(\omega). \tag{4.39}$$

The two source nodes then apply the same linear source coding matrix $\mathbf{C}$ and transmit

$$M_1(\omega) = \mathbf{C}F_1^n(\omega), \tag{4.40}$$
$$M_2(\omega) = \mathbf{C}F_2^n(\omega). \tag{4.41}$$

Worker ω combines the two descriptions as

$$\begin{aligned} M_1(\omega) + M_2(\omega) &= \mathbf{C}\left(F_1^n(\omega) + F_2^n(\omega)\right) \\ &= \mathbf{C}G^n(\omega). \end{aligned} \tag{4.42}$$

It then applies the corresponding source decoder to recover $G^n(\omega)$. For every realization $i \in [n]$, the decoded structured quantities are used to construct and evaluate the worker polynomial $\mathbf{p}_i(\omega)$.

Körner–Marton encoding therefore constitutes the source coding stage of the proposed DSPolyDot construction. It preserves the additive structure generated by separate source-side transformations, while the polynomial-coding stage introduces redundancy across the workers. The receiver can consequently recover $\mathbf{A}_i^\intercal \mathbf{B}_i$ from a sufficiently large subset of the returned worker computations through polynomial interpolation.

We emphasize that the structured source-side mechanism used in this step builds on the prior distributed structured matrix computation frameworks in [109], [110]. Its role in the present chapter is to provide the source coding stage that is subsequently combined with the worker-side polynomial redundancy in the DSPolyDot construction.

4.4 DSPolyDot Codes

We now describe the DSPolyDot construction developed in [108]. The source-side structured transformations used in this construction follow the prior frameworks in [109], [110], while the worker-side redundancy is combined with the PolyDot-code principle in [112]. Unless stated otherwise, each step is carried out independently for every realization $i \in [n]$.

Splitting source matrices Source node I splits $\mathbf{A}_i$ into submatrices $\alpha_i(j,k)$ and source node II splits $\mathbf{B}_i$ into submatrices $\beta_i(j,k)$ for $j \in S_r$, $k \in S_c$, assuming that $S_r \triangleq \{0,\dots,s_r-1\}$, $S_c \triangleq \{0,\dots,s_c-1\}$, where s_r divides m_A and s_c divides m. Matrix $\mathbf{A}_i$ is then described as

$$\mathbf{A}_i \triangleq \begin{bmatrix} \alpha_i(0,0) & \dots & \alpha_i(0,s_c-1) \\ \vdots & \ddots & \vdots \\ \alpha_i(s_r-1,0) & \dots & \alpha_i(s_r-1,s_c-1) \end{bmatrix}, \tag{4.43}$$

where $\alpha_i(j,k) \in \mathbb{F}_q^{\frac{m_A}{s_r}\times\frac{m}{s_c}}$ and $q \geq 2$. Similarly, matrix $\mathbf{B}_i$ is described as

$$\mathbf{B}_i \triangleq \begin{bmatrix} \beta_i(0,0) & \dots & \beta_i(0,s_c-1) \\ \vdots & \ddots & \vdots \\ \beta_i(s_r-1,0) & \dots & \beta_i(s_r-1,s_c-1) \end{bmatrix}, \tag{4.44}$$

where $\beta_i(j,k) \in \mathbb{F}_q^{\frac{m_A}{s_r}\times\frac{m}{s_c}}$.

Polynomial construction Given the block representation in (4.43), to determine $\mathbf{A}_i^\intercal \mathbf{B}_i \in \mathbb{F}_q^{m\times m}$, inspired by PolyDot codes in [112], we define the following linear pre-processing functions at the source nodes for different coefficients x_ω across the workers $\omega \in \Omega \triangleq \{1, \cdots, N\}$.

$$\tilde{\mathbf{A}}_i(\omega) \triangleq \sum_{j=0}^{s_c-1}\sum_{k=0}^{s_r-1} \alpha_i(j,k) x_\omega^j x_\omega^{s_c k} \in \mathbb{F}_q^{\frac{m_A}{s_r}\times\frac{m}{s_c}}, \tag{4.45}$$
$$\tilde{\mathbf{B}}_i(\omega) \triangleq \sum_{j=0}^{s_r-1}\sum_{k=0}^{s_c-1} \beta_i(j,k) x_\omega^{s_c(s_r-1-j)} x_\omega^{s_c(2s_r-1)k} \in \mathbb{F}_q^{\frac{m_A}{s_r}\times\frac{m}{s_c}}.$$

To devise non-linear mappings of the constructed polynomials in (4.45) for DSPolyDot codes, each source node uses the following row-block representations to construct the corresponding submatrices of the polynomial evaluations:

$$\tilde{\mathbf{A}}_i(\omega) = \begin{bmatrix}\tilde{\mathbf{A}}_{i1}(\omega)\\ \tilde{\mathbf{A}}_{i2}(\omega)\end{bmatrix}, \quad \tilde{\mathbf{B}}_i(\omega) = \begin{bmatrix}\tilde{\mathbf{B}}_{i1}(\omega)\\ \tilde{\mathbf{B}}_{i2}(\omega)\end{bmatrix}, \quad \omega \in \Omega\,, \tag{4.46}$$

where $\tilde{\mathbf{A}}_{i1}(\omega), \tilde{\mathbf{A}}_{i2}(\omega), \tilde{\mathbf{B}}_{i1}(\omega), \tilde{\mathbf{B}}_{i2}(\omega) \in \mathbb{F}_q^{\frac{m_A}{2s_r}\times\frac{m}{s_c}}$.

Symmetry condition and scope of the DSPolyDot code construction. The DSPolyDot post-processing operation considered in this section does not apply to arbitrary pairs of source matrices. Its correctness depends on a symmetry condition, involving the row blocks of the polynomial evaluations. In particular, for every realization $i \in [n]$ and every worker $\omega \in \Omega$, we assume

$$\tilde{\mathbf{B}}_{i1}^\intercal(\omega)\tilde{\mathbf{A}}_{i1}(\omega) = \tilde{\mathbf{A}}_{i1}^\intercal(\omega)\tilde{\mathbf{B}}_{i1}(\omega). \tag{4.47}$$

We refer to (4.47) as the blockwise cross-product symmetry condition. To see why this condition is required, recall that the worker-side post-processing operation developed below (cf. (4.52)) cancels the two self-product terms and yields

$$\tilde{\mathbf{A}}_{i2}^\top(\omega)\tilde{\mathbf{B}}_{i2}(\omega) \oplus_q \tilde{\mathbf{B}}_{i1}^\top(\omega)\tilde{\mathbf{A}}_{i1}(\omega).$$

On the other hand, the desired polynomial evaluation is

$$\tilde{\mathbf{A}}_i^\top(\omega)\tilde{\mathbf{B}}_i(\omega) = \tilde{\mathbf{A}}_{i1}^\top(\omega)\tilde{\mathbf{B}}_{i1}(\omega) \oplus_q \tilde{\mathbf{A}}_{i2}^\top(\omega)\tilde{\mathbf{B}}_{i2}(\omega).$$

Therefore, these two expressions coincide precisely under

$$\tilde{\mathbf{B}}_{i1}^\top(\omega)\tilde{\mathbf{A}}_{i1}(\omega) = \tilde{\mathbf{A}}_{i1}^\top(\omega)\tilde{\mathbf{B}}_{i1}(\omega),$$

which is exactly the condition in (4.47). Without this condition, an undesired cross-product term remains and the worker output is not an evaluation of the polynomial associated with the desired matrix product. Consequently, polynomial interpolation at the receiver no longer guarantees recovery of $\mathbf{A}_i^\top \mathbf{B}_i$. We emphasize that this symmetry condition is not a general requirement of DMM, but a structural requirement of the particular DSPolyDot post-processing construction considered here. Accordingly, all DSPolyDot decodability, recovery-threshold, communication-cost, and computation-cost results in this chapter are established under this condition.

Non-linear pre-processing To devise novel polynomial codes that capture the computation structure for evaluating $\mathbf{A}_i^\intercal \mathbf{B}_i$, and following [109], source nodes I and II construct non-linear mappings $f_{1i}(\omega), f_{2i}(\omega) \in \mathbb{F}_q^{(\frac{m_A}{s_r}+\frac{m}{s_c})\times\frac{m}{s_c}}$ of $\tilde{\mathbf{A}}_i(\omega)$ and $\tilde{\mathbf{B}}_i(\omega)$, for each $i \in [n]$, $\omega \in \Omega$, respectively as

$$f_{1i}(\omega) \triangleq \begin{bmatrix} \tilde{\mathbf{A}}_{i2}(\omega) \\ \tilde{\mathbf{A}}_{i1}(\omega) \\ \tilde{\mathbf{A}}_{i2}^\intercal(\omega)\tilde{\mathbf{A}}_{i1}(\omega) \end{bmatrix}, \; f_{2i}(\omega) \triangleq \begin{bmatrix} \tilde{\mathbf{B}}_{i1}(\omega) \\ \tilde{\mathbf{B}}_{i2}(\omega) \\ \tilde{\mathbf{B}}_{i1}^\intercal(\omega)\tilde{\mathbf{B}}_{i2}(\omega) \end{bmatrix}. \tag{4.48}$$

Each source node then reshapes the non-linear mappings $f_{1i}(\omega), f_{2i}(\omega)$, as the vector representations of the form

$$F_{1i}(\omega) \triangleq \left[f_{1i}(:,1) \; f_{1i}(:,2) \; \ldots \; f_{1i}(:,\tfrac{m}{s_c}) \right]^\intercal,$$
$$F_{2i}(\omega) \triangleq \left[f_{2i}(:,1) \; f_{2i}(:,2) \; \ldots \; f_{2i}(:,\tfrac{m}{s_c}) \right]^\intercal,$$

where $F_{1i}(\omega), F_{2i}(\omega) \in \mathbb{F}_q^{(\frac{m_A}{s_r}+\frac{m}{s_c})\frac{m}{s_c}\times 1}$.

The source nodes concatenate the above-mentioned vectors to build new vectors

$$F_1^n(\omega) \triangleq [F_{11}, F_{12}, \ldots, F_{1n}]^\intercal \in \mathbb{F}_q^{n(\frac{m_A}{s_r}+\frac{m}{s_c})\frac{m}{s_c}\times 1},$$
$$F_2^n(\omega) \triangleq [F_{21}, F_{22}, \ldots, F_{2n}]^\intercal \in \mathbb{F}_q^{n(\frac{m_A}{s_r}+\frac{m}{s_c})\frac{m}{s_c}\times 1}. \tag{4.49}$$

Körner-Marton encoding Given two length-n source sequences X^n and Y^n, the Körner-Marton approach addresses the rate region for computing the sum of binary sources in the i.i.d. setting, provided that the joint distribution of sources is symmetric [50]. In this framework, both encoders apply the same linear mapping $\mathbf{C} \in \mathbb{F}_2^{k\times n}$ to their respective source sequences, and the decoder reconstructs the sum $Z^n = X^n \oplus Y^n$ by exploiting the linearity of the encoding.

As shown in [50, Theorem 1], there exist such linear encoders each with rate $\frac{k}{n}$ arbitrarily close to $H(Z)$ for which Z^n can be recovered with small probability of error as $n \to \infty$, where $H(Z)$ denotes the entropy of the binary random variable $Z = X \oplus Y$.

Extensions of the Körner-Marton framework to finite fields $\mathbb{F}_q$ with $q > 2$ can be found in, e.g., [109], [172].

Each source node then encodes vectors $F_1^n(\omega), F_2^n(\omega)$, given in (4.49), with a common linear encoder $\mathbf{C} \in \mathbb{F}_q^{k_n \times n'}$, where dimensions k_n and n' are expressed as

$$k_n \triangleq \max_{\omega \in \Omega} H_q\Big(F_1^n(\omega) \oplus_q F_2^n(\omega)\Big),$$
$$n' \triangleq n\Big(\frac{m_A}{s_r} + \frac{m}{s_c}\Big)\frac{m}{s_c}.$$

Each source node then transmits the Körner-Marton encoded data, demonstrated as $\mathbf{C}F_1^n(\omega), \mathbf{C}F_2^n(\omega) \in \mathbb{F}_q^{k_n \times 1}$ to each worker $\omega \in \Omega$ at an operating rate of $\frac{k_n}{n'}$.

Körner–Marton decoding Worker ω first recovers the modulo-q sum $F_1^n(\omega) \oplus_q F_2^n(\omega)$ by applying Körner–Marton decoding to the received sequence $\mathbf{C}F_1^n(\omega) \oplus_q \mathbf{C}F_2^n(\omega)$. The validity of this decoding follows from a generalization of Elias' lemma [173], which guarantees reliable recovery of $F_1^n(\omega) \oplus_q F_2^n(\omega)$ whenever $\frac{k_n}{n'} > H_q(F_1^n(\omega) \oplus_q F_2^n(\omega))$. Consequently, the decoder can reconstruct $F_{1i}(\omega) \oplus_q F_{2i}(\omega)$ in an asymptotically lossless manner. Next, from the recovered sum $F_{1i}(\omega) \oplus_q F_{2i}(\omega)$, worker ω computes the corresponding polynomial outputs $\mathbf{p}_i(\omega) \triangleq \{p_i^{(1)}(\omega), p_i^{(2)}(\omega), p_i^{(3)}(\omega)\}$, given by

$$\begin{aligned} p_i^{(1)}(\omega) &\triangleq \tilde{\mathbf{A}}_{i2}(\omega) \oplus_q \tilde{\mathbf{B}}_{i1}(\omega) \in \mathbb{F}_q^{\frac{m_A}{2s_r} \times \frac{m}{s_c}}, \\ p_i^{(2)}(\omega) &\triangleq \tilde{\mathbf{A}}_{i1}(\omega) \oplus_q \tilde{\mathbf{B}}_{i2}(\omega) \in \mathbb{F}_q^{\frac{m_A}{2s_r} \times \frac{m}{s_c}}, \\ p_i^{(3)}(\omega) &\triangleq \tilde{\mathbf{A}}_{i2}^{\intercal}(\omega)\tilde{\mathbf{A}}_{i1}(\omega) \oplus_q \tilde{\mathbf{B}}_{i1}^{\intercal}(\omega)\tilde{\mathbf{B}}_{i2}(\omega) \in \mathbb{F}_q^{\frac{m}{s_c} \times \frac{m}{s_c}}. \end{aligned} \tag{4.50}$$

Here, $p_i^{(1)}(\omega)$ and $p_i^{(2)}(\omega)$ are linear polynomials in the submatrices defined in (4.46), while $p_i^{(3)}(\omega)$ is a parity polynomial obtained through non-linear processing as defined in (4.48). Although $p_i^{(3)}(\omega)$ involves non-linear terms, it remains linearly separable in $\tilde{\mathbf{A}}_i(\omega)$ and $\tilde{\mathbf{B}}_i(\omega)$, which enables the asymptotically lossless construction of $\mathbf{p}_i(\omega)$ using the Körner–Marton encoding.

Post-processing After evaluating $\mathbf{p}_i(\omega)$, each worker $\omega \in \Omega$ locally combines the recovered polynomial components. The correctness of this step relies

on the blockwise cross-product symmetry condition in (4.47). Accordingly, the DSPolyDot construction and the resulting recovery guarantees apply only when (4.47) holds for every $i \in [n]$ and $\omega \in \Omega$.

Using (4.45), (4.46), and (4.50), worker ω computes

$$p_i(\omega) \triangleq \left(p_i^{(1)}(\omega)\right)^{\intercal} p_i^{(2)}(\omega) \ominus_q p_i^{(3)}(\omega). \tag{4.51}$$

Substituting the definitions of the three polynomial components in (4.50) into (4.51) yields

$$\begin{aligned} p_i(\omega) &= \left(\tilde{\mathbf{A}}_{i2}(\omega) \oplus_q \tilde{\mathbf{B}}_{i1}(\omega)\right)^{\intercal} \left(\tilde{\mathbf{A}}_{i1}(\omega) \oplus_q \tilde{\mathbf{B}}_{i2}(\omega)\right) \\ &\quad \ominus_q \left(\tilde{\mathbf{A}}_{i2}^{\intercal}(\omega)\tilde{\mathbf{A}}_{i1}(\omega) \oplus_q \tilde{\mathbf{B}}_{i1}^{\intercal}(\omega)\tilde{\mathbf{B}}_{i2}(\omega)\right) \\ &= \tilde{\mathbf{A}}_{i2}^{\intercal}(\omega)\tilde{\mathbf{B}}_{i2}(\omega) \oplus_q \tilde{\mathbf{B}}_{i1}^{\intercal}(\omega)\tilde{\mathbf{A}}_{i1}(\omega). \end{aligned} \tag{4.52}$$

The blockwise cross-product symmetry condition (4.47) converts the reversed cross-product $\tilde{\mathbf{B}}_{i1}^{\top}(\omega)\tilde{\mathbf{A}}_{i1}(\omega)$ remaining in (4.52) into the desired term $\tilde{\mathbf{A}}_{i1}^{\top}(\omega)\tilde{\mathbf{B}}_{i1}(\omega)$, and as a result, the whole expression in (4.52) becomes

$$\begin{aligned} p_i(\omega) &= \tilde{\mathbf{A}}_{i1}^{\intercal}(\omega)\tilde{\mathbf{B}}_{i1}(\omega) \oplus_q \tilde{\mathbf{A}}_{i2}^{\intercal}(\omega)\tilde{\mathbf{B}}_{i2}(\omega) \\ &= \tilde{\mathbf{A}}_{i}^{\intercal}(\omega)\tilde{\mathbf{B}}_{i}(\omega). \end{aligned} \tag{4.53}$$

Consequently, each worker obtains an evaluation of the polynomial associated with the desired matrix product and transmits

$$\begin{aligned} p_i(\omega) &= \tilde{\mathbf{A}}_{i}^{\intercal}(\omega)\tilde{\mathbf{B}}_{i}(\omega) \\ &= \sum_{(j,k,j',k')\in\mathcal{S}} \alpha_i^{\intercal}(j,k)\beta_i(j',k')x_\omega^{j+s_c(s_r-1-j'+k)+s_c(2s_r-1)k'} \end{aligned} \tag{4.54}$$

to the receiver, where

$$\mathcal{S} \triangleq S_c \times S_r \times S_r \times S_c$$

denotes the Cartesian product of the corresponding index sets.

We next discuss an important parameter in the context of DMM, known as the *recovery threshold*, which is the minimum number of workers needed to successfully evaluate $\mathbf{A}^{\intercal}\mathbf{B}$ at the receiver, and ensures the successful recovery of the matrix product in DMM with any subset of the workers.

Definition 22 (Recovery threshold)**.** *For a fixed distributed computing scheme with worker set Ω, the recovery threshold is the smallest integer N_r such that, for every subset $\mathcal{I} \subseteq \Omega$ satisfying $|\mathcal{I}| = N_r$, the receiver can recover the desired matrix product from the outputs of the workers in $\mathcal{I}$.*

For the proposed construction, this definition applies to the polynomial-decoding stage conditioned on successful Körner–Marton decoding at the responding workers. The corresponding end-to-end recovery guarantee is asymptotically lossless as the source coding blocklength tends to infinity.

Receiver-side decoding The receiver collects evaluations of $\tilde{\mathbf{A}}_i^\intercal(\omega)\tilde{\mathbf{B}}_i(\omega)$ from a subset of cooperative workers $\omega \in \Omega$. Once a sufficient number of such evaluations (equal to the recovery threshold N_r) is obtained, the receiver applies polynomial interpolation, following the approach in [112], to reconstruct the desired matrix product $\mathbf{A}_i^\intercal\mathbf{B}_i$ (cf. Proposition 3).

Next, we evaluate the computation and communication complexities of the aforementioned system model.

4.5 Main Results

We next provide mathematical analysis of the memory requirements and the recovery threshold as well as a cost analysis for computation and communication of the proposed DSPolyDot framework shown in Figure 4.1, respectively.

4.5.1 Computation Cost of DSPolyDot Codes

In this part, we detail the complexity of computation at the source nodes as well as the worker nodes, respectively. Note that we ignore the bounded complexity terms imposed by reshape or concatenation operations.

Proposition 2. (Computation cost per source node.) *The total computation cost of each source node, including the Körner-Marton linear encoding matrix, equals*

$$N\left(m_A m + \frac{m^2 m_A}{2 s_r s_c^2}\right). \tag{4.55}$$

Proof. For each $\omega \in \Omega$, each source node first computes $\tilde{\mathbf{A}}_i(\omega)$ or $\tilde{\mathbf{B}}_i(\omega)$ (cf. (4.45)) with a complexity of $s_c \times s_r \times \frac{m_A}{s_r} \times \frac{m}{s_c}$. Each source node then computes one of the non-linear mappings f_{1i} and f_{2i}, each with a complexity of $\frac{m}{s_c} \times \frac{m_A}{2s_r} \times \frac{m}{s_c}$. Considering N workers and summing the aforementioned costs completes the proof. □

The encoding process in (4.50), incorporating the dimensions of polynomials, dictates to each worker a storage limit

$$M_{\text{DSPolyDot}} = \frac{m_A m}{s_r s_c} + \frac{m^2}{s_c^2}. \tag{4.56}$$

The proposed DSPolyDot codes require storage no greater than that of PolyDot codes in [112], given as

$$M_{\text{PolyDot}} = \frac{2 m_A m}{s_r s_c},$$

when $\frac{m}{s_c} \leq \frac{m_A}{s_r}$.

We next characterize the *recovery threshold* for DSPolyDot codes.

Proposition 3 (Achievable recovery threshold of DSPolyDot codes)**.** *Assume that the blockwise cross-product symmetry condition in* (4.47) *holds for every* $i \in [n]$ *and* $\omega \in \Omega$*. Let* N_r *denote the recovery threshold of the proposed DSPolyDot construction according to Definition 22. Then,*

$$N_r \leq \overline{N}_r \triangleq s_c^2(2s_r - 1). \tag{4.57}$$

Equivalently, $\overline{N}_r = s_c^2(2s_r - 1)$ *is an achievable recovery threshold for the proposed construction.*

In the asymptotic source coding regime, in which the Körner–Marton decoding error probability tends to zero as $n \to \infty$*, the receiver can therefore recover the desired matrix product from the outputs of any* $\overline{N}_r$ *responding workers. Consequently, when* $N \geq \overline{N}_r$*, the construction guarantees tolerance of up to* $N - \overline{N}_r$ *straggling workers.*

Proof. Under the blockwise cross-product symmetry condition in (4.47), the worker-side post-processing operation produces

$$p_i(\omega) = \tilde{\mathbf{A}}_i^{\intercal}(\omega)\tilde{\mathbf{B}}_i(\omega).$$

Thus, the value transmitted by worker ω is the evaluation, at $x = x_\omega$, of the polynomial

$$\begin{aligned} p_i(x) &\triangleq \tilde{\mathbf{A}}_i^{\intercal}(x)\tilde{\mathbf{B}}_i(x) \\ &= \sum_{(j,k,j',k') \in \mathcal{S}} \boldsymbol{\alpha}_i^{\intercal}(j,k)\boldsymbol{\beta}_i(j',k')x^{e(j,k,j',k')}, \end{aligned} \tag{4.58}$$

where

$$\mathcal{S} = \mathcal{S}_c \times \mathcal{S}_r \times \mathcal{S}_r \times \mathcal{S}_c,$$

with

$$\mathcal{S}_c = \{0, 1, \ldots, s_c - 1\}, \qquad \mathcal{S}_r = \{0, 1, \ldots, s_r - 1\},$$

and

$$e(j,k,j',k') \triangleq j + s_c(s_r - 1 - j' + k) + s_c(2s_r - 1)k'. \tag{4.59}$$

The exponent in (4.59) is increasing in j, k, and k', and decreasing in j'. Its largest possible value over $\mathcal{S}$ is therefore attained at

$$j = s_c - 1, \qquad k = s_r - 1, \qquad j' = 0, \qquad k' = s_c - 1.$$

Consequently,

$$\begin{aligned}\deg\Big(p_i(x)\Big) &\le (s_c-1)+s_c\Big((s_r-1)+(s_r-1)\Big)\\ &\quad + s_c(2s_r-1)(s_c-1)\\ &= s_c^2(2s_r-1)-1. \end{aligned} \tag{4.60}$$

Each responding worker provides one evaluation

$$p_i(\omega) = p_i(x_\omega).$$

Because the evaluation points $\{x_\omega : \omega \in \Omega\}$ are distinct, any

$$\overline{N}_r = s_c^2(2s_r-1)$$

such evaluations uniquely determine every polynomial of degree at most

$$s_c^2(2s_r-1)-1.$$

Hence, the receiver can reconstruct $p_i(x)$ from the outputs of every subset $\mathcal{I} \subseteq \Omega$ satisfying

$$|\mathcal{I}| = \overline{N}_r.$$

It can then extract the coefficients corresponding to the desired block products and reconstruct $\mathbf{A}_i^\intercal \mathbf{B}_i$.

Thus, every subset of $\overline{N}_r$ responding workers is sufficient for recovery. By Definition 22,

$$N_r \le \overline{N}_r = s_c^2(2s_r-1),$$

which proves (4.57). □

The achievable recovery threshold in (4.57) coincides with the interpolation-based recovery guarantee of the original PolyDot construction in [112] when its partitioning parameters are identified as $s = s_r$ and $t = s_c$. This provides a common recovery guarantee for comparing their communication and computation costs.

We next evaluate the computation cost of DSPolyDot codes.

Proposition 4. (Computation cost of a worker node.) *The computation cost of the worker $\omega \in \Omega$ is equal to*

$$\frac{m_A m}{s_r s_c} + \frac{m_A m^2}{2 s_r s_c^2} + \frac{2m^2}{s_c^2}\,. \tag{4.61}$$

Proof. Each worker $\omega \in \Omega$ evaluates the summation of f_{1i} and f_{2i} with a cost of $\frac{m_A m}{s_r s_c} + \frac{m^2}{s_c^2}$ to build the polynomials $p_i^{(1)}(\omega)$, $p_i^{(2)}(\omega)$, and $p_i^{(3)}(\omega)$. Each worker then computes $p_i(\omega)$ (cf. (4.51), (4.54)), exploiting the dimensions of each computational term, with a cost of $\frac{m}{s_c} \times \frac{m_A}{2s_r} \times \frac{m}{s_c}$ for multiplying $\left(p_i^{(1)}(\omega)\right)^\intercal$ and $p_i^{(2)}(\omega)$), followed by the addition of the terms $\left(p_i^{(1)}(\omega)\right)^\intercal p_i^{(2)}(\omega) \in \mathbb{F}_q^{\frac{m}{s_c} \times \frac{m}{s_c}}$ and $p_i^{(3)}(\omega) \in \mathbb{F}_q^{\frac{m}{s_c} \times \frac{m}{s_c}}$ with a complexity of $\frac{m^2}{s_c^2}$. Summing all the costs, the total computation cost per worker equals (4.61). □

4.5.2 Communication Cost of DSPolyDot Codes

We here determine the communication complexity from the source node to the worker nodes as well as from the workers to the receiver, respectively, as follows.

Proposition 5. (Communication cost per source node.) *The total communication cost of each source node for transmitting* $\mathbf{C}F_{1i}(\omega), \mathbf{C}F_{2i}(\omega)$ *to all workers* $\omega \in \Omega$ *is given by*

$$H_q\Big(F_{1i}(\omega) \oplus_q F_{2i}(\omega)\Big) = N\Big(\frac{m_A m}{s_r s_c} + \frac{m^2}{s_c^2}\Big)\,. \tag{4.62}$$

Proof. Exploiting the dimensions of each polynomial in (4.50), the communication cost of each source node for transmitting $\mathbf{C}F_{1i}(\omega), \mathbf{C}F_{2i}(\omega)$ to N workers equals (4.62). □

Proposition 6. (Communication cost per worker node.) *The communication cost of worker* $\omega \in \Omega$ *is*

$$H_q(p_i(\omega)) = H_q(\tilde{\mathbf{A}}_i^\intercal(\omega)\tilde{\mathbf{B}}_i(\omega)) = \frac{m^2}{s_c^2}\,. \tag{4.63}$$

Proof. Employing (4.54), worker $\omega \in \Omega$ transmits $p_i(\omega) \in \mathbb{F}_q^{\frac{m}{s_c} \times \frac{m}{s_c}}$ to the receiver with a cost equivalent to (4.63). □

Given N_r responding (active) workers, the total communication cost of workers is given as

$$\frac{m^2}{s_c^2} \times N_r = m^2(2s_r - 1)\,. \tag{4.64}$$

We next present the numerical results for the proposed DMM framework with DSPolyDot codes and compare them to the state-of-the-art.

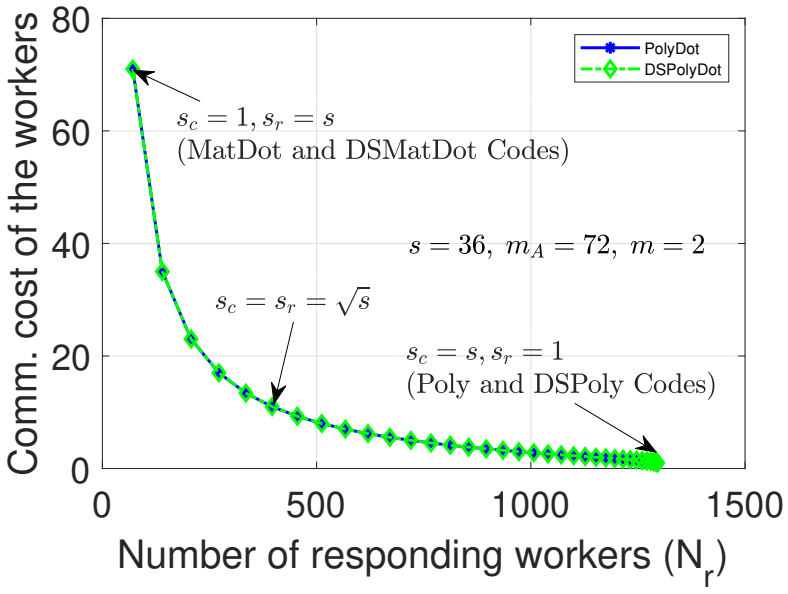

Figure 4.3: Communication cost of workers for computing an element of $\mathbf{A}^\intercal\mathbf{B}$ (normalized: # symbols/m^2).

4.6 Numerical Evaluations of DSPolyDot Codes

We now present numerical evaluations of the proposed DSPolyDot codes and compare their performance with the polynomial code and PolyDot-code constructions in [95], [112]. The numerical evaluations in this section are obtained directly from the analytical communication- and computation-cost expressions and do not depend on particular source-matrix realizations. All DSPolyDot curves are interpreted under the blockwise cross-product symmetry condition in (4.47).

In Figure 4.3, utilizing (4.64), we numerically evaluate the total communication cost of workers for computing an element of $\mathbf{A}^\intercal\mathbf{B} \in \mathbb{F}_q^{m\times m}$, i.e., the total number of transmitted symbols normalized by m^2. The curves for PolyDot and DSPolyDot codes overlap because in both approaches worker $\omega \in \Omega$ computes and then transmits $\tilde{\mathbf{A}}_i^\intercal\tilde{\mathbf{B}}_i$.

Furthermore, for the total communication cost (from source nodes to workers and workers to the receiver), as shown in Figure 4.4, the proposed approach incurs a modest increase in cost when only a small number of workers are active, given the same set of parameters. This modest increase in communication cost results from the additional structured source coding and worker-side post-processing operations. However, as the number of responding

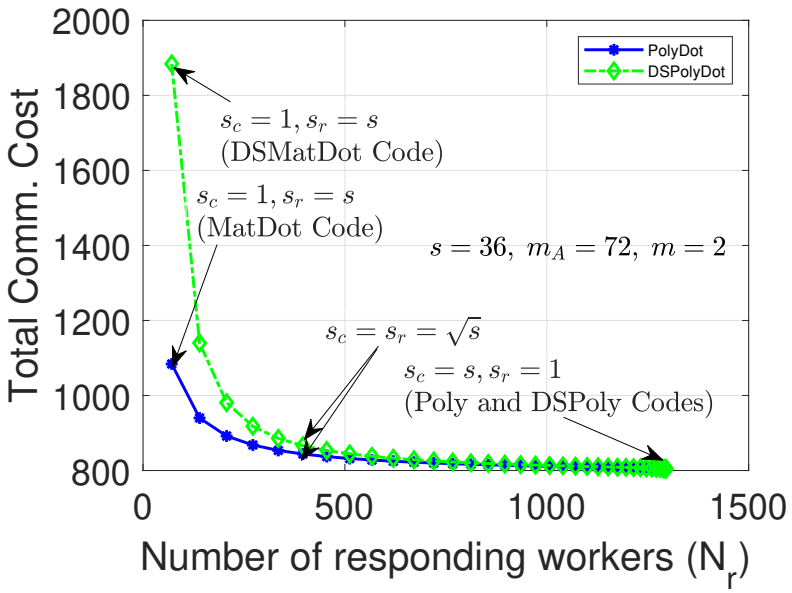

Figure 4.4: Total communication cost (from sources to the workers and from workers to the receiver).

workers increases, the performance converges to that of PolyDot codes [112].

In Figure 4.5, exploiting (4.61), we illustrate the computation cost per worker for computing $\mathbf{A}^\intercal\mathbf{B}$, i.e., the total number of operations normalized by N, which demonstrates savings for a limited number of responding workers, however, asymptotically converging to that of PolyDot codes [112].

Finally, Figure 4.6 illustrates the total computation cost of the proposed DMM framework, including contributions from the source nodes, workers, and the receiver. When only a small number of workers participate, the proposed approach incurs a bounded increase in total computation cost. As the number of responding workers increases, the computation cost decreases and asymptotically converges to that of PolyDot codes [112].

4.7 Conclusion

We studied the DSPolyDot code construction developed in [108] for straggler-resilient DMM. By combining structured source coding with the algebraic structure of matrix multiplication through a non-linear parity polynomial, the proposed construction extends conventional polynomial coding approaches to a distributed setting with two separate source nodes and multiple workers. Under the blockwise cross-product symmetry condition specified in Section 4.4,

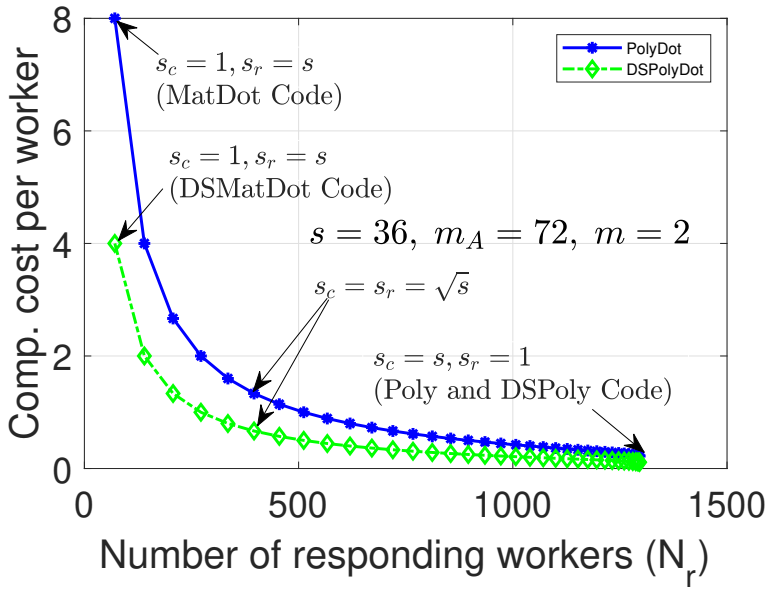

Figure 4.5: Computation cost per worker for computing $\mathbf{A}^\intercal\mathbf{B}$ (normalized: # operations/N).

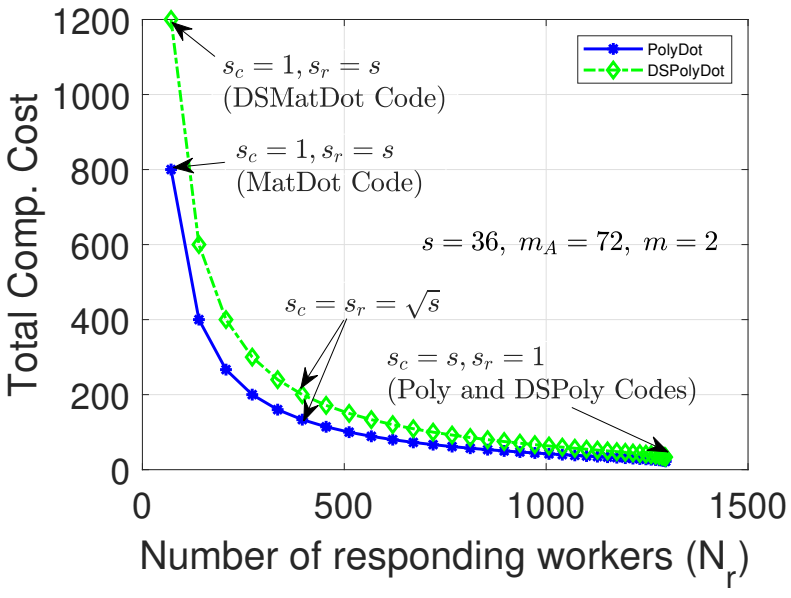

Figure 4.6: Total computation cost (sources-workers-receiver).

the resulting scheme enables recovery of the desired matrix product from a sufficiently large subset of worker responses and therefore provides resilience

against stragglers. Compared to the polynomial code and PolyDot code constructions in [95], [112], the proposed DSPolyDot codes preserve the same asymptotic communication and computation orders while offering a structurally different encoding mechanism for DMM. These properties make DSPolyDot codes a promising alternative for delay-sensitive and straggler-resilient distributed computing systems.

An important direction for future work is to alleviate the blockwise cross-product symmetry condition and extend the construction to arbitrary pairs of source matrices with arbitrary source data distributions.

Chapter 5

Conclusion and Future Research Directions

In this thesis, we investigated three fundamental problems in distributed computing under communication and computation constraints, with particular emphasis on exploiting the structure of the functions being computed. Specifically, we studied: (i) the role of data placement in the distributed computation of Boolean functions, where a sensitivity-based analysis reveals how the influence of individual datasets shapes the communication and computation requirements; (ii) the computation of non-linearly separable multivariate functions in multi-user systems, where a structured tensor formulation enables efficient task allocation and reduces the required number of servers; and (iii) straggler-resilient DMM, where prior structured source coding techniques for distributed matrix computation are combined with polynomial coding in the DSPolyDot construction to enable recovery from a subset of worker responses.

Across these problems, we adopted a unified perspective centered on the interaction among function structure, data placement, computation assignment, and communication cost. The following sections summarize the principal contributions of each technical chapter and discuss the corresponding directions for future research.

5.1 Summary of Contributions and Future Outlooks for Chapter 2

In Chapter 2, we investigated the role of distributed input dataset placement across multiple servers in the distributed computation of Boolean functions through a sensitivity-based framework. By leveraging notions of influence and sensitivity of Boolean functions, we established a direct link between the

structural properties of Boolean functions and the resulting dataset placement configuration and the incurred communication cost.

We showed that the placement of datasets across distributed servers is not merely an implementation detail, but a fundamental design parameter that can significantly impact system performance. Our analysis provided new bounds and insights into how sensitive datasets should be allocated to the servers in order to minimize communication overhead while respecting computation constraints. This approach offered a principled method to characterize and optimize distributed Boolean computation beyond conventional formulations.

A natural extension of this chapter is to generalize the sensitivity-based framework to broader classes of functions, including non-Boolean and general non-linear functions, as well as probabilistic models. Another promising direction lies in bridging sensitivity measures with information-theoretic quantities, potentially leading to bounds on communication complexity. Additionally, exploring adaptive or learning-based placement strategies in dynamic systems could reveal the fundamental performance limits of practical system designs.

5.2 Summary of Contributions and Future Outlooks for Chapter 3

Chapter 3 addressed the distributed computation of non-linearly separable functions through a sparse tensor factorization framework. We introduced a tensor-based representation of user demands that captures the inherent multi-dimensional structure of polynomial computations requested by multiple users. This formulation stems from the observation that non-linearly separable functions can be expressed as multivariate polynomials over a set of basis subfunctions, where interactions across these subfunctions prohibit a simple linear decomposition. To model the resulting computation and communication constraints, we characterized the system through three key design parameters: (i) the computation budget Γ, which limits the number of simultaneously active dimensions (i.e., interacting subfunctions) that each server can process; (ii) the multiplication budget Λ, which governs the local complexity associated with forming higher-order terms within each dimension; and (iii) the communication constraint Δ, which restricts the number of users that can be simultaneously served per transmission. Together, these parameters provide a structured and flexible framework to capture the complexity of polynomial-based interactions, and they play a central role in guiding the design of efficient distributed computation schemes.

Building on the tensor-based representation, we proposed an achievable

scheme based on fixed-support tensor decompositions that helps us model our distributed computing problem as a sparse tensor factorization problem, combined with multi-dimensional tiling that allows us to construct achievable solutions for the sparse tensor factorization problem. This construction exploits sparsity and structured overlaps across dimensions, leading to substantial reductions in the required number of servers compared to matrix-based approaches. Furthermore, we incorporated a combinatorial reduction step based on bipartite matching and flow formulations, which eliminates redundant assignments and improves efficiency. Numerical evaluations confirmed that the proposed schemes achieve significant gains by requiring significantly fewer computational resources.

Future work could focus on developing converse bounds on the number of required servers to measure the gap between achievable schemes and converse bounds, particularly through refined counting arguments or geometric interpretations of tensor tilings. Extending the framework to heterogeneous systems, where computation and communication capabilities vary across servers, is another natural direction. Moreover, scalable algorithmic implementations of the combinatorial reduction step, especially for high-dimensional settings, remain an open challenge. Connections to learning-based tensor decompositions may also provide new insights.

5.3 Summary of Contributions and Future Outlooks for Chapter 4

In Chapter 4, we studied straggler-resilient DMM through the DSPolyDot construction developed in [108]. The construction builds on prior structured source coding techniques for distributed matrix computation, particularly Körner–Marton-style coding and its extensions [50], [109], [110], and combines them with PolyDot-style polynomial redundancy commonly used in CDC [112]. Accordingly, the contribution of the chapter lies in the particular DSPolyDot construction and its straggler-resilient analysis, rather than in the underlying structured source coding principle.

The construction addresses the latency caused by slow or unavailable workers by introducing redundancy across the distributed computations. Through the resulting source encoding, worker-side polynomial computation, and receiver-side interpolation procedures, the desired matrix product can be recovered from a sufficiently large subset of worker responses without waiting for all assigned tasks to be completed. We characterized the corresponding achievable recovery threshold and analyzed the storage, communication, and

computation requirements of the construction. The analysis also makes explicit the blockwise cross-product symmetry condition required for the worker-side post-processing operation.

Several directions remain open for future work. One important objective is to derive tighter bounds on the recovery threshold and to determine the regimes in which the proposed structured construction provides gains over conventional polynomial coding schemes. It would also be worthwhile to study the effects of heterogeneous worker capabilities, including different storage capacities, computation speeds, and communication delays.

Another promising direction is to extend the proposed framework to other bilinear and multilinear distributed computations, such as tensor contractions, matrix chains, and large-scale signal processing operations. Finally, adapting the scheme to dynamic distributed systems with worker arrivals, departures, intermittent connectivity, and time-varying task assignments could further improve its practical applicability.

5.4 Overall Applicability, Limitations, and Outlook

The three technical chapters of this thesis examine distributed computation from different but complementary perspectives. Chapter 2 studies how the sensitivity of a demanded Boolean function can guide dataset placement. Chapter 3 develops a tensor-based representation for assigning non-linearly separable polynomial computations under communication and computation constraints. Chapter 4 combines structured source coding with polynomial coding to provide straggler resilience in distributed matrix multiplication. Although these chapters address different system models, they share a common principle: the structure of the demanded computation should be incorporated directly into the design of data placement, task assignment, encoding, and communication.

Applicability of the proposed frameworks. The sensitivity-based approach of Chapter 2 is most applicable when the relative importance of the input datasets can be quantified through the influence that they exert on the demanded function. Rather than treating all datasets identically, the framework identifies placements that group strongly interacting variables and thereby reduce unnecessary transmissions. The resulting viewpoint may be useful in distributed systems where the requested function is known before placement or task assignment and where the communication cost depends

strongly on the particular datasets stored at each server.

The tensor framework of Chapter 3 applies to multi-user settings in which the requested functions contain non-linear interactions among a finite set of basis subfunctions. In particular, it provides a natural representation for multivariate polynomial demands with prescribed degree bounds. By assigning different exponent dimensions to different tensor modes, the framework avoids treating every monomial as an unrelated basis function. This representation is especially relevant when direct linearization would create a prohibitively large effective function space. The proposed tiling and assignment constructions then translate the algebraic structure of the demands into explicit computation and communication assignments.

The structured coding framework of Chapter 4 applies to DMM with separately observed source matrices and potentially straggling workers. It illustrates how source-side algebraic structure and worker-side polynomial redundancy can be combined within the same distributed computation scheme. Conditional on successful structured source decoding at the responding workers, polynomial interpolation enables the receiver to recover the desired matrix product from a sufficiently large subset of worker responses. The framework is therefore relevant to systems in which waiting for every worker would create an excessive completion delay.

Common modeling assumptions. The results in this thesis rely on several assumptions that make it possible to obtain explicit analytical guarantees. First, the demanded functions and their coefficients are assumed to be known before the placement, encoding, or task-assignment stages. The proposed schemes are therefore designed for a fixed computation task rather than for demands that arrive unexpectedly after the placement has been determined. Second, task assignments and server–user connectivity patterns are generally treated as static during each problem instance. The models do not explicitly account for time-varying link qualities, changing server availability, or online migration of computational tasks.

The constructions also rely on centralized coordination. In Chapters 2 and 3, a master or coordinator determines the placement and computation assignments. In Chapter 4, the source nodes and the receiver use a predetermined coding construction and a prescribed set of evaluation points. Such coordination simplifies the analysis and ensures that the local operations are globally compatible. In a large-scale or rapidly changing network, however, maintaining a complete and current view of the system may itself require non-negligible communication and computation resources.

Several of the explicit expressions are additionally derived under homo-

geneous resource assumptions. For example, the principal constructions in Chapter 3 assume common computation, communication, and exponent-range constraints across the servers, together with divisibility conditions that permit regular tensor partitions. These assumptions yield transparent closed forms and structured tilings, but they do not capture systems in which servers have substantially different computational speeds, memory capacities, communication fan-outs, or supported exponent ranges. Although the underlying tensor formulation can represent more general constraints, the corresponding assignment and factorization problems become more difficult.

Recovery criteria and robustness. The recovery guarantees also differ across the three chapters. Chapters 2 and 3 require exact recovery of the requested functions under their stated algebraic models. Chapter 4 combines two types of guarantees. Polynomial interpolation is exact once sufficiently many correct worker evaluations are available, whereas the Körner–Marton source-coding component provides an asymptotically lossless guarantee whose decoding-error probability tends to zero as the source-coding blocklength increases. Consequently, the overall Chapter 4 construction is asymptotically lossless rather than strictly zero-error at every finite blocklength.

The models primarily address missing or delayed worker responses rather than arbitrary computational errors. In particular, Chapter 4 treats stragglers through redundant polynomial evaluations but does not develop protection against malicious workers that return incorrect results. Similarly, noisy communication links, finite-precision effects, and quantization errors are not included in the main analytical models. Extending the proposed frameworks to approximate, noisy, or adversarial computation would require different recovery criteria and additional forms of redundancy.

Limits of the current optimality guarantees. The results in Chapter 3 are achievability results. The proposed tensor factorizations and the capacitated tile-assignment algorithm construct valid schemes and provide upper bounds on the minimum number of required servers, or equivalently lower bounds on the maximum achievable system rate. However, matching converse bounds are not established for the general non-linearly separable setting. The remaining gap therefore leaves open whether the proposed constructions are optimal or whether other tensor decompositions, task assignments, or coding mechanisms could further reduce the number of required servers. This limitation is closely related to the computational difficulty of sparse and structured factorization problems. The assignment algorithm removes redundant contributions created by overlapping tile closures, but it does not solve

a general minimum-rank tensor factorization problem. Its output depends on the set of candidate tiles and the admissible tuple-to-tile relationships specified by the construction. A broader optimization over tile geometries, support patterns, and factorization orders could potentially produce further improvements, although such an optimization may become computationally intractable for large tensor dimensions.

Chapter 4 also relies on a restrictive structural assumption. The worker-side post-processing operation requires the blockwise cross-product symmetry condition specified in that chapter. This condition ensures that the independently encoded source components can be combined into the polynomial evaluations required for decoding. It does not hold for arbitrary pairs of source matrices. The DSPolyDot construction and its recovery guarantee must therefore be interpreted within this restricted class of matrix pairs. Removing or weakening this condition is necessary before the construction can be applied to general distributed matrix multiplication instances.

Directions for future research. A first cross-cutting direction is to develop adaptive versions of the proposed schemes. Rather than fixing all assignments in advance, a system could update its placement, tiling, or coding decisions according to observed server speeds, network congestion, user demands, or straggling patterns. Such an extension would connect the structural methods developed in this thesis with online scheduling and dynamic resource allocation.

A second direction is to incorporate heterogeneous server capabilities directly into the analytical constructions. In the tensor setting, this would allow the computation budget, communication fan-out, and supported exponent ranges to depend on the server index. The resulting problem could be viewed as a heterogeneous capacitated assignment in which different servers are matched to different tensor regions according to their capabilities. Identifying regimes in which such assignments admit closed-form guarantees would be particularly valuable.

A third direction is to derive converse bounds for the non-linearly separable computation model of Chapter 3. Lower bounds based on tensor support, multilinear rank, communication fan-out, or combinatorial covering arguments could help quantify the optimality gap of the proposed schemes. Even bounds that apply only to particular demand tensors or parameter regimes would clarify when the current constructions are close to optimal.

For Chapter 4, an important objective is to remove the blockwise cross-product symmetry requirement. This may require a different source-side transformation, additional worker messages, or a coding construction that

allows the undesired cross terms to be separated during interpolation. Another relevant extension is to quantify the finite blocklength behavior of the structured source-coding stage, rather than relying only on its asymptotic guarantee. Furthermore, the exact-recovery formulations considered in this thesis could be extended to approximate computation. In many learning and signal processing applications, a controllable approximation error may be acceptable in exchange for lower communication, computation, or latency. Combining function sensitivity, tensor approximation, and progressive coded computation may provide a unified framework in which the quality of the recovered result improves as additional server responses become available.

Finally, experimental validation on distributed computing platforms would help assess the practical trade-offs that are abstracted by the analytical models. Such studies could account for encoding and decoding overhead, memory access, synchronization, finite-field arithmetic, network congestion, and the computational complexity of the assignment algorithms. This would clarify the operating regimes in which exploiting functional and algebraic structure provides measurable end-to-end gains.

Overall, the results of this thesis provide structured foundations rather than universally optimal solutions for all distributed computing systems. They demonstrate that substantial gains can be obtained by moving beyond function-agnostic placement and coding and by explicitly representing the interactions present in the demanded computations. At the same time, the assumptions and open problems identified above highlight the need for adaptive, heterogeneous, and decentralized formulations. Addressing these issues may further bridge the gap between the theoretical limits of structured distributed computation and its deployment in large-scale networked systems.

5.5 Drafts and Publications

The main contributions of this thesis have led to the following drafts and publications.

Journal Draft (Submitted to IEEE Transactions on Communications):

- A. Khalesi, A. Tanha, D. Malak, and P. Elia, “Distributed Computing under Communication Constraints: A Sparse Tensor Factorization Approach”, 2026. arXiv version ([103]) [Online] arXiv:2601.16171v2

Conference Papers:

- A. Tanha and D. Malak, "The Influence of Placement on Transmission in Distributed Computing of Boolean Functions," in Proceedings, IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Lucca, Italy, Sep. 2024. ([100])
  [Online] arXiv:2406.07088.

- A. Khalesi, A. Tanha, D. Malak, and P. Elia, "Non-Linearly Separable Distributed Computing: A Sparse Tensor Factorization Approach," in Proceedings, IEEE International Symposium on Information Theory (ISIT), Guangzhou, China, Jun. 2026. ([102])
  [Online] arXiv:2601.16171v1.

- A. Tanha, M. R. D. Salehi, M. Dutta, and D. Malak, "Cooperative Coded Matrix Multiplication in Secrecy-Constrained Vehicular Networks," in IEEE Vehicular Technology Conference (VTC), to appear, 2026. ([108])
  [Online] HAL:hal-05641781.

Recent Result Posters:

- A. Tanha and D. Malak, "Sensitivity of Distributed Computing to Placement", Recent Results, IEEE European School of Information Theory, 1-5 July 2024, Eindhoven, The Netherlands & France PhD Workshop on Information Theory, 7 June 2024, Paris, France. ([99])
  [Online] EURECOM/publication/7747 & EURECOM/publication/7754

- M. R. D. Salehi, A. Tanha, D. Malak, "Structured Polynomial Codes", Recent Results, IEEE International Symposium on Information Theory (ISIT), 7-12 July 2024, Athens, Greece. ([106])
  [Online] EURECOM/publication/7724

- A. Khalesi, A. Tanha, D. Malak, and P. Elia, "Tessellated Distributed Computing of Non-Linearly Separable Functions", Workshop on Distributed Computing, Optimization & Learning (WDCL), 3-5 September 2025, Munich, Germany. ([101]) [Online] EURECOM/publication/8357

- A. Tanha, M. R. D. Salehi, D. Malak, "Structured Coded Matrix Multiplication", Recent Results, IEEE Communication Theory Workshop (CTW), 4-7 May 2025, Venice, Italy. ([107])
  [Online] EURECOM/publication/8179

5.6 Disclaimer

Use of generative AI tools. ChatGPT (OpenAI) was used as a supporting tool for language editing, reformulation, and the preparation of Figure 1.1. All scientific arguments, mathematical derivations, results, references, and final manuscript content were analyzed, validated, and reviewed by the author, who holds full responsibility for the thesis.

Bibliography

[1] Focus Group on Technologies for Network 2030 (FG NET-2030), "Network 2030 – A Blueprint of Technology, Applications and Market Drivers Towards the Year 2030 and Beyond", International Telecommunication Union (ITU), Technical Report, May 2019.

[2] Ericsson, *Traffic growth in mobile networks driven by 5G*, `https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-traffic-forecast`, 2025.

[3] Nokia, *Global network traffic report*, `https://onestore.nokia.com/asset/213660`, 2025.

[4] Q. Yan, S. Yang, and M. Wigger, "Storage-computation-communication tradeoff in distributed computing: Fundamental limits and complexity", *IEEE Transactions on Information Theory*, vol. 68, no. 8, pp. 5496–5512, 2022.

[5] E. Ozfatura, S. Ulukus, and D. Gündüz, "Straggler-aware distributed learning: Communication-computation latency trade-off", *Entropy*, vol. 22, no. 5, p. 544, 2020.

[6] H. Park, M. Cho, Y. Jang, and J.-W. Choi, "Latency analysis of in-vehicle network for advanced driver assistance system", in *Proceedings, IEEE Vehicular Technology Conference (VTC)*, Washington, DC, USA, Oct. 2024, pp. 1–7.

[7] T. Kosar and M. Livny, "A framework for reliable and efficient data placement in distributed computing systems", *Journal of Parallel and Distributed Computing*, vol. 65, no. 10, pp. 1146–1157, 2005.

[8] B. Bai, "Forget BIT, it is all about TOKEN: Towards semantic information theory for LLMs", *arXiv preprint arXiv:2511.01202*, 2025.

[9] T. Zhang, J. Yi, B. Yao, Z. Xu, and A. Shrivastava, "NoMAD-Attention: Efficient LLM inference on CPUs through multiply-add-free attention", in *Proceedings, Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, Vancouver, Canada, Dec. 2024, pp. 112 706–112 730.

[10] A. Khalesi and M. R. D. Salehi, "Mixture-of-experts under finite-rate gating: Communication–generalization trade-offs", *IEEE Communications Letters*, vol. 30, pp. 1568–1572, 2026.

[11] B. Nazer and M. Gastpar, "Computation over multiple-access channels", *IEEE Transactions on Information Theory*, vol. 53, no. 10, pp. 3498–3516, Sep. 2007.

[12] O. Abari, H. Rahul, and D. Katabi, "Over-the-air function computation in sensor networks", *arXiv preprint arXiv:1612.02307*, Dec. 2016.

[13] A. Şahin and R. Yang, "A survey on over-the-air computation", *IEEE Communications Surveys & Tutorials*, vol. 25, no. 3, pp. 1877–1908, 2023.

[14] S. Razavikia, J. M. Barros da Silva, and C. Fischione, "ChannelComp: A general method for computation by communications", *IEEE Transactions on Communications*, vol. 72, no. 2, pp. 692–706, 2024.

[15] W. Y. B. Lim et al., "Federated learning in mobile edge networks: A comprehensive survey", *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[16] H. Yang, T. Ding, and X. Yuan, "Federated learning with lossy distributed source coding: Analysis and optimization", *IEEE Transactions on Communications*, vol. 71, no. 8, pp. 4561–4576, Aug. 2023.

[17] A. Akram, P. Zimmer, C. Gritti, G. Karame, and M. Önen, "Fundamentals of privacy-preserving and secure machine learning", in *Trustworthy AI in Medical Imaging*, ser. The MICCAI Society Book Series, Academic Press, 2025, pp. 385–409.

[18] A. Aghabagherloo, A. Abadi, S. Sarkar, V. A. Dasu, and B. Preneel, "Impact of data duplication on deep neural network-based image classifiers: Robust vs. standard models", in *Proceedings, IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA, USA, May 2025, pp. 177–183.

[19] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning", in *Proceedings, 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2015, pp. 1310–1321.

[20] M. Abadi et al., "Deep learning with differential privacy", in *Proceedings, ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 308–318.

[21] M. Soleymani, H. Mahdavifar, and A. S. Avestimehr, "Privacy-preserving distributed learning in the analog domain", *arXiv preprint arXiv:2007.08803*, 2020.

[22] M. S. Lori, W. Huang, Z. Tian, and J. Cheng, "Thermohydraulic performance of spray cooling systems: A general model by machine learning", *Artificial Intelligence Review*, vol. 59, no. 1, p. 29, 2025.

[23] H. Liu and D. Orban, "Cloud MapReduce: A MapReduce implementation on top of a cloud operating system", in *Proceedings, IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Newport Beach, California, May 2011, pp. 464–474.

[24] D. Dahiphale, R. Karve, A. V. Vasilakos, et al., "An advanced MapReduce: Cloud MapReduce, enhancements and applications", *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 101–115, 2014.

[25] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges", *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[26] K. Li, M. Tao, and Z. Chen, "Exploiting computation replication for mobile edge computing: A fundamental computation-communication tradeoff study", *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4563–4578, 2020.

[27] X. Kong, Y. Wu, H. Wang, and F. Xia, "Edge computing for internet of everything: A survey", *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 472–23 485, 2022.

[28] S. Daei, F. Haddadi, and A. Amini, "Living near the edge: A lower bound on the phase transition of total variation minimization", *IEEE Transactions on Information Theory*, vol. 66, no. 5, pp. 3261–3267, 2019.

[29] A. Khalesi, S. Daei, M. Kountouris, and P. Elia, "Multi-user distributed computing via compressed sensing", in *Proceedings, IEEE Information Theory Workshop (ITW)*, Saint-Malo, France, Apr. 2023, pp. 509–514.

[30] M. Chen et al., "Distributed learning in wireless networks: Recent progress and future challenges", *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[31] H. L. N. Thi et al., "Coded distributed computing for vehicular edge computing with dual-function radar communication", in *Proceedings, IEEE Vehicular Technology Conference (VTC-Fall)*, Hong Kong, Hong Kong, Oct. 2023.

[32] A. Bušić, "Decision and control in networks with stochastic demand and supply", HDR thesis, Habilitation à diriger des recherches, Université Paris-Saclay, 2024.

[33] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[34] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, 2012.

[35] Apache Software Foundation, *Apache Hadoop*, `http://hadoop.apache.org`, 2026.

[36] Apache Hadoop, *Hadoop terasort example*, `https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html`, 2015.

[37] O. O'Malley, "Terabyte sort on Apache Hadoop", Yahoo!, Technical Report, May 2008.

[38] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet, "AlphaSort: A cache-sensitive parallel external sort", *The International Journal on Very Large Data Bases (VLDB)*, vol. 4, no. 4, pp. 603–628, Oct. 1995, ISSN: 1066-8888.

[39] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets", in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, Boston, MA, USA, Aug. 2010.

[40] H. Jin, X. Yang, X.-H. Sun, and I. Raicu, "ADAPT: Availability-aware MapReduce data placement for non-dedicated distributed computing", in *Proceedings, IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2012, pp. 516–525.

[41] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce", in *Proceedings, 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2015, pp. 964–971.

[42] X. Xu, M. Tang, and Y.-C. Tian, "Theoretical results of QoS-guaranteed resource scaling for cloud-based MapReduce", *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 879–889, 2018.

[43] Y. Bi, M. Wigger, and Y. Wu, "Normalized delivery time of wireless MapReduce", *IEEE Transactions on Information Theory*, vol. 70, no. 10, pp. 7005–7022, 2024.

[44] C. E. Shannon, "A mathematical theory of communication", *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[45] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources", *IEEE Transactions on Information Theory*, vol. 19, no. 4, pp. 471–480, 1973.

[46] J. Körner, "Coding of an information source having ambiguous alphabet and the entropy of graphs", in *Proceedings, 6th Prague Conference on Information Theory*, Prague, Czechoslovakia, Sep. 1973, pp. 411–425.

[47] N. Alon and A. Orlitsky, "Source coding and graph entropies", *IEEE Transactions on Information Theory*, vol. 42, no. 5, pp. 1329–1339, Sep. 1996.

[48] A. Orlitsky and J. R. Roche, "Coding for computing", *IEEE Transactions on Information Theory*, vol. 47, no. 3, pp. 903–917, Mar. 2001.

[49] S. Feizi and M. Médard, "On network functional compression", *IEEE Transactions on Information Theory*, vol. 60, no. 9, pp. 5387–5401, Sep. 2014.

[50] J. Körner and K. Marton, "How to encode the modulo-two sum of binary sources", *IEEE Transactions on Information Theory*, vol. 25, no. 2, pp. 219–221, Mar. 1979, Correspondence.

[51] T. S. Han and K. Kobayashi, "A dichotomy of functions $F(X, Y)$ of correlated sources (X, Y)", *IEEE Transactions on Information Theory*, vol. 33, no. 1, pp. 69–76, Jan. 1987.

[52] A. Padakandla and S. S. Pradhan, "Computing sum of sources over an arbitrary multiple access channel", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Istanbul, Turkey, Jul. 2013, pp. 2144–2148.

[53] M. A. Sohail, T. Anwar Atif, and S. S. Pradhan, "Unified approach for computing sum of sources over cq-mac", in *2022 IEEE International Symposium on Information Theory (ISIT)*, Espoo, Finland, Jun. 2022, pp. 1868–1873.

[54] Y. Birk and T. Kol, "Informed-source coding-on-demand (ISCOD) over broadcast channels", in *Proceedings, IEEE International Conference on Computer Communications (INFOCOM)*, vol. 3, San Francisco, CA, USA, Aug. 1998, 1257–1264 vol.3.

[55] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information", *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1479–1494, 2011.

[56] A. Khalesi and P. Elia, "Hyper-minrank: A unified hypergraph characterization of multi-sender index coding", *arXiv preprint arXiv:2512.13615*, 2025.

[57] B. Nazer and M. Gastpar, "Compute-and-forward: Harnessing interference through structured codes", *IEEE Transactions on Information Theory*, vol. 57, no. 10, pp. 6463–6486, 2011.

[58] J. Zhan, B. Nazer, U. Erez, and M. Gastpar, "Integer-forcing linear receivers", *IEEE Transactions on Information Theory*, vol. 60, no. 12, pp. 7661–7685, 2014.

[59] F. Shirani and S. S. Pradhan, "Lattices from linear codes: Source and channel networks", in *2022 IEEE International Symposium on Information Theory (ISIT)*, Espoo, Finland, Jun. 2022, pp. 3238–3243.

[60] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow", *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[61] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding", *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[62] S. S. Pradhan, A. Padakandla, and F. Shirani, "An algebraic and probabilistic framework for network information theory", *Foundations and Trends in Communications and Information Theory*, vol. 18, no. 2, pp. 173–379, Jan. 2021, ISSN: 1567-2190.

[63] S. Li, M. A. Maddah-Ali, and S. Avestimehr, "Coded distributed computing: Straggling servers and multistage dataflows", in *Allerton Conference on Communication, Control, and Computing*, 2016, pp. 164–171.

[64] M. V. Jamali, M. Soleymani, and H. Mahdavifar, "Coded distributed computing: Performance limits and code designs", in *Proceedings, IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

[65] C.-S. Yang and A. S. Avestimehr, "Coded computing for secure Boolean computations", *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 326–337, 2021.

[66] K. Wan, M. Ji, and G. Caire, "Topological coded distributed computing", in *Proceedings, IEEE Global Communications Conference (GLOBECOM)*, 2020, pp. 1–6.

[67] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Compressed coded distributed computing", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2032–2036.

[68] E. Ozfatura, S. Ulukus, and D. Gündüz, "Coded distributed computing with partial recovery", *IEEE Transactions on Information Theory*, vol. 68, no. 3, pp. 1945–1959, 2022.

[69] F. Brunero, K. Wan, G. Caire, and P. Elia, "Coded distributed computing for sparse functions with structured support", in *Proceedings, IEEE Information Theory Workshop (ITW)*, Saint-Malo, France, Apr. 2023, pp. 474–479.

[70] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy", in *Proceedings, International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019, pp. 1215–1225.

[71] M. Soleymani, H. Mahdavifar, and A. S. Avestimehr, "Analog lagrange coded computing", *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 283–295, 2021.

[72] A. Lenz, R. Bitar, A. Wachter-Zeh, and E. Yaakobi, "Function-correcting codes", *IEEE Transactions on Information Theory*, vol. 69, no. 9, pp. 5604–5618, 2023.

[73] R. Premlal and B. S. Rajan, "On function-correcting codes", *IEEE Transactions on Information Theory*, vol. 71, no. 8, pp. 5884–5897, 2025.

[74] N. Charalambides, H. Mahdavifar, and A. O. Hero, "Generalized fractional repetition codes for binary coded computations", *IEEE Transactions on Information Theory*, vol. 71, no. 3, pp. 2170–2194, 2025.

[75] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 2900–2904.

[76] D. Fathollahi and M. Mondelli, "Polar coded computing: The role of the scaling exponent", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Espoo, Finland, Jun. 2022, pp. 2154–2159.

[77] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "OverSketch: Approximate matrix multiplication for the cloud", in *Proceedings, IEEE International Conference on Big Data*, 2018, pp. 298–304.

[78] T. Jahani-Nezhad and M. A. Maddah-Ali, "Codedsketch: A coding scheme for distributed computation of approximated matrix multiplication", *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 4185–4196, 2021.

[79] D. P. Woodruff, "Sketching as a tool for numerical linear algebra", *Foundations and Trends in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.

[80] O. Ordentlich and Y. Polyanskiy, "Optimal quantization for matrix multiplication", *IEEE Transactions on Information Theory*, vol. 72, no. 3, pp. 1943–1972, 2026.

[81] M. Rudow, N. Charalambides, A. O. Hero, and K. V. Rashmi, "Compression-informed coded computing", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, 2023, pp. 2177–2182.

[82] N. Agrawal et al., "A learning-based approach to approximate coded computation", in *Proceedings, IEEE Information Theory Workshop (ITW)*, 2022, pp. 600–605.

[83] M. Egger, R. Bitar, A. Wachter-Zeh, and D. Gündüz, "Efficient distributed machine learning via combinatorial multi-armed bandits", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Espoo, Finland, Jun. 2022, pp. 1653–1658.

[84] S. Salamatian, N. Shlezinger, Y. C. Eldar, and M. Médard, "Task-based quantization for recovering quadratic functions using principal inertia components", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Paris, France, Jul. 2019.

[85] O. A. Hanna, Y. H. Ezzeldin, T. Sadjadpour, C. Fragouli, and S. Diggavi, "On distributed quantization for classification", *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 237–249, Apr. 2020.

[86] A. Mostaani, T. X. Vu, S. Chatzinotas, and B. Ottersten, *Task-based information compression for multi-agent communication problems with channel rate constraints*, Submitted to IEEE Transactions on Signal Processing, May 2020.

[87] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes", *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[88] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic MDS codes and expander graphs", *IEEE Transactions on Information Theory*, vol. 66, no. 12, pp. 7475–7489, 2020.

[89] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning", *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 277–291, 2020.

[90] T. Jahani-Nezhad and M. A. Maddah-Ali, "Optimal communication-computation trade-off in heterogeneous gradient coding", *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 1002–1011, 2021.

[91] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2403–2407.

[92] A. B. Das, A. Ramamoorthy, and N. Vaswani, "Efficient and robust distributed matrix computations via convolutional coding", in *Proceedings, IEEE International Symposium on Information Theory (ISIT 2021)*, Melbourne, Australia, Jul. 2021, pp. 1724–1729.

[93] A. B. Das and A. Ramamoorthy, "Distributed matrix-vector multiplication: A convolutional coding approach", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 3022–3026.

[94] A. Ramamoorthy and A. B. Das, "A survey on coded matrix computations and gradient coding", *IEEE BITS the Information Theory Magazine*, pp. 1–12, 2026.

[95] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication", in *Proceedings, Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, Long Beach, CA, USA, Dec. 2017, pp. 4403–4413.

[96] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication", in *Proceedings, International Conference on Machine Learning (ICML)*, PMLR, 2018, pp. 5152–5160.

[97] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions", *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, 1990.

[98] R. Bitar, M. Xhemrishi, and A. Wachter-Zeh, "Adaptive private distributed matrix multiplication", *IEEE Transactions on Information Theory*, vol. 68, no. 4, pp. 2653–2673, 2022.

[99] A. Tanha and D. Malak, *Sensitivity of distributed computing to placement*, Recent Results, IEEE European School of Information Theory (ESIT) & France Workshop on Information Theory, Eindhoven, The Netherlands & Paris, France, Jul. 2024.

[100] A. Tanha and D. Malak, "The influence of placement on transmission in distributed computing of Boolean functions", in *Proceedings, IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Lucca, Italy, Sep. 2024.

[101] A. Khalesi, A. Tanha, D. Malak, and P. Elia, *Tessellated distributed computing of non-linearly separable functions*, Recent Results, Workshop on Distributed Computing, Optimization, and Learning (WDCL), Munich, Germany, Sep. 2025.

[102] A. Khalesi, A. Tanha, D. Malak, and P. Elia, "Non-linearly separable distributed computing: A sparse tensor factorization approach", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Guangzhou, China, Jun. 2026.

[103] A. Khalesi, A. Tanha, D. Malak, and P. Elia, "Multi-user non-linearly separable distributed computing", *arXiv preprint arXiv:2601.16171v2*, 2026.

[104] A. Khalesi and P. Elia, "Multi-user linearly-separable distributed computing", *IEEE Transactions on Information Theory*, vol. 69, no. 10, pp. 6314–6339, 2023.

[105] A. Khalesi and P. Elia, "Tessellated distributed computing", *IEEE Transactions on Information Theory*, vol. 71, no. 6, pp. 4754–4784, 2025.

[106] M. R. Deylam-Salehi, A. Tanha, and D. Malak, *Structured polynomial codes*, Recent Results, IEEE International Symposium on Information Theory (ISIT), Athens, Greece, Jul. 2024.

[107] A. Tanha, M. R. D. Salehi, and D. Malak, *Structured coded matrix multiplication*, Recent Results, IEEE Communication Theory Workshop (CTW), Venice, Italy, May 2025.

[108] A. Tanha, M. R. D. Salehi, M. Dutta, and D. Malak, "Cooperative coded matrix multiplication in secrecy-constrained vehicular networks", in *IEEE 103rd Vehicular Technology Conference (VTC)*, to appear, Nice, France, Jun. 2026.

[109] D. Malak, "Distributed structured matrix multiplication", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Athens, Greece, Jul. 2024.

[110] D. Malak, "Structured codes for distributed matrix multiplication", *IEEE Transactions on Information Theory*, vol. 72, no. 7, pp. 5160–5182, Jul. 2026.

[111] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding", *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1920–1933, 2020.

[112] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication", *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 278–301, 2020.

[113] H. Li, Y. Chen, K. W. Shum, and C. W. Sung, "Data allocation for approximate gradient coding in edge networks", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Taipei, Taiwan, 2023, pp. 2541–2546.

[114] M. Sefidgaran and A. Tchamkerten, "Computing a function of correlated sources: A rate region", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, St. Petersburg, Russia, 2011, pp. 1856–1860.

[115] M. Sefidgaran, A. Gohari, and M. R. Aref, "On Körner-Marton's sum modulo two problem", in *Proceedings, Iran Workshop on Communication and Information Theory*, May 2015, pp. 1–6.

[116] D. Malak, "Distributed computing of functions of structured sources with helper side information", in *Proceedings, IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Shanghai, China, Sep. 2023, pp. 276–280.

[117] D. Malak, M. R. Deylam Salehi, B. Serbetci, and P. Elia, "Multi-server multi-function distributed computation", *Entropy*, vol. 26, no. 6, p. 448, 2024.

[118] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing", *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, 2018.

[119] K. Wan, H. Sun, M. Ji, and G. Caire, "Distributed linearly separable computation", *IEEE Transactions on Information Theory*, vol. 68, no. 2, pp. 1259–1278, Feb. 2022.

[120] A. Khalesi and M. R. Deylam Salehi, "Typical solutions of multi-user linearly-decomposable distributed computing", *IEEE Networking Letters*, vol. 8, pp. 10–13, 2026.

[121] F. Mirkarimi, S. Rini, and N. Farsad, "Benchmarking neural capacity estimation: Viability and reliability", *IEEE Transactions on Communications*, vol. 71, no. 5, pp. 2654–2669, 2023.

[122] J. Bourgain, J. Kahn, G. Kalai, Y. Katznelson, and N. Linial, "The influence of variables in product spaces", *Israel Journal of Mathematics*, vol. 77, pp. 55–64, 1992.

[123] S. Jukna, *Boolean Function Complexity: Advances and Frontiers*. Springer, 2012.

[124] S. Jin, Y. Cui, H. Liu, and G. Caire, "Uncoded placement optimization for coded delivery", in *Proceedings, International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, Shanghai, China, May 2018.

[125] R. Heckel, S. Schober, and M. Bossert, "Harmonic analysis of Boolean networks: Determinative power and perturbations", *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2013, no. 6, May 2013.

[126] A. Biswas and P. Sarkar, "Influence of a set of variables on a Boolean function", *SIAM Journal on Discrete Mathematics*, vol. 37, no. 3, pp. 2148–2171, 2023.

[127] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning", *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, 2020.

[128] A. S. Avestimehr, S. M. M. Kalan, and M. Soltanolkotabi, "Fundamental resource trade-offs for encoded distributed optimization", *Information and Inference: A Journal of the IMA*, vol. 10, no. 1, pp. 231–260, 2021.

[129] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning", in *Proceedings, International Conference on Machine Learning (ICML)*, PMLR, Sydney, Australia, Aug. 2017, pp. 3368–3376.

[130] M. Ye and E. Abbe, "Communication-computation efficient gradient coding", in *Proceedings, International Conference on Machine Learning (ICML)*, PMLR, Stockholm, Sweden, Jul. 2018, pp. 5610–5619.

[131] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform", *arXiv preprint arXiv:1804.09791*, 2018.

[132] S. Vithana and S. Ulukus, "Private read-update-write with controllable information leakage for storage-efficient federated learning with top-r sparsification", *IEEE Transactions on Information Theory*, vol. 70, no. 5, pp. 3669–3692, 2023.

[133] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, Barcelona, Spain, Dec. 2016.

[134] A. Ramamoorthy, L. Tang, and P. O. Vontobel, "Universally decodable matrices for distributed matrix-vector multiplication", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Paris, France, Jul. 2019, pp. 1777–1781.

[135] J. Maheri, K. K. K. Namboodiri, and P. Elia, "Universal and asymptotically optimal data and task allocation in distributed computing", *arXiv preprint arXiv:2601.05873*, 2026.

[136] Z. Jia and S. A. Jafar, "On the capacity of secure distributed batch matrix multiplication", *IEEE Transactions on Information Theory*, vol. 67, no. 11, pp. 7420–7437, 2021.

[137] K. Wan, H. Sun, M. Ji, and G. Caire, "On secure distributed linearly separable computation", *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 3, pp. 912–926, 2022.

[138] M. Sefidgaran, A. Zaidi, and P. Krasnowski, "Minimum description length and generalization guarantees for representation learning", in *Proceedings, Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[139] M. Kavian, R. Chor, M. Sefidgaran, and A. Zaidi, "Heterogeneity matters even more in distributed learning: Study from generalization perspective", *arXiv preprint arXiv:2503.01598*, 2025.

[140] D. Haziza, B. Steiner, E. Yang, M. Sirotenko, B. Jacob, and A. Vaswani, "2:4 sparsity for activation functions in LLMs", in *Proceedings, International Conference on Learning Representations (ICLR)*, Singapore, May 2025.

[141] A. Hassani et al., "Generalized neighborhood attention: Multi-dimensional sparse attention at the speed of light", *arXiv preprint arXiv:2504.16922*, 2025.

[142] S. M. Jayakumar et al., "Multiplicative interactions and where to find them", in *Proceedings, International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020.

[143] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "CodedReduce: A fast and robust framework for gradient aggregation in distributed learning", *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 148–161, 2022.

[144] P. Moradi, H. Akbari-Nodehi, and M. A. Maddah-Ali, "General coded computing: Adversarial settings", in *Proceedings, IEEE International Symposium on Information Theory (ISIT)*, Ann Arbor, MI, USA, Jun. 2025, pp. 1–6.

[145] N. A. Khooshemehr and M. A. Maddah-Ali, "Vers: Coded computing system with distributed encoding", *IEEE Transactions on Information Theory*, vol. 71, no. 10, pp. 7609–7625, 2025.

[146] A. Decurninge, I. Land, and M. Guillaud, "A tensor-based approach to massive random access", in *Proceedings, 54th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Nov. 2020, pp. 1147–1151.

[147] S. A. Vavasis, "On the complexity of nonnegative matrix factorization", *SIAM Journal on Optimization*, vol. 20, no. 3, pp. 1364–1377, 2009.

[148] K. Hanauer, M. P. Seybold, and J. Unterweger, "Covering rectilinear polygons with area-weighted rectangles", in *Proceedings, Symposium on Algorithm Engineering and Experiments (ALENEX)*, SIAM, 2024, pp. 157–169.

[149] C. J. Hillar and L.-H. Lim, "Most tensor problems are NP-hard", *Journal of the ACM*, vol. 60, no. 6, pp. 1–39, 2013.

[150] Q.-T. Le, E. Riccietti, and R. Gribonval, "Spurious valleys, NP-hardness, and tractability of sparse matrix factorization with fixed support", *SIAM Journal on Matrix Analysis and Applications*, vol. 44, no. 2, pp. 503–529, 2023.

[151] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, "A multilinear singular value decomposition", *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[152] S. V. Dolgov and D. V. Savostyanov, "Alternating minimal energy methods for linear systems in higher dimensions", *SIAM Journal on Scientific Computing*, vol. 36, no. 5, A2248–A2271, 2014.

[153] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications", *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[154] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[155] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2022.

[156] A. Tanha, *Sparse Tensor Factorization with Combinatorial Reduction*, `https://github.com/ahmadtanhaa/MUNLSDC`.

[157] J. Shamsi, M. A. Khojaye, and M. A. Qasmi, "Data-intensive cloud computing: Requirements, expectations, challenges, and solutions", *Journal of Grid Computing*, vol. 11, no. 2, pp. 281–310, Jun. 2013.

[158] A. Toshniwal et al., "Storm@Twitter", in *Proceedings, ACM SIGMOD International Conference on Management of Data*, ACM, 2014, pp. 147–156.

[159] F. Xiao and D. T. M. Slock, "Performance analysis of hyperparameter optimization in sparse bayesian learning via Stein's unbiased risk estimator", in *33rd European Signal Processing Conference (EUSIPCO)*, 2025.

[160] Z. Zhao, C. Forsch, L. Cottatellucci, and D. T. M. Slock, "Distributed iterative ML and message passing for grant-free cell-free massive MIMO systems", in *Proceedings, IEEE Middle East Conference on Communications and Networking (MECOM)*, 2025.

[161] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load", *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2722–2734, Feb. 2020.

[162] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Collaborative decoding of polynomial codes for distributed computation", in *Proceedings, IEEE Information Theory Workshop (ITW)*, Visby, Sweden, Aug. 2019, pp. 1–5.

[163] A. Fidalgo-Díaz and U. Martínez-Peñas, "Distributed matrix multiplication with straggler tolerance using algebraic function fields", *IEEE Transactions on Information Theory*, vol. 71, no. 2, pp. 996–1006, 2025.

[164] J. Li, S. Li, and C. Xing, "Algebraic geometry codes for distributed matrix multiplication using local expansions", *IEEE Transactions on Information Theory*, vol. 72, no. 2, pp. 946–960, 2026.

[165] Y. Kuang, J. Li, S. Li, and C. Xing, "Coded distributed (batch) matrix multiplication over galois ring via rmfe", *IEEE Transactions on Information Theory*, vol. 71, no. 11, pp. 8358–8367, 2025.

[166] L. Hu, J. Shao, Z. Wang, and X. Zhu, "Grouped systematic matdot codes for three-dimensional coding of distributed matrix multiplication", *IEEE Transactions on Communications*, vol. 73, no. 9, pp. 7032–7043, 2025.

[167] B. Brock and R. Golin, "Slicing is all you need: Towards a universal one-sided algorithm for distributed matrix multiplication", in *SC25-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, St. Louis, MO, USA, Nov. 2025, pp. 1276–1288.

[168] J. Gómez-Vilardebò, "Multivariate polynomial codes for efficient matrix chain multiplication in distributed systems", in *Proceedings, IEEE International Conference on Communications (ICC)*, Glasgow, United Kingdom, May 2026, pp. 1–6.

[169] A. Morteza and R. A. Chou, "Distributed batch matrix multiplication: Trade-offs in download rate, randomness, and privacy", *IEEE Transactions on Information Theory*, pp. 1–1, 2026.

[170] D. Malak, "Structured polynomial codes for distributed matrix multiplication", *IEEE Journal on Selected Areas in Information Theory*, vol. 7, pp. 175–191, Apr. 2026.

[171] R. Ahlswede and T. Han, "On source coding with side information via a multiple-access channel and related problems in multi-user information theory", *IEEE Transactions on Information Theory*, vol. 29, no. 3, pp. 396–412, 1983.

[172] V. Lalitha, N. Prakash, K. Vinodh, P. V. Kumar, and S. S. Pradhan, "Linear coding schemes for the distributed computation of subspaces", *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 4, pp. 678–690, Apr. 2013.

[173] R. G. Gallager, *Information Theory and Reliable Communication*. Wiley: New York, 1968.