UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Master Degree: Computer Engineering

Curriculum: Bioinformatics

Computational methods to analyze biological networks from transcriptomics data

Student:

Lucchiari Alessandro

Supervisors:

Prof. Baruzzo Giacomo, PhD

Prof. Di Camillo Barbara, PhD

Dott. Cesaro Giulia

Academic Year: 2024-2025

Graduation Date: 5th December 2024

Abstract

The regulation of biological processes within each single cell governs all the main cellular mechanisms aimed at the development, differentiation, and proper maintenance of the cell itself. It determines, at the tissue level and more generally at the organism level, the actual role assumed by the cell. It is of vital interest to the scientific and biological community to establish precisely how such regulatory processes are concretized in various cell types, whether they are considered healthy or diseased. Understanding these mechanisms necessarily involves identifying and sequencing the gene products that each cell synthesizes and possesses at a given moment. The technique at the core of this cellular study is called single-cell analysis, which is capable of generating multi-omics data that represent all the main regulatory mechanisms occurring within the isolated and analyzed cells. This technology can be considered as the basis for a more accurate study of the roles' heterogeneity exploited by cells within their respective tissues. In this regard, an effective representation of such processes is provided by gene regulatory networks, a tool of particular interest to the biological and scientific community. However, the inference of such networks remains an unresolved issue, with methods that are not simultaneously effective and efficient. This situation is further complicated by the need to study data from single-cell analyses and multi-omics data, which have a significantly larger dimension than the sequencing data obtained from previous generation technologies.

This work aims to analyze two bioinformatics methods for the inference of gene regulatory networks, highlighting their biological potentials and computational limits. Of particular interest, in evaluating these methods for real applicability in experiments with single-cell and multi-omics data, is the scalability of the processes used by such software. This property is also evaluated based on the possibilities for parallelization that these software packages present already in their implementation.

This analysis aims to evaluate potential computational improvements, also utilizing the capabilities of parallel computing on GPU architectures. The main tools used to implement these modifications are software libraries considered state-of-the-art in parallel computing, coupled with GPU structures capable of fully exploiting their potential.

The improved versions of the two bioinformatics methods show significantly reduced execution times compared to those obtained from the original version, with comparable use of memory and resources.

Index

1. Introduction

The ability of a cell to differentiate itself and to modify its biological processes in response to external stimuli is a fundamental characteristic of living organisms [1,2,3]. The instructions for performing these adaptive mechanisms are encoded within DNA, owned by each cell of the same organism. Despite sharing an identical genetic blueprint, cells can perform highly specialized functions, fulfilling distinct roles in the same organism. This apparent paradox is explained by the central dogma of biology [4,5], which states the unidirectional flow of genetic information: from DNA to RNA, through the transcription process, and from RNA to protein via translation. This tightly regulated process controls the protein synthesis, enabling the cell to orchestrate all the biological pathways essential for the organism. The synthesis of a specific protein can be up or downregulated or modulated by the promotion or the inhibition of the expression of each gene contained in the DNA.

Gene expression is the process through which a specific gene product is synthesized and involves multiple layers of regulation, including signal transduction and post-transcriptional modification [6,7]. Among these, transcriptional regulation has an important role: the regulation of the transcription process can be modulated by specialized proteins, such as repressors and promoters. In this set of controlling proteins, a fundamental role is exploited by the Transcription Factors (TF). These proteins can bind to specific sites in the DNA sequence (regulatory or binding sites), and orchestrate the regulation of the downstream target genes. The transcriptional activity of these genes, reflected in the quantity of RNA transcripted and subsequently translated into protein, depends directly on the action of this type of protein. Transcription factors, as proteins themselves, are subject to a multitude of regulatory mechanisms, including feedback loops and interactions with other signaling molecules. This further degree of regulation adds a complexity layer to this fundamental biological process and underlines the importance of all the TFs involved.

For the scientific community, understanding the details of regulatory processes and comprehensively evaluating the biological mechanisms occurring within the cells of a considered sample can be considered a crucial goal. In this sense, the development of a model capable of representing these regulatory processes in a structured and biologically meaningful form is essential and has found a realization in the gene regulatory network (GRN) [8,9]. Visualized in Figure 1, this representation can collect all the interactions among all the regulatory components, such as TF, and repressors, using a coherent structure.

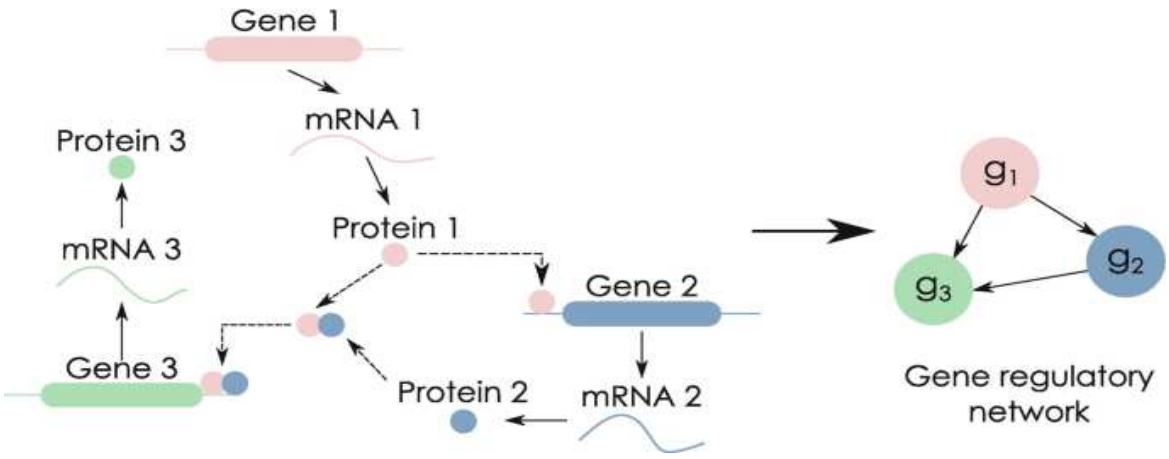


Figure 1: Graphical representation of a GRN: each node is a gene, a protein, or a biological process, and each arc represents the regulation executed from one node on the other. Image taken from [10].

The evaluation of the gene regulation of a given sample can be conducted by quantifying the RNA copies produced within the sample's cells, through a next-generation sequencing technique called RNA-seq [11,12]. This analysis can be classified as next-generation sequencing because exploits a high-throughput approach to sequence a massive number of reads, fragments of 50 to 400 bases obtained by the double-stranded DNA complementary to the RNA analyzed fragments (ds-cDNA). While the first part of this procedure is executed in vivo in a wet lab and has as output the sequences of the basis of each read considered, the so-called secondary analysis is executed in silico and, after steps needed to identify which gene is responsible for the presence of each read, generates as output a gene expression matrix. This matrix is composed of the amount of RNA copies associated with each gene (the rows) and each sample (the columns). One of the most relevant parameters of this matrix is the sequencing depth: the number of RNA copies annotated for each sample.

Once the expression matrix has been generated, the *tertiary* analysis can be executed aiming to obtain relevant information about the statistically significant differences among all the samples considered, such as the differential abundance of a set of genes of a specific sample subset. This last phase has become even more significant thanks to a new sequencing approach: the single-cell technology.

1.1 Single-cell technology

A relevant limitation of the sequencing and analysis methods presented so far is the inability to capture the biological materials synthesized at the individual cellular level. The specific biological processes that each cell carries out in response to external stimuli or during its differentiation are aggregated together and consequently obscured. This limitation hinders the

identification and high-resolution representation of the heterogeneity exhibited by individual cells within a sample, a critical feature in organisms where cellular differentiation is an impactful property.

The sequencing technique that has revolutionized these aspects, changing the perspective from analyzing groups of cells to considering each cell as a distinct and separable element, is single-cell technology [13]. This technique can be defined and recognized as a cellular-level sequencing process capable of quantifying the biological products synthesized by each cell within a sample. This technology goes beyond just the analysis and sequencing of the cellular transcriptome; it extends its potential to the characterization of the proteome and genome of individual cells, opening a new and revolutionary way to evaluate the biological processes occurring in the examined tissue at the moment of detection.

This technique was developed with the idea and goal of highlighting the specific biological processes of each cell belonging to the considered tissue or sample, in particular, to evaluate all the biological compounds, such as proteins, nucleic acids like RNA in its various forms, and DNA, that a cell possesses at the time of analysis. The key principles that distinguish this technique from previous ones can be identified as follows:

- *Cell isolation*: Each cell considered in this type of analysis results distinctly separated from others, ensuring a more accurate representation of sample heterogeneity and avoiding biological contamination between distinct cells.
- *Measurement sensitivity*: The biological compounds of each cell can be quantified more precisely, despite the different scales of quantities they represent compared to the genetic and protein materials measured by sequencing technologies that operated at the level of cell groups.
- *Cellular resolution*: Single-cell technology prioritizes the identification of biological materials assembled at the cellular level. This key principle must be actualized through the ability to distinguish cellular differences, even when they are minimal, such as in cells with similar functions and where comparable biological mechanisms are applied.

The first fundamental step in conducting a single-cell sequencing analysis is the identification and isolation of cells deemed biologically relevant from the target organism. This phase can be executed through several techniques that, all presenting high sensitivity and cellular resolution, can accurately preserve the heterogeneity of the sample. Common single-cell isolation techniques include sorting, which, through fluorescence (FACS) or magnetism (MACS), can label with detectable antibodies selected cells considered interesting for study

during the identification phase; microfluidics, in which individual cells are isolated using miniaturized devices for the flow of microscopic fluids; and micromanipulation, where micropipettes and lasers are used to collect the biological materials contained within the identified cells. Once the relevant cells for the considered study case are identified and isolated, the collected biological material has to be sequenced. The characteristics of this phase naturally differ based on the type of biological material that has been collected from the cells, which usually, given the scientific relevance of such topics, involves the proteome, genome, or transcriptome. Transcriptome analysis is particularly important and interesting as it provides critical insights into the biological processes occurring within each cell, characterizing its role in the context of the analyzed sample [14].

This sequencing technology focuses on the gene expression of each cell considered in the reference sample. The main biological material of interest for this analysis is the messenger RNA (mRNA) produced during the transcription phase of the genetic information contained within the DNA. Considering the process carried out to perform this analysis, the isolation of individual cells and the extraction and purification of the mRNA are followed by the amplification and conversion of this nucleic acid into complementary DNA (cDNA) through the reverse transcription reaction. To ensure stability and traceability, each cDNA read is uniquely tagged with a molecular barcode (UMI). This barcode allows the association of each read with its cell of origin, preserving essential cell-specific information during subsequent processing. Following this initial phase, sequencing is carried out using high-resolution sequencers such as Illumina or Nanopore. The characteristics and advantages of NGS technologies can also be applied in this type of analysis, ensuring the sequencing of a large number of reads and thus representing even low-expressed genes within individual cells. [15]

The bioinformatics pipeline following this sequencing phase is similar to the one usually employed for bulk cell transcriptome analysis. The main activities that can be identified in this pipeline, in order of execution, are data pre-processing to remove technical errors and low-quality sequencing reads; quantification of gene expression by associating each read with the gene from which it was transcribed; cell clustering, where subgroups or communities of cells with similar transcriptomic characteristics can be identified from the gene expression matrix, or evaluating differences in the abundance of detected gene expression; and finally, visualizing and biologically interpreting the results through dimensionality reduction operations and statistical analysis of the obtained data.

At the end of the secondary analysis process, a gene expression matrix is generated where each column is uniquely associated with the gene expression of a single cell, providing the

possibility to analyze the heterogeneity of the considered cell sample with high resolution and high sensitivity, characteristics that are entirely unattainable using previous sequencing technologies. Another significant advantage of this type of technology is its applicability; in fact, this method imposes no limits regarding the type of cells whose transcriptome can be analyzed, leading to a series of possible case studies ranging from the spatial analysis of tumor processes to the investigation of the development of drug resistance in specific cells.

Despite the evident advantages that this technique brings to the study of the transcriptome, some limitations can become important challenges to the application of this technology, such as:

- Complexity in sample preparation, requiring high accuracy and sensitivity during the initial phase of isolation and processing of messenger RNA fragments;
- Cost and need for advanced instrumentation, requiring highly specialized laboratory tools;
- Management of extremely large datasets, due to the increase in data collected from cellular-level analysis, which can become a significant limitation regarding computational capacity, especially in the case of comparing gene expression between different patient groups;
- The necessity of specialized bioinformatics software to analyze this type and large volume of data, an aspect that programs must manage in terms of both time efficiency and memory usage.

While this sequencing technique provides data about the gene expression on an individual cell level, other analyses generate information about other relevant aspects of the molecular biology of the sample considered. For example, the study of the genome, the proteome, the microbiome, and the epigenome can produce relevant notions able to characterize the biological processes of the considered organisms. A holistic approach able to combine all these types of data, generically called ‘multi-omics’ data, is the horizon for the scientific community. This new type of approach needs to interface with, not only different data types but also, an unprecedented scale of data. The complex biological big data, originating from the combination of multi-omics data, introduces a unique set of challenges: to be analyzed it demands significant computational requirements (in terms of optimized software and advanced hardware) that can become the bottleneck for the entire workflow analysis process.

A relevant case study in which the computational requirements can be considered as a possible process limitation is the GRN inference.

1.2 Gene Regulatory Network Inference

The ability to infer biologically significant GRNs has become essential for the scientific community. A reliable GRN that accurately represents the biological processes in a patient's cell sample can provide critical insights. In particular, areas and purposes where the inference of this tool can be decisive include the analysis of the characteristics of GRNs associated with patients suffering from diseases that have altered regulation of expression, such as diabetes, cancer, or autoimmune disorders. It is therefore evident that the representative capacity of a GRN should be regarded as the main characteristic to be obtained from the inference of the network itself.

Despite the importance of this tool and the attention of the scientific community towards the development and research in this field, some elements slow down or prevent the implementation of a representative and efficient GRN inference method, especially when using data not derived from single-cell technologies [16,17].

The first issue is biological complexity: developing an inference method that can accurately represent biological processes despite the inability to capture the intrinsic cellular heterogeneity of the samples and tissues analyzed in a case study becomes a challenging problem. In particular, the gene expression deduced from bulk data analysis represents a cumulative view of all the biological processes occurring within the cells composing the sample, and thus the inference of a network from this type of data will not be able to represent the intrinsic heterogeneity of the sample itself. An additional layer of complexity in GRN inference is represented by the non-linearity of gene regulations. Indeed, the mechanisms that orchestrate regulatory processes cannot all be effectively inferred using a linear model, as there are many biological mechanisms regulated through activation or repression thresholds. These types of regulation do not guarantee a proportional relationship between changes in the expression of a transcription factor (TF) and the actual changes in the expression of the target gene. Another example of biological complexity that is challenging to represent using additive models is regulation through protein complexes: TFs can be combined to carry out their regulatory functions and the variation of just one of these components does not directly cause a proportional increase in gene expression unless it is adequately assisted with the regulation of the other TFs.

Alongside these biological limitations, there is a computational problem: considering the need to represent the regulation of several genes in the order of tens of thousands, the amount of data to be inferred—i.e., the weight of each arc connecting two generic nodes—becomes

difficult to manage in terms of execution time and memory required when using sequential computational approaches. In addition to these two significant limitations, other problems arise, such as the effects of noise or the incompleteness of sequencing data on the inferred network, the difficulty of scientifically validating whether the obtained network is representative of the acquired biological sample or the challenge of inferring and interpreting GRNs based on an approach that uses only one type of data.

Over the years, various methods for inferring these types of networks have been implemented, based on approaches that tackled this problem from different perspectives, yielding methods with differing characteristics and performances. The first approach is based on calculating the correlation between the expression of each pair of genes, leveraging statistical tools such as Pearson's correlation coefficient. This set of methods aims to identify the nature of regulatory relationships assuming a dependency (linear in the case of calculating Pearson's correlation coefficient, non-linear when using Spearman's correlation coefficient [18,19]) between each regulatory gene and its respective target gene. Such an assumption, as explained previously, is reductive for a significant number of biological mechanisms, preventing the inferred networks from effectively representing more complex or indirect regulatory relationships.

Another type of approach for inferring GRNs is based on linear regression methodologies [20]. By leveraging the potential of regression techniques or penalized regression, such as LASSO and elastic net, it is possible to infer the regulatory relationships between regulatory genes and target genes, resulting in GRNs that are more representative than those obtained through simpler methods.

One of the most recent and relevant approaches has been achieved with Bayesian networks [21]: probabilistic models that leverage directed acyclic graphs (DAGs) to represent, in this case, the conditional dependencies inferred among the considered genes. This approach, despite the high computational cost in the case of large datasets, proves particularly effective in inferring the relationships of gene expression regulation, maintaining robustness even with noisy input datasets. Another improvement that this approach has implemented, compared to the previously presented, is the ability to model and infer causal relationships between genes, establishing which one regulates the other. The main limitations of this methodology lie in the indispensable use of directed acyclic graphs, which consequently prevent the representation of cyclic regulatory biological processes, and the computational inefficiency in the case of large datasets. This latter limitation, in particular, is the most stringent from the perspective of computational biology, which is oriented towards analyzing large quantities of data, such as those originating from single-cell analyses or multi-omics approaches.

1.3 Gene Regulatory Network Inference on single-cell RNA sequencing data

The inference of a GRN aimed at accurately representing the regulatory biological processes of gene expression at the cellular level is currently one of the most important fields of research and development in computational biology. The ability to infer GRNs from single-cell gene expression data would be considered a fundamental tool for studying, analyzing, and validating scientific theories regarding the mechanisms that lead each single cell to characterize the synthesis of its genetic products, such as RNA and proteins. The advantages and possibilities that this tool could bring are related to the level of resolution that this type of data, provided as input, can guarantee, along with the use of a biologically representative tool such as a GRN [22]. These can be summarized as:

A more accurate representation of the heterogeneity of a biological sample: By isolating, characterizing, and abstracting the processes that occur in every single cell, rather than relying on the cumulative gene expression of bulk data, it is possible to represent the heterogeneity within a sample in a specific and accurate GRN.

Creating GRNs representing the metabolic processes of transition and differentiation: By considering only the cells that are differentiating, such as stem cells, or that are changing their gene expression, as in the case of tumor cells, the GRN obtained would represent the common processes regulating such complex mechanisms.

Obtaining GRNs of specific cells: By enhancing accuracy at the single-cell level, the study of cellular biological processes can be directed towards specific cells deemed scientifically relevant, whose gene expression could not be identified by considering the common properties of bulk cells.

The methods that have been studied and developed for the inference of GRNs from gene expression data can also be applied to this particular data type. At the same time, the limitations and issues characterizing these methods are amplified by the different scales of complexity that this type of biological data presents. In particular, methods that rely on calculating the correlation between gene pairs to establish the expression regulation present additional limitations when applied to scRNA-seq data: due to the heterogeneity of these data, inferring the potentially unique mechanisms regulating the processes of each single cell becomes even more complex. Linear regression models can improve their performance in terms of biological accuracy by leveraging a greater amount of data; however, they remain limited by the requirement of linearity of the relationship in gene expression regulation.

Similarly, Bayesian models, despite the robustness to noise commonly found in single-cell data and the capacity to represent more complex and realistic gene regulatory relationships than previous models, have representation limitations due to the need to use directed acyclic networks. Another fundamental limitation of this approach is the computational scalability, which hinders the real applicability of these methods on significant-sized datasets, such as those produced by single-cell sequencing.

More advanced approaches for inferring networks from scRNA-seq data leverage the potential of machine learning and deep learning to describe gene regulatory relationships more accurately and complexly. In particular, algorithms like SCENIC [23] or GRNBoost2 [24] are optimized for analyzing large quantities of data, and algorithms that utilize Graph Neural Networks [25] represent a significant increase in inference capabilities, even for the most intricate gene relationships. However, the limitations that these methodologies present are difficult to overcome, as they are due to the intrinsic nature of these approaches, such as the training phase requirements: is challenging to provide in practical reality a considerable amount of data distinct from those used for the real application of the algorithms; and the difficult biological validation of the results represented within the inferred networks.

Using single-cell gene expression data as a starting point introduces additional challenges to the already complex task of GRN inference. These challenges arise from the inherent characteristics of single-cell data, such as technical sequencing errors and the sparsity of the gene expression matrix, which typically results from the nature of gene expression and technical limitations of single-cell RNA sequencing (scRNA-seq) technologies [26]. To address these issues, it is essential to implement robust preprocessing and noise reduction strategies. Key approaches include the imputation of Non-Biological Zeros and data normalization[27,28]. By applying these preprocessing techniques, the biological representativeness of scRNA-seq data can be significantly enhanced, providing a more reliable foundation for GRN inference.

Other complexities related to the study of scRNA-seq data cannot be easily overcome. Limitations such as the high dimensionality of these data have significant impacts on the actual efficiency that GRN inference algorithms can aspire to achieve. Indeed, despite the capacity of certain inference methods to handle large amounts of gene expression data as input, the main obstacle is the temporal and memory efficiency that such approaches can reach using their sequential implementation. An algorithm capable of inferring a GRN from a large single-cell sequencing dataset within an impractical timeframe or requiring excessive

memory is of limited practical utility. The computational efficiency of such methods, using scRNA-seq data as a starting point for inference, becomes fundamentally important in this application. [29]

1.4 Study presentation

From this brief overview of two of the most current objectives of the biological and bioinformatics scientific community, the inference of GRN and the effective integration of scRNA-seq data into the analysis pipeline, it is evident that there is a need for a tool capable of performing these tasks while respecting the time and memory constraints imposed by the real applicability of such methods. The main problem that unites all the methods presented for the inference of GRN from single-cell data is the scalability of these approaches to gene expression matrices with a potentially much higher cell count than those generally considered for bulk data bioinformatics analysis.

Once the importance and complexity inherent to this topic are established, it is necessary to consider an implementation approach for the realization and optimization of inference methods that can handle this scale of biological data and, at the same time, integrate information from different data types to ensure the biological significance and relevance of the results. In this perspective, this study aims to analyze and optimize two algorithms, PANDA and LIONESS [30,31], capable of generating, from multi-omics data, GRNs that represent the biological processes characterizing the considered sample-specific GRNs associated with individual samples or, in the case of input scRNA-seq, individual cells. This study focuses on leveraging parallel computation to execute the most resource-intensive tasks associated with this pair of methods, optimizing both computation time and memory usage. This choice aims to obtain inference algorithms with scalability that allows the workflow to remain executable in real analysis contexts.

This study begins with a presentation of the main software suites developed to efficiently manage parallel computing through the use of GPU architectures. The software presented in Chapter 1 represents the state-of-the-art in this computational field, and its presentation aims to highlight all the performance potential when applied to important processes commonly performed sequentially, even in the bioinformatics field, such as Whole Exome Analysis.

Following the presentation of the state-of-the-art parallel computational methods, the two GRN inference algorithms are introduced, focusing on the characteristics that make them scientifically interesting, such as the message-passing approach underlying PANDA or the possibility of inferring sample-specific GRNs using LIONESS. A detailed explanation of the

mathematical tools that ensure the biological significance of the inferred networks is then presented, outlining all the steps involved in these two workflows.

Given the importance of the scalability issue for the study case of scRNA-seq data in the applicability of GRN inference algorithms, it is necessary to evaluate the performance in response to variations of biologically and computationally significant parameters, such as the cell count, the hardware setup, and the sparsity of the regulation matrices. The results obtained will be evaluated based on the execution times associated with each part of the program, the memory occupied, the actual scalability, and the impact of the already implemented parallelization.

Once the performance analysis of these methods is completed, a code analysis, aimed at parallelizing the processes identified as time-limiting in the previous study, is presented. This implementation takes advantage of the capabilities of the parallel computing packages exposed in Chapter 1, assigning the identified processes to software that represents the state-of-the-art computational fields. Finally, this study presents an evaluation of the improvements that such modifications have on the temporal performance of the two algorithms, comparing, using the same input dataset size and hardware structure, the scalability of this optimized version with the previously analyzed implementation of PANDA.

The presentation of the results obtained through parallelization, along with the limitations characterizing the version proposed in this study and the possible future developments, represent the final part of this work. This study lays the groundwork for future improvements to these biologically and computationally interesting algorithms to enhance their scalability and applicability in significant cases of multi-omics single-cell data studies.

2. GPU Computing Technologies

Due to the increasing amount of biological data obtainable through single-cell sequencing techniques and the decreasing price of next-generation sequencing (NGS), the ability to extract and interpret relevant biological information has become a computational challenge, with computing power being a major limitation. This problem can be addressed by utilizing GPU computing technologies developed in the last decade.

In the following paragraphs, some of the main parallel computing applications created for omics data analysis will be presented.

2.1 Nvidia Parabricks

Parabricks is a suite of high-performance GPU computing and deep learning algorithms for NGS data processing [32,33]. It represents the main Nvidia product for designing a parallel bioinformatics pipeline involving all the secondary genomics analysis steps. This workflow includes, for instance, the alignment, the variant calling phase but also pre-processing, and the quality check process. Using its greater computational power, Parabricks can perform these fundamental genomics steps achieving previously unattainable performances in terms of throughput time and cost savings.

The next paragraphs will present (i) the main characteristics of this suite, (ii) the algorithms composing Parabricks, (iii) its hardware, software, and system requirements, (iv) the performances achieved in terms of execution time, memory memory usage and cost saving achieved.

2.1.1 Main Features

The main features of Parabricks are:

- *GPU acceleration*: The use of GPU instead of clusters of central processing units is the most relevant and interesting characteristic of Parabricks, it allows the execution of several classical biological processes in a fraction of the time and costs needed by a sequential corresponding framework, hiding the complexities related to the design of a parallel data architecture;
- *Correspondence with the well-known sequential algorithms*: Parabricks doesn't aim to revolutionize the reasoning at the basis of the most important genomics procedures, it provides the possibility to compute the same methodologies using a parallelized version of them;

- *Modularity, flexibility, and compatibility*: given Parabrick's structure: a suite of algorithms divided into sets based on the function, each process can be inserted in a bioinformatics workflow not necessarily using in all the steps the same type of processing units. Indeed, the input and output formats required by the GPU algorithms stay equal to the ones used in the sequential approach, guaranteeing compatibility between CPU and GPU-based processes. These characteristics provide flexibility to this product;
- *Working on Short and Long Reads*: the algorithms provided by Parabricks can handle the short reads produced by second-generation sequencers, such as Element, Illumina, MGI, Singular, Thermo Fisher e Ultima, but also the long reads sequenced by Oxford Nanopore and PacBio. This feature expands the possibility of using Parabricks for different purposes: assembling genomes from various samples and analysing the DNA sequences efficiently, saving costs;
- *Availability as Docker image:* Parabricks is deployed using a Docker image (and it can be imported as a singularity image if necessary);
- *Running from the Command Line:* each program that composes this product can be executed from the command line specifying input file, output directories, and the optional flags.

2.1.2 Software Overview

The software included in Parabricks can execute read alignment, processing and quality control (QC), and variant calling [34]. All these procedures use the potentialities of the GPU computation, guaranteeing the quality standard achieved by the well-known sequential version and compatibility with them.

The following table (Table 1) is reported: the main 6 categories in which Parabricks can be divided (already cited), the names of the algorithms composing each category, and a brief description of their computation. The last category is "Pipeline": it includes software obtained by the concatenation of several individual algorithms, and its execution can be done using using a single command.

Category	Algorithm	Description
Alignment	fq2bam (bwa-mem)	Run bwa-mem, co-ordinate sorting, marking duplicates, and Base Quality Score Recalibration
	fq2bam_meth	Run bwa-meth compatible alignment, co-ordinate sorting, marking duplicates, and Base Quality Score Recalibration
	fq2bamfast (bwa-mem)	Run newly optimized version of bwa-mem, co-ordinate sorting, marking duplicates, and Base Quality Score Replication
	rna_fq2bam (STAR)	Run RNA-seq data: starting from a FASTQ file, it performs the alignment with STAR algorithm and produces as output a BAM file.
	MiniMap2	Align long read sequences against a large reference database to convert FASTQ to BAM/CRAM
Preprocessing	applybqsr	Apply BQSR report to a BAM file and generate a new BAM file
	bam2fq	Convert a BAM file to FASTAQ
	bqsr	Collect BQSR report on a BAM file
	bamsort	Sort a BAM file
	markdup	Identifies duplicate reads
Variant Calling	deepvariant	Run GPU-DeepVariant for calling germline variants
	deepsomatic	Run GPU-DeepSomatic for calling somatic variants
	haplotypecaller	Run GPU-HaplotypeCaller for calling germline variants
	mutectcaller	Run GPU-Mutect2 for tumor-normal analysis
	starfusion	Identify candidate fusion transcripts supported by Illumina reads using the STAR-Fusion algorithm.

Quality Check	bammetrics	Collects WGS Metrics on a BAM file
	collectmultiple metrics	Collect multiple classes of metrics on a BAM file
GVCF Processing	indexgvcf	Index a GVCF file
	dbsnp	Annotate variants based on a dbsnp
	genotypegvfc	Convert a GVCF to VCF
	prepon	Build an index for PON file, which is the prerequisite to performing mutect pon
	postpon	Generate the final VCF output of doing mutect pon
Pipelines	deepvariant_germline	Run the germline pipeline from FASTQ to VCF using a deep neural network analysis
	germline (GATK germline pipeline)	Run the germline pipeline form FASTQ to VCF
	pacbio_germline	Run the germline pipeline from FASTQ to VCF by aligning long read sequences with minimap2 and using a deep neural network analysis
	somatic (Somatic Variant Caller)	Run the somatic pipeline from FASTQ to VCF

Table 1: Parabricks algorithms' categories and descriptions, from Nvidia Documentation [34]

2.1.3 Requirements

To execute all the algorithms included in the suite Parabricks these are the requirements, taken from the Nvidia documentation [32], to be satisfied:

Hardware Requirements

Any NVIDIA GPU that supports CUDA architecture 70, 75, 80, 86, 89 or 90 and has at least 16GB of GPU RAM. NVIDIA Parabricks has been tested on the following NVIDIA GPUs:

- V100
- T4
- A10, A30, A40, A100, A6000
- L4, L40
- H100, H200

- Grace Hopper Superchip

The fq2bam tool requires at least 24 GB of GPU memory by default; the --low-memory option will reduce this to 16 GB of GPU memory at the cost of slower processing. All other tools require at least 16 GB of GPU memory per GPU.

System Requirements:

- A 2 GPU system should have at least 100GB CPU RAM and at least 24 CPU threads.
- A 4 GPU system should have at least 196GB CPU RAM and at least 32 CPU threads.
- A 8 GPU system should have at least 392GB CPU RAM and at least 48 CPU threads.

Software Requirements

The following are software requirements for running Parabricks:

- An NVIDIA driver with version 525.60.13 or greater.
- Any Linux Operating System that supports Nvidia-docker2 Docker version 20.10 (or higher)

2.1.4 Performances

In this paragraph, the Parabricks' performances will be presented considering the time and cost savings obtained by studies executed on different settings of processor unities. The category of algorithms chosen as a reference for the comparison between CPU and GPU performance is the 'Pipeline' set because each element in it represents a complete secondary analysis going from a FASTQ file and producing as output a VCF file. In particular, the results presented in the following section are provided by the execution of 'germline' (cited in Table 1), composed of BWA-MEM, bamsort, markdup, applybqsr, and a Variant Caller algorithm (DeepVariant or HaplotypeCaller). This choice is justified by the will to show clearly the impact of a parallel approach on entire workflows, therefore a specific analysis of the savings produced by every single step included in the germline set of algorithms is not provided and is beyond our interests.

The machine settings considered are 32 vCPU, 2 GPU, 4 GPU, and 8 GPU. The results in terms of times are presented in Figure 2.

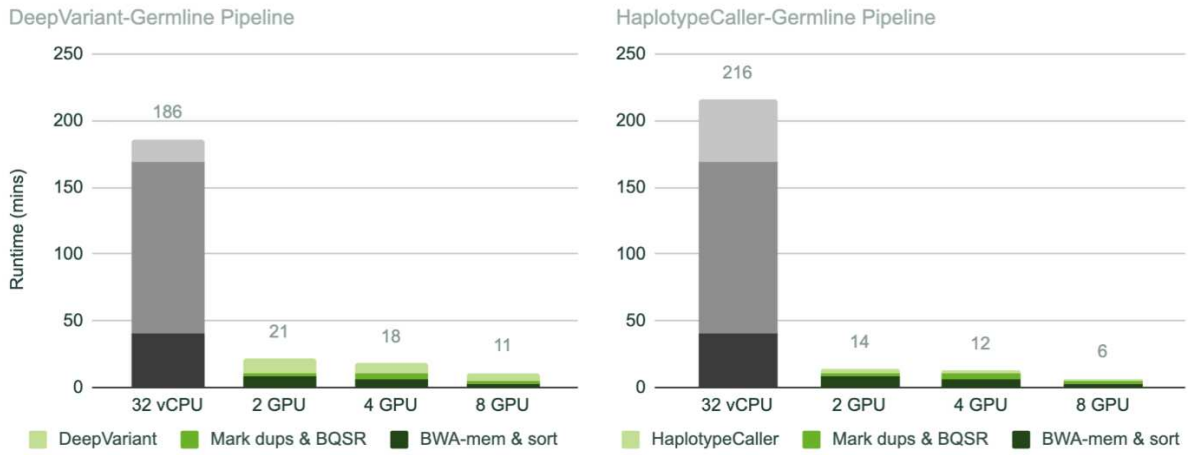


Figure 2: Runtimes of germline pipeline (with different Variant Caller algorithm) on CPU and GPU machines. Image taken from [35]

The time results are followed by the cost savings in terms of cost per exome analyzed, as reported in Figure 3.

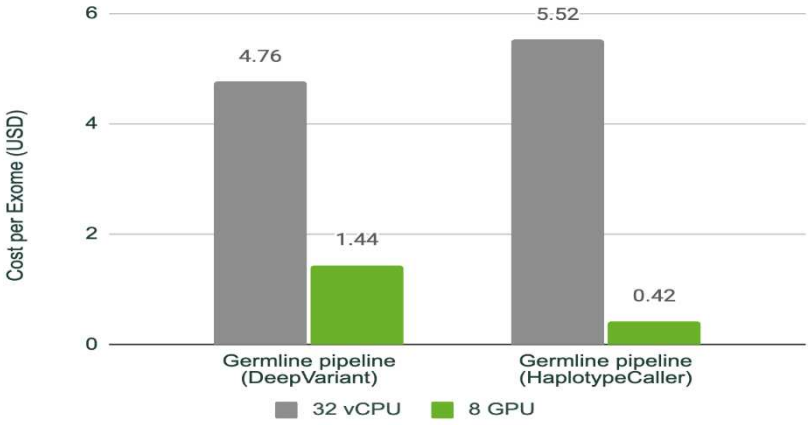


Figure 3: Cost per Exome (USD) using sequential (gray) and parallel (green) approaches. Image taken from [35]

2.2 Nvidia Rapids

Nvidia Rapids is a suite of libraries and APIs for the tertiary analysis of biological data, performing processes like the loading and pre-elaboration of the datasets but also the plot representations of the workflow results, including machine learning processes essential for tasks such as network analysis [36,37]. This product is based on Nvidia CUDA-X, a series of software that can optimize the performances of AI and High-Performance Computing applications using the opportunities given by a GPU setting. In particular, Rapids uses the CUDA primitives to increase the efficiency of low-level computation hiding the complexities of working with the GPU programs.

2.2.1 Main Features

The principal characteristics of Rapids are:

- *Open-source*: The source code of this product is freely available and editable, the main consequence is the community of programmers (not only from Nvidia Group) designing new features and reporting and fixing bugs;
- *General Purposes*: Rapids is created for a parallel approach to the main data analysis processes, not only for bioinformatics applications;
- *Sequential Correspondence*: Each package contained in Rapids Library is the parallel version of well-known sequential software and with these share the functions' names and in general the related APIs. This approach provides two significant results: (i) there are few different code lines between sequential and parallel versions of these procedures, (ii) the use of the Python language and interface hides the programming complexity related to the GPU computations;
- *Scientific Workflow*: When Rapids is used in a scientific process, it supports the entire data analysis workflow providing a set of software divided into these 5 macro-areas: data loading, pre-elaboration, machine learning algorithms, plot analysis and visualization;
- *Interoperability with frameworks*: Rapids libraries can easily be inserted in different scientific frameworks such as Apache Spark, Dask, Numba but also with deep learning frameworks like PyTorch, TensorFlow, and Apache MxNext.

2.2.2 Software Overview

As cited in the previous paragraphs, Rapids is divided into libraries representing each parallelization of a well-known sequential library and therefore regarding different aspects of data analysis, management, and visualization. The most bioinformatics relevant ones are presented in the Table 2.

Library	API	Language	Tasks
cuDF	PANDAs	Python/C++	Dataset Management: loading, joining, aggregating, filtering, and otherwise manipulating data
cuML	Scikit-learn	Python/C++	Machine learning algorithms designed for data science and analytical tasks (e.g. Clustering, Dimensionality Reduction, Linear Models for Regression or Classification)
cuPlot	NetworkX	Python/C++	Plot analysis algorithms (e.g. Centrality, Community, Link Analysis and Prediction, Sampling, Traversal, Structure)
cuXFilter	DashBoard	Python	Data Visualization and filtering
cuPy	Numpy	Python	Array Operations

Table 2: Rapids libraries that can be used in a genomic pipeline

It's important to clarify that cuPy library is not included in the Rapids suite, despite the shared features like being the parallelization of a vastly used product like Numpy. Nvidia supports the development of cuPy but doesn't hold the property on it.

2.3 Rapids-singlecell

Rapids-singlecell is a GPU-accelerated tool designed for scRNA-seq analysis using Rapids libraries and cuPy as a starting point to obtain a parallel substitute to the state-of-art sequential single-cell analysis methods [38]. In particular, the ecosystem containing Rapids-singlecell isn't a general-purpose suite, like in the previous cases, is a specific bioinformatics set of tools: scverse. Rapids-singlecell emulates and converts in a set of GPU-accelerated processes the most relevant tools contained in the programming ecosystem scverse, guaranteeing compatibility.

2.3.1 scverse and anndata

In the next paragraphs, scverse and anndata libraries will be briefly presented to clarify how some of the software and the main data formats of the sequential processes can be transposed in a parallel approach [39,40].

The core of scverse project is based on the design of omics data analysis tools, able to describe the main properties of spatial omics, regulatory genomics, trajectory inference, and visualization. These processes are divided into several packages, the ones considered foundational are scanpy (for analyzing single-cell gene expression in anndata format), muon

(framework for multimodal omics analysis), scvi-tools (machine learning algorithms using PyTorch and anndata), scirpy (for the analysis of T-cell receptors or B-cell receptors from scRNA sequencing data) and squidpy (for the visualization of spatial molecular data).

Anndata is a Python library introducing the management of a widely used data structure: the annotated data object. Let's describe all the components, represented in Figure 4, of this object:

- *X:* Sparse or dense matrix containing the gene expression data, with rows representing genes and columns representing cells. Using the function 'layer' it's possible to create modified versions of the raw count matrix without losing it and guaranteeing the same relationship between the new generated version and the annotation matrices had by the initial matrix.
- *obs* and *var*: PANDAs dataframes for various annotations (var refers to gene annotations and obs for cell annotations) about the data, such as gene names and IDs, cell barcodes and metadata, and cluster assignments.
- *obsm, varm* and *uns*: Results of various analysis steps, such as dimensionality reduction or clustering, are stored: in obsm if they are structured and associated with cells, in varm if they are structured and associated with genes, or in uns if the results are dictionaries in the form of unstructured data.
- *obsp* and *varp*: dataframes containing pair information about genes and cells observed, such as distances, similarity score or any other measure of cell comparison in the case of obsp or measures of gene comparison in the case of varp.

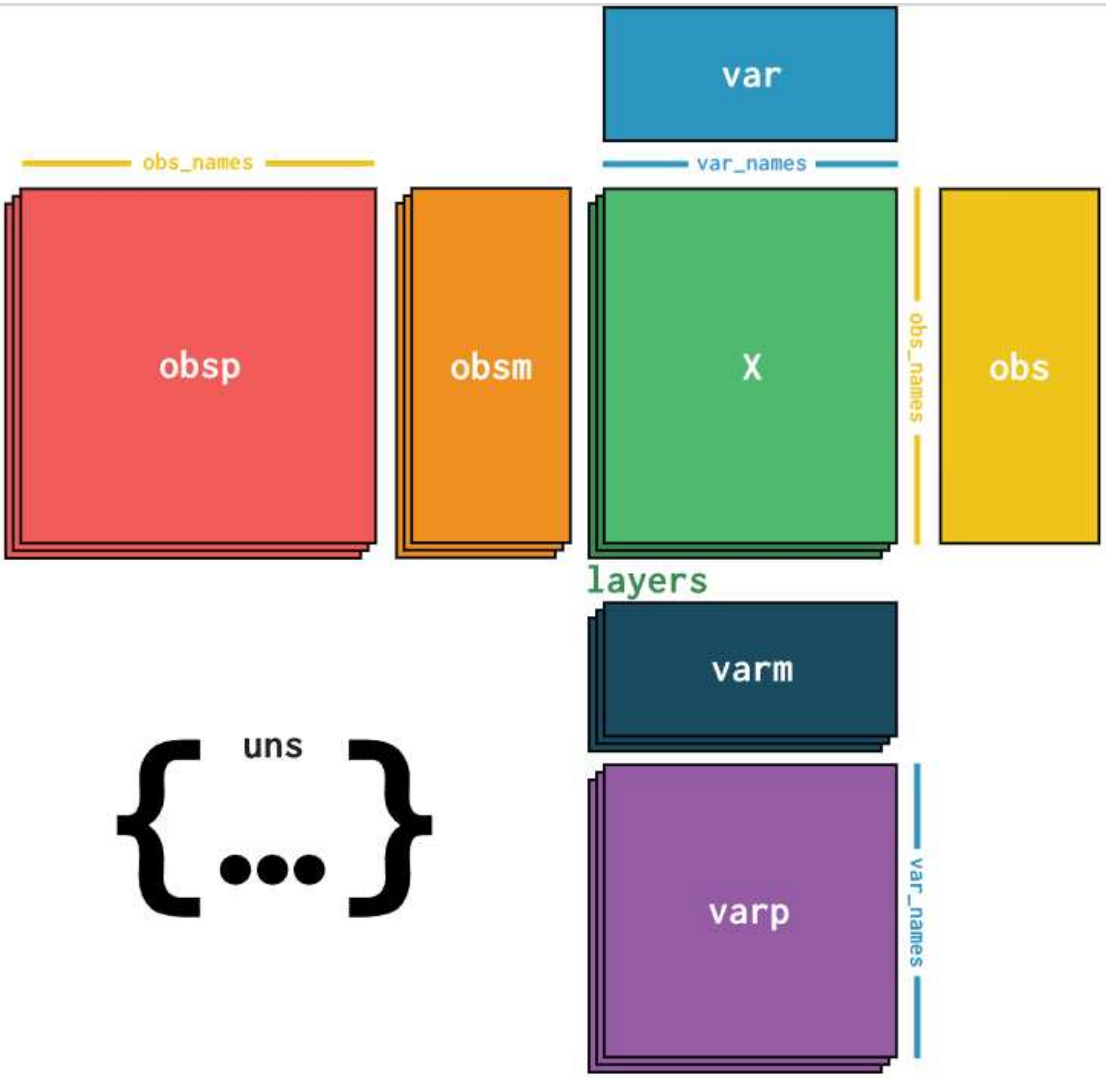


Figure 4: Anndata structure representation. Image taken from [40]

2.3.2 Rapids-singlecell Features and Performances

Rapids-singlecell considers a parallel version of anndata, cunndata represented in Figure 5, as the main data format on which all the packages can execute their functions. The structure of this parallel version replicates all the anndata' features: the presence of a datamatrix X divided into several layers and described by several annotations matrices related to genes and cells, the opportunity of using a dictionary for the unstructured information, the correlated dimensions and indexing of each described cunndata's matrix. The main difference between these two formats is that cunndata matrix X and its layers are stored on the GPU as CuPy sparse matrices, instead of being stored on the local memory; this change provides faster and more efficient computations on the count matrix. All the annotation matrices are stored in the

host memory, except for the pair-wise information matrices, obsp and varp, which are not part of the data structure and can be consulted only using external references.

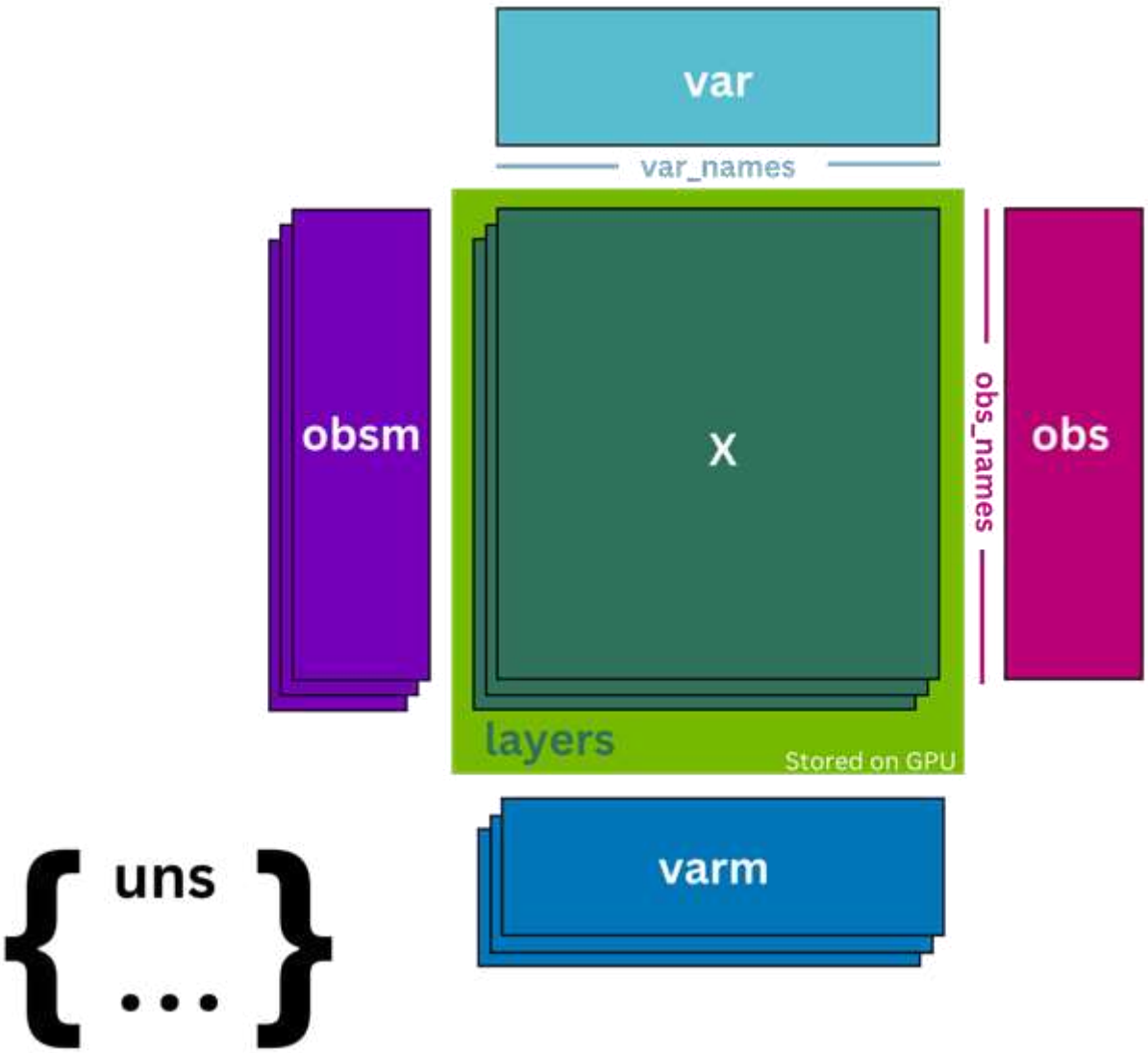


Figure 5: Representation of cunndata structure. Image taken from [41]

Given this parallel data structure, the Rapids-singlecell team recreated some of the most biologically relevant functions from scverse. For instance, from Scanpy, they implemented UMAP, nearest neighbors on the GPU, and accelerated plot-based clustering using Leiden and Louvain. From scvi-tools, they included functions for embedding single-cell data, and they are now adding Squidpy functions to this suite of libraries. This process of adapting well-known sequential algorithms to the new data architecture has resulted in performance improvements with speed factors ranging from 4x to over 60x compared to the sequential algorithm times on a dataset of 500,000 cells, as reported in Table 3.

Function	CPU	GPU (A100)	GPU (3090)	Speedup
Preprocessing	305 s	28 s	169 s	10x
PCA	86 s	3.7 s	35 s	23x
Neighbors	74 s	17.1 s	18.3 s	4.3x
UMAP	281 s	6.7 s	7.6 s	60x
Louvain	283 s	4.5 s	5.7 s	62x
Logistic Regression	452 s	33 s	63 s	13x

Table 3: Performances of CPU and GPU-based algorithm on a dataset of 500.000 cells [41]

3. Gene Regulatory Network Inference

One of the primary objectives within the bioinformatics community is to attain a profound understanding of the intricate biological processes that govern the phenotype of a specific cell or sample. A single entity, such as a gene or a pathway, cannot sufficiently elucidate the phenotype exhibited by a cell; a more holistic perspective is required to comprehend the underlying mechanisms. Consequently, constructing a network that represents the gene expression regulating all cellular mechanisms has become a priority in systems biology. Indeed, the inference and analysis of a representative plot are essential tools, allowing biologically meaningful structures to be distilled into significant network patterns and, more broadly, features.

This type of plot is known as a GRN and aims to capture these cellular processes by leveraging the influence that each gene exerts on others. Specifically, this structure is modeled using network entities such as "nodes" representing the genes, "edges" representing the inferred interactions, and "weights" representing the type of interaction between two genes, such as activation, repression, or absence of regulation.

In this chapter, we present a GRN inference algorithm called "PANDA" in detail, considering the importance of a multi-omics approach in this context. We will discuss the input required, provide a step-by-step analysis of PANDA’s computation, and describe the output produced—a network that globally represents the features of all the samples considered. The second algorithm discussed in this chapter, "LIONESS", addresses the limitation of having a GRN that captures only shared processes. Instead, LIONESS computes several sample-specific gene regulatory plots, following the same detailed presentation structure. Finally, we provide some applications and considerations regarding these two algorithms. Specifically, by using increasing size benchmarks to evaluate both algorithms' performance, we can highlight their limitations and propose potential solutions.

3.1 PANDA Algorithm

The GRN models able to integrate different data types have reached significant results, representing efficiently the complex biological systems considered as references. The accuracy of these models stems from the successful amalgamation of information through the integration process, providing a more profound understanding of the complex molecular processes underlying biological systems than models using a single data type alone. One of the most prominent strategies for integrating multiple data types is the Message-Passing approach. This approach is utilized in various gene regulation model procedures, such as

estimating signaling pathways or parameters related to physical network models. In this method, the data types are treated as entities that send and receive "messages" containing information aimed at developing a cumulative data structure.

PANDA (Passing Attributes between Networks for Data Assimilation) is a GRN inference algorithm within the collection of open-source methods known as NetworkZoo [42]. In this collection, several algorithms elaborate different aspects of the GRN inference process, such as the normalization of the RNA-seq data or the analysis of the community properties of the GRN produced. NetworkZoo represents a significant set of tools for the inference and analysis of GRN elaborated considering as input multi-omic data, and it is adaptable to several programming languages, such as Python, R, MATLAB, and C. This study will consider only the two algorithms presented in this chapter, PANDA and LIONESS, and the Python implementation of this suite, called netZooPy.

By utilizing multiple sources of “omics” data, as specified in the Input section, PANDA generates a Transcription Factor-by-Gene matrix that can be readily converted into a biologically accurate GRN. This software implements the concept of merging information from diverse data sources through an iterative message-passing approach.

Compared to previous methods, the primary innovation introduced by the PANDA algorithm is the iterative update of all data sources. Once pieces of information are extracted from these sources, they are considered as messages that are processed and shared with all other nodes. This iterative refinement, represented in Figure 6, enhances the representation of biological pathways within the distinct data structures.

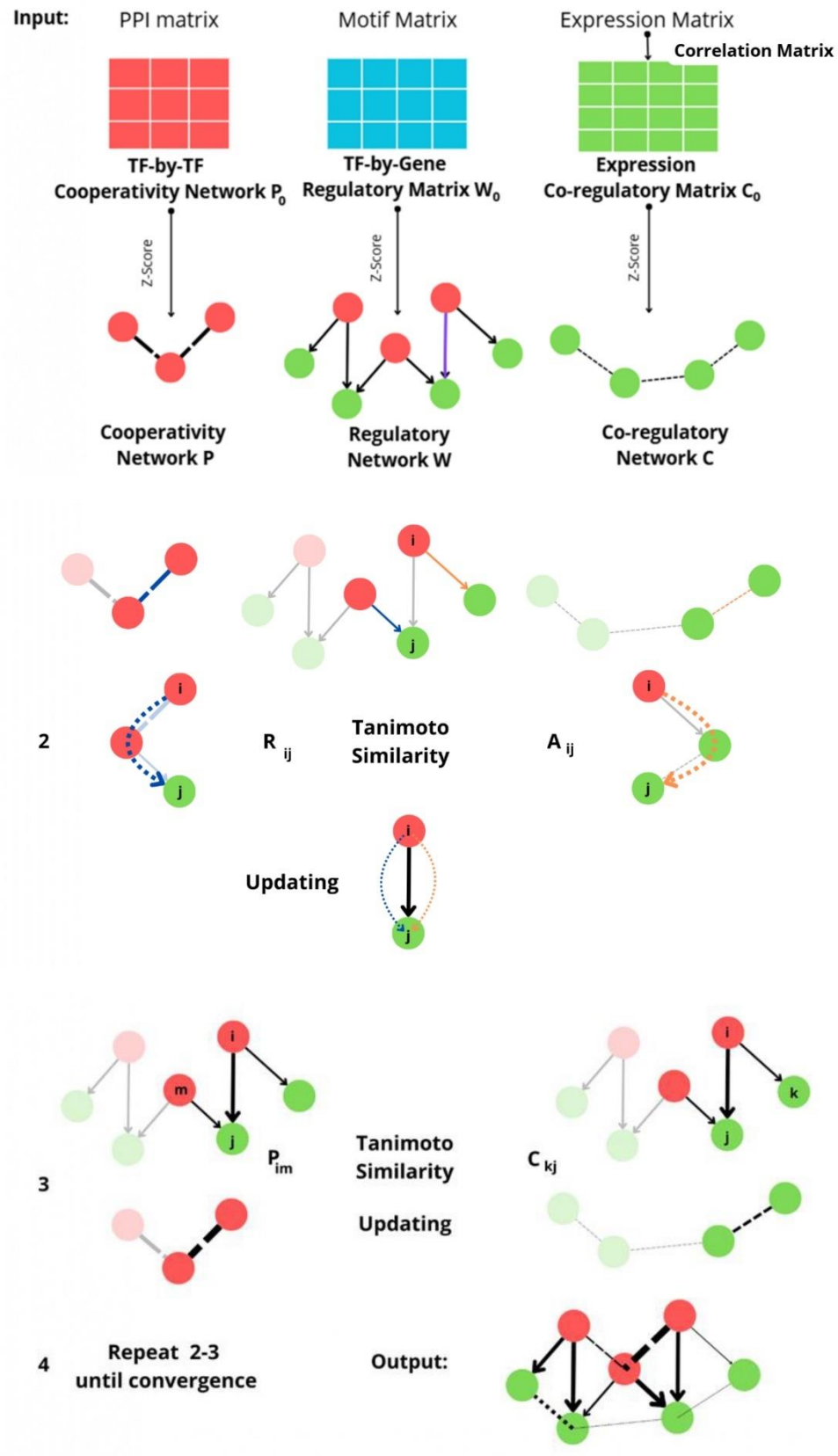


Figure 6: PANDA's workflow: from the loading and initialization of the input matrices (1) to the Responsibility and Availability Estimation (2), followed by the update of the two networks P and C (3). This process is repeated until convergence (4) and produces a GRN.

PANDA takes as input three matrices, each of which can be replaced with a 'None' value if the user chooses not to provide one or more of them. Each matrix collects information about one of the relationships among the considered entities: Transcription Factors (TFs) and Regulated Genes.

- **Transcription Factors-by-Gene Regulatory Matrix (W_0)**: This matrix records known relationships between TFs and regulated genes. These relationships are identified by comparing the promoter regions of the genes and the transcription factor binding sites (TFBS). If the TFBS are within a specified size window, the TF is considered to regulate the gene expression.
- **TF-by-TF Cooperativity Network (P_0):** Based on a protein-protein interaction (PPI) matrix, this table represents the relationships between pairs of proteins that can form multi-protein complexes to regulate specific genes. This matrix can be generated from data collected through in vitro experiments, text mining, or computational inference and is used in the algorithm as the primary source of information regarding potential indirect gene regulation by complex TFs.
- **Expression Co-regulatory Matrix (C_0):** Derived from an expression matrix, this matrix computes the pairwise Pearson Correlation Coefficient (PCC) for each pair of genes. The final result is a gene-by-gene matrix that captures the correlation between genes. This correlation is useful in the PANDA computation because genes regulated by the same TFs are expected to exhibit correlated gene expression patterns.

PANDA's primary method for inferring the GRN involves the iterative update of the input matrices to maximize the agreement among them. This process aims to compute biologically accurate and consistent predictions regarding the influence of each gene on the expression of others. In this sense, two initial formal definitions are needed to define rigorously the concept of agreement between two vectors representing information about genes, and all the modeled relations among genes.

The concept of agreement between data of different networks is defined and quantified in PANDA using the Tanimoto Similarity:

$$T_z(\bar{x},\bar{y}) = \frac{\bar{x}\bar{y}}{\sqrt{\left|\left|\bar{x}\right|\right|^2 + \left|\left|\bar{y}\right|\right|^2 + |\bar{x}\bar{y}|}} = \frac{\sum_i x_i\, y_i}{\sqrt{\sum_i {x_i}^2 + \sum_i {y_i}^2 + |\sum_i x_i\, y_i|}}$$

This formula expresses the similarity between two z-score normalized vectors () that represent two sets of network edge weights. When applied to the vectors associated with edges in two

different networks, a T_z value near 1 indicates a strong positive agreement between them. Conversely, a T_z value around 0 suggests no significant similarity, and a negative T_z value, which can range down to -1, indicates an inverse relationship between the two networks.

To elaborate:

- Positive Agreement ($Tz \approx 1$): This suggests that the two networks share a similar pattern of edge weights, indicating that the regulatory relationships between genes are consistent across both networks.
- No Agreement ($Tz \approx 0$): This indicates that there is little to no similarity in the edge weights between the two networks, suggesting that the regulatory relationships differ significantly.
- Negative Agreement ($Tz \approx -1$): This signifies that the edge weights in one network are inversely related to those in the other, implying that regulatory relationships are oppositely oriented between the two networks.

This measure of similarity is crucial in the iterative update process of PANDA, ensuring that the final inferred GRN maximizes the agreement among the integrated data sources, thereby producing a more accurate and biologically relevant network

The second formal set of definitions, as clarified by Figure 7, is about the nodes and edges types: PANDA configures two types of nodes in the considered networks: “effector” and “target”. The elements in the first set exploit an influence on the expression of the genes considered in the second group, modeling the biological relation between the Transcription Factors and the Genes regulated by them. Given these two different entities, the expression of the genes is governed by the influence of the effectors in two different ways: “directly”, a set of TFs is responsible for the regulation of these affected nodes in a relation defined as “routes of affection”; “indirectly”, a TF can create a protein complex to regulate genes not directly influenced by it, in this case, the two TFs perform a “cooperative effect” on the considered

gene, or, given two genes with similar expression is possible that they are targets of the same group of effectors, exploiting a "co-affection" relationship between them.

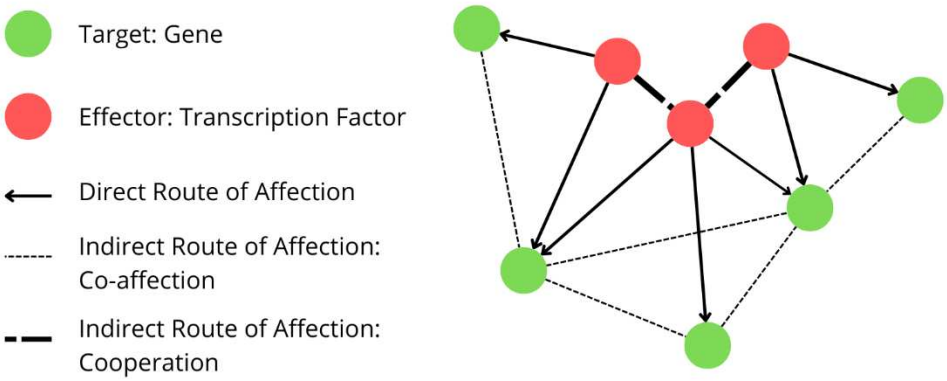


Figure 7: This figure represents the two types of nodes (effectors and targets) and the three types of edges (direct and indirect route of affection) of a GRN considered by PANDA.

Let's now explain PANDA's workflow step-by-step in detail, as represented in Figure 6.

1st Step: Initialization of the Networks

The first step is the initialization of three different networks from the input matrices: The values collected in these matrices, P_0, W_0, and C_0, are standardized using a Z-score computation across both rows and columns, guaranteeing the same scale for all the tables. These are from now on considered networks describing the regulation between Transcription Factors in the Cooperative Network (P), between Genes in the Co-regulatory Network (C), and between each TF and each Gene in the Regulatory Network (W).

2nd Step: Responsibility and Availability

While the values contained in W represent the direct regulation operated by the effectors to the targets, the undirected influences, described previously and represented plotically in Figure 8, need two additional parameters to be taken into account in the global regulation effect. Therefore, let's consider the "cooperative effect" that a Transcription Factor i can have, through the formation of a protein complex with other TFs, on a target gene j: this is not reported in any of the input networks but can be modeled considering a compositional effect

and quantified in the "Responsibility" value R_{ij}. To do that, a similarity score between all the edges linking the TF i with all the other TFs, $P_{i.}$, and the edges between all the TFs and the gene j, $W_{.j}$, is computed using the Tanimoto relation:

$$R_{ij} = T_z(P_{i.}, W_{.j}) = \frac{\sum_m P_{im} W_{mj}}{\sqrt{\sum_m (P_{im})^2 + \sum_m (W_{mj})^2 + \left|\sum_m P_{im} W_{mj}\right|}}$$

The second type of undirected regulation has been previously called "co-affection" and is present when the same transcription factor regulates two different genes. When this happens, it's notable that the regulation of these two genes assumes comparable features, underling the relation given by being the targets of the same effector. Using the Tanimoto similarity, is possible to measure, through the edges' weights, how much this effect influences the regulation of a gene i by the action of a TF j. In this sense, let's consider the genes targeted by j ($W_{j.}$) and all the genes co-regulated with gene i ($C_{.i}$): when a generic gene k is targeted by j and co-regulated with i the scalar product $W_{jk}C_{ki}$ is different from 0 and contributes to generate a more significant availability value A_{ij}.

$$A_{ij} = T_z(W_{j.}, C_{.i}) = \frac{\sum_k W_{jk} C_{ki}}{\sqrt{\sum_k (W_{jk})^2 + \sum_k (C_{ki})^2 + \left|\sum_k W_{jk} C_{ki}\right|}}$$

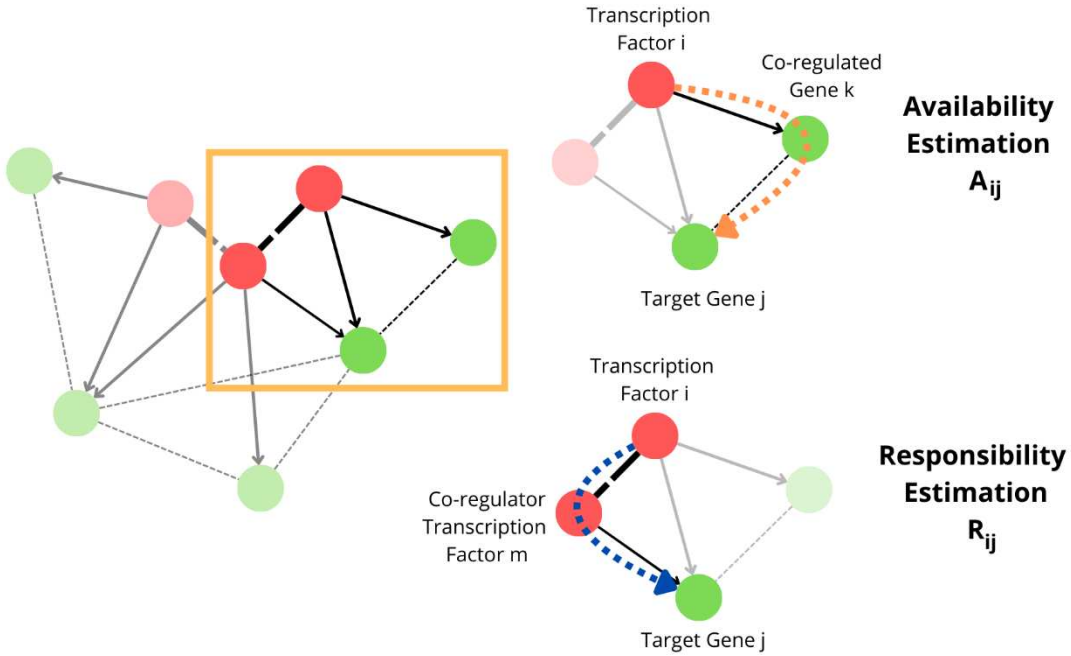


Figure 8: Considering a TF i and a target gene j, two measures, Responsibility and Availability, are computed to estimate the undirected regulation of j by i.

3° Step: Estimation and Updating

When these two quantities related to phenomena are computed, the estimation $\widetilde{W}_{ij}$ of these undirected regulation effects is obtained as the mean value between A_{ij} and R_{ij}:

$$\widetilde{W}_{ij} = \frac{A_{ij} + R_{ij}}{2}$$

This value will be added to the previous regulation value $W_{ij}^{(t)}$ in proportion to a learning rate α (in]0,1[), to obtain an updated version for the next iteration (t+1):

$$W_{ij}^{(t+1)} = (1-\alpha)W_{ij}^{(t)} + \alpha\widetilde{W}_{ij}^{(t)}$$

The message-passing approach, finalized to find agreement among the 3 different data types, in this algorithm is expressed in the updating of the regulatory network W and then also of cooperativity and co-regulatory ones. Therefore, once the new version of W is estimated based on the computation of responsibility and availability, it's time to estimate the new matrices P and C.

Matrix P is updated considering that the values representing the regulation executed by each transcription factor can be different from an iteration to the following, therefore is necessary to consider how these new features can generate new information about TF co-regulation. This is done considering the similarities between the edges of two different effectors, i and j:

$$P_{ij} = T_z(W_{j.}, W_{i.}) = \frac{\sum_k W_{jk}\, W_{ik}}{\sqrt{\sum_k (W_{jk})^2 + \sum_k (W_{ik})^2 + \left|\sum_k W_{jk}\, W_{ik}\right|}}$$

And then updating the value associated with them in the next iteration, namely $P_{ij}^{(t+1)}$:

$$P_{ij}^{(t+1)} = (1-\alpha)P_{ij}^{(t)} + \alpha\tilde{P}_{ij}^{(t)}$$

Similarly, knowing how each relation between TF and gene changes by a single iteration, the co-regulated genes can present different features that can be quantified by estimating and updating the similarity between the regulation of each pair of genes i and j:

$$C_{ij} = T_z(W_{.i}, W_{.j}) = \frac{\sum_k W_{ki}\, W_{kj}}{\sqrt{\sum_k (W_{kj})^2 + \sum_k (W_{ki})^2 + \left|\sum_k W_{kj}\, W_{ki}\right|}}$$

$$C_{ij}^{(t+1)} = (1-\alpha)C_{ij}^{(t)} + \alpha\tilde{C}_{ij}^{(t)}$$

When all the values in all the three matrices are updated, the process is iteratively repeated until convergence, which is guaranteed, is achieved.

The output returned is a matrix where each row identifies a transcription factor-gene pair, followed by two values: the motif value, a binary number representing the presence of this association in the original motif matrix W_0; and the weight of this effector-target's edge.

3.2 LIONESS Algorithm

The network inferred by PANDA's computation is a plot representing the biological processes involved in the gene expression of a given group of samples. This data structure is valuable because it can transform a biological question (such as the differences between two groups of gene expressions) into a mathematical analysis (identifying patterns and pathways differentially present in two networks), thereby facilitating the integration and management of data from multiple sources.

This approach ensures accurate and biologically meaningful results when the differentially expressed pathways are common within the samples of the same group and thus present in the consensus. However, processes that cause phenotype changes in a single sample, or in a subset of samples, are not easily detectable by analyzing the aggregate network of samples. This limitation is significant in several contexts. For instance, in single-cell differential expression estimation, the goal is to identify the unique active gene regulatory processes that define the specific phenotype and functions of individual cells within a tissue. Similarly, recognizing the distinct regulatory mechanisms in an individual within a group exhibiting slight phenotypic differences can be challenging.

To overcome this limitation, NetworkZoo developed LIONESS (Linear Interpolation to Obtain Network Estimates for Single Samples), an algorithm designed to estimate sample-specific plots from an aggregate network using a linear interpolation approach and represented in Figure 9. LIONESS is intended to be applied after PANDA's computation, thereby enhancing the algorithm's capabilities. This creates a suite of software tools that can infer GRNs from multi-omics data for both groups of samples and for subsets of them, particularly enabling the accurate estimation of sample-specific GRNs.

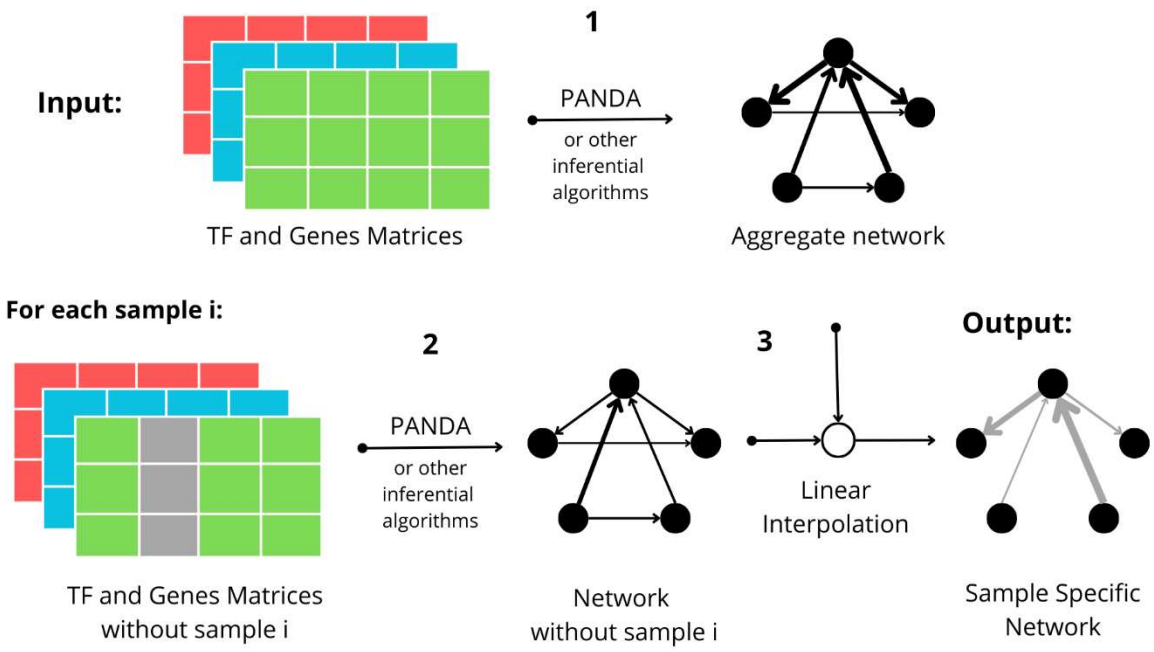


Figure 9: LIONESS Workflow, given the same set of input matrices described for PANDA, LIONESS produces N sample specific GRN.

Input

The inputs LIONESS requires are (i) the gene expression matrix, in which each column represents a sample and each row a gene; (ii) the protein-protein interaction (PPI) matrix, defining the interaction among TF; (iii) the Transcription Factors-by-Gene Regulatory Matrix presenting the relationships between TFs and regulated genes. Note also that the aggregate regulatory network produced as output by inference algorithm as PANDA can be passed as input to LIONESS, if this happens the first step of its workflow is skipped because already satisfied.

Mathematical explanation

The main assumption on which the entire algorithm works is that the aggregate network taken as input is obtained by a linear combination of the N samples' contributions. In particular, if we consider a value e associated with an edge going from gene i to gene j in the aggregate network α ($e_{ij}^{(\alpha)}$), it can be modeled as the sum of the same edges' values ($e_{ij}^{(s)}$) belonging to the N sample networks weighted by a scalar factor $w_s^{(\alpha)}$ characterizing the impact that the generic s sample network has on the global network α.

$$e_{ij}^{(\alpha)} = \sum_{s=1}^{N} w_s^{(\alpha)} e_{ij}^{(s)}$$

The sum of all the sample networks' weights is set equal to 1, as specified in the next equation:

$$\sum_{s=1}^{N} w_s^{(\alpha)} = 1$$

The LIONESS's final goal is to obtain N sample-specific networks, which can be declined in the computation of the weights associated with each single edge composing the network. The weight of the edge of a generic sample network q can be calculated using the values associated with the plot obtained by considering all the samples except q, as explained in the following procedure.

The edges in the network α-q can be defined similarly to before, with a different distribution of weights for the other networks.

$$e_{ij}^{(\alpha-q)} = \sum_{s\neq q}^{N} w_s^{(\alpha-q)} e_{ij}^{(s)}$$

Where: $\sum_{s\neq q}^{N} w_s^{(\alpha-q)} = 1$

From these two definitions, let's compute the generic weight edge $e_{ij}^{(q)}$

In particular, let's subtract these equations and rearrange them:

$$e_{ij}^{(\alpha)} - e_{ij}^{(\alpha-q)} = \sum_{s=1}^{N} w_s^{(\alpha)} e_{ij}^{(s)} - \sum_{s\neq q}^{N} w_s^{(\alpha-q)} e_{ij}^{(s)}$$

$$= w_q^{(\alpha)} e_{ij}^{(q)} + \sum_{s\neq q}^{N} w_s^{(\alpha)} e_{ij}^{(s)} - \sum_{s\neq q}^{N} w_s^{(\alpha-q)} e_{ij}^{(s)}$$

$$= w_q^{(\alpha)} e_{ij}^{(q)} + \sum_{s\neq q}^{N} (w_s^{(\alpha)} - w_s^{(\alpha-q)}) e_{ij}^{(s)}$$

Finally, let's isolate $e_{ij}^{(q)}$:

$$e_{ij}^{(q)} = \frac{1}{w_q^{(\alpha)}} \left[e_{ij}^{(\alpha)} - e_{ij}^{(\alpha-q)} + \sum_{s\neq q}^{N} (w_s^{(\alpha-q)} - w_s^{(\alpha)}) e_{ij}^{(s)} \right]$$

Knowing that: $e_{ij}^{(\alpha-q)} = \sum_{s\neq q}^{N} w_s^{(\alpha-q)} e_{ij}^{(s)}$

$$e_{ij}^{(q)} = \frac{1}{w_q^{(\alpha)}}\left[e_{ij}^{(\alpha)} + \sum_{s\neq q}^{N} w_s^{(\alpha)} e_{ij}^{(s)}\right]$$

It's now necessary to clarify the values associated with the sample network contribution weights. We can assume the existence of a relation between $w_s^{(\alpha)}$ and $w_s^{(\alpha-q)}$ using a constant C: $w_s^{(\alpha)}=Cw_s^{(\alpha-q)}$, now it's possible to construct the following chain of equations:

$$1 = \sum_{s\neq q}^{N} w_s^{(\alpha-q)} = \sum_{s=1}^{N} w_s^{(\alpha)} = w_q^{(\alpha)} + \sum_{s\neq q}^{N} w_s^{(\alpha)} = w_q^{(\alpha)} + C\sum_{s\neq q}^{N} w_s^{(\alpha-q)} = w_q^{(\alpha)} + C$$

Finally, let's substitute these results in the previous formula and let's consider the weights associated with each sample network: these can be established by estimating the quality scores of the data for individual samples, or, in the absence of this information or similar measures, with the further assumption that in the aggregate network α each sample network influences it equally (therefore $w_q^{(\alpha)} = \frac{1}{N}$):

$$e_{ij}^{(q)} = \frac{1}{w_q^{(\alpha)}}\left[e_{ij}^{(\alpha)} - C\sum_{s\neq q}^{N} w_s^{(\alpha-q)} e_{ij}^{(s)}\right]$$

$$e_{ij}^{(q)} = \frac{1}{w_q^{(\alpha)}}\left[e_{ij}^{(\alpha)} - (1 - w_q^{(\alpha)})e_{ij}^{(\alpha-q)}\right]$$

$$e_{ij}^{(q)} = \frac{1}{w_q^{(\alpha)}}\left[e_{ij}^{(\alpha)} - e_{ij}^{(\alpha-q)}\right] + e_{ij}^{(\alpha-q)}$$

$$e_{ij}^{(q)} = N\left[e_{ij}^{(\alpha)} - e_{ij}^{(\alpha-q)}\right] + e_{ij}^{(\alpha-q)}$$

The final result of this mathematical study is that each sample-specific network edge can be modeled using only two estimations: the weight of the same edge in the aggregate network and the weight of the same edge in the network obtained using all the samples except the considered one. This result can now be applied to the algorithm, representing the most crucial theoretical notion exploited by LIONESS, which doesn't depend on the inference method used to estimate the aggregate network.

Step-by-Step Algorithm and Output

LIONESS' workflow can be expressed as the sequence of the following steps:

1. Aggregate GRN Inference: The gene expression matrix and the other two optional tables (Transcription Factors-by-gene regulatory matrix and TF-by-TF cooperativity network) are given as arguments to the network inference algorithm, the result is the GRN representing the processes shared by all the samples; in this first phase other inference algorithms different than PANDA can be used;
2. Sample-specific Network inference loop: For each sample present in the considered gene expression, an execution of the inference algorithm PANDA is needed to compute the network composed by all the samples except for the one thought.
3. Linear Interpolation: The weights of the edges in the aggregate network and of the networks computed in the previous point are used to estimate the weights of the sample-specific plots.

The output of this workflow are N sample-specific networks represented, as in the case of the aggregate network, as a matrix in which each row defines a pair of gene j and transcription factor i and the value of the regulation that i exploits on j. These N matrices can be easily translated into GRN and used as primary elements to perform plot analysis, discovering biologically meaningful features using a mathematical approach.

4. PANDA Performance

This chapter describes the experiments conducted and the results obtained from the execution of the two previously introduced algorithms, PANDA and LIONESS, across various experimental conditions. The objective of this analysis is to evaluate which are the tasks optimizable in terms of execution time and memory required. Assessing and analyzing the performance of each algorithm becomes, in this sense, essential and several computational aspects can be inferred by varying:

- the hardware configuration, using CPU and GPU setups;
- the datasets used, ranging from 'toy' datasets provided by the netZooPy project to a dataset of dimensions comparable to realistic single-cell data;
- the intrinsic characteristics of the datasets, including variations in the frequency of protein complex formation and the number of genes regulated by the same TF.

The experiments are followed by a comprehensive analysis of results, discussing performance metrics, limitations, and encountered challenges.

It is important to note that version 0.8 of netZooPy was used in this analysis[43]. This version was selected because it is automatically installed via pip, whereas the latest version, 0.10, requires a specific installation procedure.

4.1 Hardware Setups

To compare the performance of sequential versus parallel approaches, various hardware configurations were selected to support all necessary experiments and effectively address the research questions. Specifically, the following hardware setups were employed:

- 1 Intel Xeon Gold 5118 CPU
- 1 Nvidia RTX 3090 GPU
- 1 Nvidia A40 GPU

In particular, the GPUs are used considering in all the following experiments one single core, and, when the functions contained by the tested programs cannot support a GPU computation a CPU is used to compute it.

4.2 Datasets

The primary objective of testing this GRN inference algorithm is to assess its scalability, using a set of datasets with an increasing cell count while maintaining also certain crucial features constant throughout the experiments. These criteria are essential due to the nature of

the scRNA-seq matrix analysis: usually, a high number of cells are required to characterize the status of a single patient, therefore when the research is aimed to discover specific patterns potentially shared by all the patients of the same cohort, comparisons among groups of patients become essential, obtaining an expression matrix with a number of cells in the order of hundreds of thousands or millions. The requirement to maintain specific features constant ensures comparability among the conducted experiments and at the same time, it's supported by biological constraints as the natural fixed upper bound of the number of genes; in this way, the only variable is cell count, minimizing the influence of other factors that could confound the analysis.

The starting point for the PANDA performance analysis is a dataset published by the netZooPy development team, here called 'Toy' dataset, intended for testing the software available in the netZooPy library. The dataset dimensions are detailed in Table 4. As it is designed specifically for testing purposes, this dataset lacks the properties of a realistic biological dataset, especially concerning the dimensions of the expression matrix, which are not comparable to those typically associated with single-cell data. This difference in purposes and dimensions prevents the possibility of obtaining significant results by comparison with the other dataset, limiting its usability in this analysis.

A public gene expression dataset was selected to analyze the time and memory performance with greater biological relevance, matching the characteristics of a real-world RNA sequencing matrix. Specifically, the dataset analyzed in Tirosh et al. (GEO ID: GSE72056), which has been used for annotation development and cell communication studies, captures gene expression at the cell level across diverse melanoma profiles, holding substantial biological significance due to the nature of the cells involved, such as those with drug-resistance properties observed in specific cell populations. The dataset dimensions (shown in Table 1 under the label '4k') differ substantially from those of the preliminary "toy" dataset but still fall short of typical single-cell data scales, consequentially the performance analysis regarding the GRN generation starting from this data cannot be generalized and the necessity of gene expression matrices with a higher number of cells remains. The biological characteristics represented in this dataset will be analyzed and abstracted to generate different simulated gene expression matrices with the dimensions decided a priori.

Due to this specific set of requirements, simulated datasets were generated with controlled, measurable characteristics to identify and quantify the contribution of each feature on the algorithm's performance in terms of time and memory usage. A dataset simulator was developed with the following assumptions:

- ***Sparsity of the Expression Matrix*:** the proportion of zero values in the expression matrix – a key characteristic of single-cell data - in the simulation is set to 80% and is randomly distributed across all the genes;
- ***Sequencing Depth:*** the total count of RNA sequences mapped to genes is held constant, equal to the average sequencing depth measured in the real dataset;
- ***Number of genes regulated by a single Transcription Factor:*** Based on biological knowledge, this parameter is randomly drawn from a uniform distribution, with each TF regulating between 200 and 1,000 genes;
- ***Number of Transcription Factors cooperating in the same Protein Complex:*** This parameter, also informed by biological insights, is a random value drawn from a uniform distribution, ranging from 5 to 20 TFs capable of forming a protein complex with a given TF;
- ***Distribution of the expression data:*** For RNA sequencing data, the Negative Binomial and Poisson distributions are common fits to approximate realistic gene expression values in the simulated dataset;
- ***Distribution of the regulation and coregulation data:*** the selection of which genes are regulated by which TFs and of which TFs can cooperate to create a protein complex has been left to a random choice by the simulator;

Certain choices, such as using a specific distribution or ignoring the specific pairing of genes and TFs, may not yield highly refined datasets. However, for the purposes of this study—assessing algorithm performance and usability—these simplifications are unlikely to significantly affect the quantitative aspects of computation, which are the primary focus of the results.

The assumptions outlined have produced dataset features that fall into two distinct categories: constants across all experiments—such as the Sparsity of the Expression Matrix, Sequencing Depth, and the distributions of expression, regulation, and co-regulation data, as determined by biological knowledge—and variable features, used to assess the performance of PANDA and LIONESS under different regulatory and co-regulatory conditions of the transcription factors.

These choices resulted in a set of matrices with the characteristics summarized in Table 4. The number of genes in the Gene Expression Matrix is maintained consistently from the real dataset (referred to as '4k' in the table and subsequent sections) to reflect a realistic gene count

for an expression matrix, independently of cell count. The number of cells, however, is scaled across datasets, serving as the primary variable in this analysis.

The Motif Data and Protein-Protein Interaction (PPI) Matrices comprise a total of four matrices (two for each type) simulated based on the regulatory and co-regulatory assumptions previously presented and from now on considered as features hypothetically able to influence the algorithm's performance. Specifically, the first pair of matrices models a more comprehensive interaction profile, while the second pair is sparser, with fewer interactions. This reduced interaction scenario characterizes the datasets labeled 'half' in the following table, reflecting a less dense regulatory network.

	Gene Expression Matrix		Motif Data		PPI matrix	
Dataset	Genes	Cells	TF	Gene Regulated	TF	Interactions
Toy	1000	51	87	14597	74	238
1k	23688	1000	1444	855452	869	10450
1k half	23688	1000	1457	429286	864	5079
4k	23688	4645	1444	855452	869	10450
4k half	23688	4645	1457	429286	864	5079
10k	23688	10000	1444	855452	869	10450
50k	23688	50000	1444	855452	869	10450
100k	23688	100000	1444	855452	869	10450

Table 4: Dimensions and name of each produced or considered dataset.

4.3 Time and memory analysis

This section presents all the results, in terms of execution time and memory usage, of the GRN computation performed by the PANDA algorithm. Each presented value has to be considered as the average value of a set of experiments reproduced several times in the same conditions specified. The final part of this section includes the complete set of experimental data from which all the presented plots are obtained.

4.3.1 Time performance

One of the key aspects of the PANDA execution is the amount of time in which it can converge and consequently generate a GRN representative of the matrices provided in the input. This aspect can become a critical bottleneck in the practical usability of this tool in a

hypothetical biological pipeline and its scalability to the number of cells considered in the expression matrix is the primary focus of the analysis here presented.

The first analysis centers on the difference between the time performance achieved by sequential execution of the PANDA algorithm, obtained using a single CPU, and a parallel approach, exploited using two types of GPUs and a multi-core CPU setup, as previously outlined in this chapter. The plots below (Figure 10) show the four most time-consuming tasks into which the PANDA algorithm can be divided. While performance results achieved using the "Toy Dataset" are not indicative of PANDA's scalability, the other five plots offer, in this sense, much more valuable insights, leading to the following considerations:

- ***Parallelization of the PANDA loop task***: this task is designed to be optimized, in terms of time efficiency, through parallel processing. It benefits significantly from both GPU and multi-core CPU configurations, but in particular, the GPU approach demonstrates high parallelism potential, achieving more efficient processing compared to single-core implementations.
- ***Independence of PANDA loop task time to the cell count***: As the number of cells increases changing datasets, the execution time of the PANDA loop remains unaffected, due to the fixed dimensions of the matrices after the initial normalization;
- ***Dependence of the correlation matrix computation to the cell count***: the time associated with this task is directly influenced by cell count, increasing with the increment of the number of cells. There are no constant differences between sequential and parallel computation and, more in general, there is no predictable behavior;
- ***Dependence of the loading expression matrix task to cell count***: this task can become a time-significant element as the number of cells increases, but presents constant performance across all the types of processors used, indicating that hardware differences have little impact on time efficiency;
- ***Insignificance of normalization step in time consumption***: Across all the experiments conducted, this normalization task doesn't contribute significantly to the overall time consumption, leading to the conclusion that it can be excluded in the considerations for the code optimization.

Figure 10: Comparison across all the datasets of the PANDA time performance, divided into tasks.

Dataset	Task	Hardware Setup			
		CPU	10xCPU	A40	RTX
1k	Loading expression matrix	1,3 s	1,3 s	1,5 s	1,2 s
	Correlation matrix computation	14,5 s	7,8 s	19,8 s	20,1 s
	Network normalization	46,9 s	46,1 s	26,9 s	24,4 s
	Executing the PANDA loop	142 s	741,9 s	164,1 s	156 s
4k	Loading expression matrix	17,6 s	82 s	23,3 s	16,5 s
	Correlation matrix computation	50,6 s	18,1 s	134 s	77,1 s
	Network normalization	46,8 s	46,6 s	31,8 s	25,1 s
	Executing the PANDA loop	1500,3 s	795,6 s	188 s	165 s
10k	Loading expression matrix	34,4 s	32,0 s	28,4 s	36 s
	Correlation matrix computation	87,6 s	15,7 s	68,7 s	146 s
	Network normalization	46,5 s	46,6 s	16,2 s	26,5 s
	Executing the PANDA loop	1421 s	674,8 s	161 s	147 s
50k	Loading expression matrix	514 s	545,3 s	544,6 s	600 s
	Correlation matrix computation	424,9 s	59,2 s	745,3 s	807,8 s
	Network normalization	67 s	64,9 s	31,3 s	33,1 s
	Executing the PANDA loop	1465,8 s	682,2 s	163,5 s	148,8 s
100k	Loading expression matrix	1133,7 s	1156,3 s	993,4 s	1171,5 s
	Correlation matrix computation	787,8 s	111,9 s	646,8 s	551,6 s
	Network normalization	50,4 s	48,4 s	17,7 s	27,1 s
	Executing the PANDA loop	1362,3 s	638,8 s	152 s	141,9 s

Table 5: Time performance of all the hardware setups considered divided by Dataset passed as input.

The following analysis examined the impact of interaction matrix (Motif Data and PPI Matrix) dimensions on PANDA's time performance. This set of experiments aimed to determine whether edge sparsity in the two co-regulatory networks affects the convergence time within the "PANDA Loop" task or if this process remains stable regardless of edge density. To test these hypothetical changes, two sparser versions of the Motif Data and of the PPI Matrix were simulated, each with an interaction frequency among TFs and between TFs

and genes reduced to half of the original, as described in the previous section. PANDA was then applied on these matrices and considering two different expression data matrices ("1k" and "4k").

The results of this analysis (shown in the plots of Figure 11 and described in Table 6), represented by the following plots, indicate no significant difference in the execution time between the original and the sparser version of the co-regulatory networks, suggesting that these matrix dimensions do not influence the core GRN computation. Code analysis confirms that the co-regulatory matrices are not directly involved in these steps and therefore do not impact their time complexity.

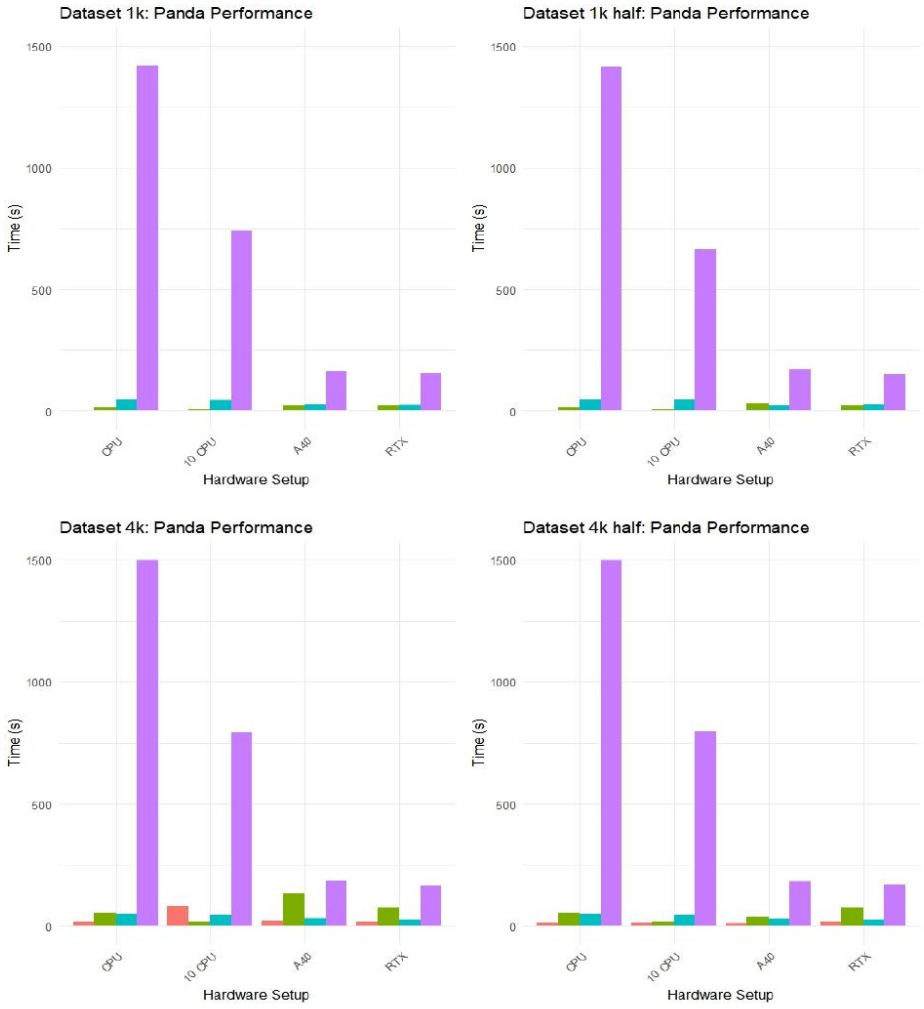


Figure 11: Comparison between time performance obtained by the use of the original co-regulatory matrices and their halved versions.

Dataset	Task	Hardware setup			
		CPU	10xCPU	A40	RTX
1k half	Loading expression matrix	1,3 s	1,3 s	1,4 s	1,5 s
	Correlation matrix computation	14,6 s	7,8 s	32,4 s	23 s
	Network normalization	47 s	47,4 s	33,9 s	28,7 s
	Executing the PANDA loop	1416,1 s	666 s	170,4 s	152 s
4k half	Loading expression matrix	16,5 s	15,9 s	12,6 s	17,6 s
	Correlation matrix computation	50,9 s	18,3 s	38,2 s	77,5 s
	Network normalization	47,7 s	46,8 s	16,3 s	24,7 s
	Executing the PANDA loop	1623 s	799,1 s	184,6 s	169 s

Table 6: Time performance of all the hardware setups considered for the 1k half and 4k half datasets.

4.3.2 Memory usage

The second part of the experiments focuses on the memory occupied by the computation of PANDA algorithm. While the time effort, even in the worst cases analyzed, can be usually considered manageable in real-world applications, being in the order of hours, the memory usage can be a more significant bottleneck as the dimensions of the expression matrix increase.

In this set of experiments, all available flags allowing to minimize memory computation were enabled. It's important to underline, however, that these options are not compatible with PANDA computations within the LIONESS pipeline. This is due to this last algorithm requirement for executing a high number of operations on the original matrices, which prevents discarding them as suggested by the memory-saver configurations.

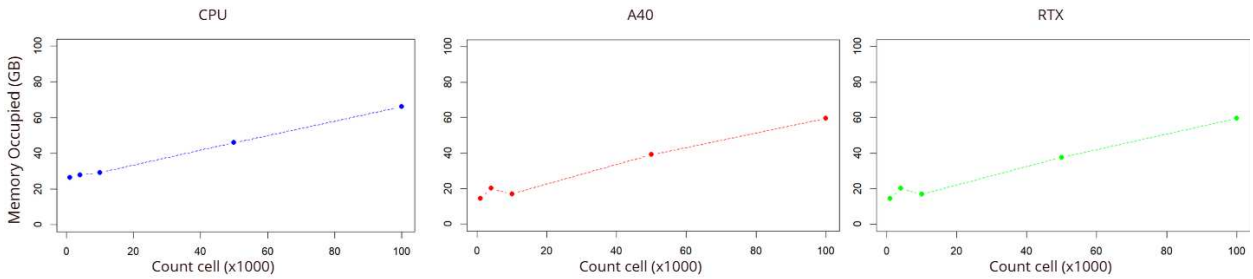


Figure 12: Comparison of memory usage in the three different hardware setups and with increasing count cells.

Hardware Setup	Dataset				
	1k	4k	10k	50k	100k
CPU	26,46GB	27,97GB	29,06GB	45,89GB	66,15GB
A40	14,21GB	20,3GB	16,8GB	37,67GB	59,51GB
RTX	14,45 GB	20,13GB	16,87GB	39,15GB	59,51GB

Table 7: Memory usage among all the hardware setups for the considered datasets.

This set of data (Figure 12 and Table 7) reveals a consistent trend across all three different hardware configurations: there is an increasing trend between the memory usage and the expression matrix dimension in all the simulated datasets. However, due to the limited number of data points, it remains unclear whether this relationship is linear or otherwise, preventing precise determination of the scaling coefficient governing this trend. The exception for this relationship across all three cases is the dataset with around 4 thousand cells, which shows unexpectedly high memory usage. These anomalies correspond with the only real dataset considered, suggesting that it has features influencing the use of memory that have been not considered in the simulation of the other dataset, leading to this different memory occupancy data.

4.3 Results Analysis

From the previously exposed analysis, it has been possible to obtain the following considerations about the core algorithm characteristics:

- ***Message Passing Approach***: the foundational principle of the algorithm, the message passing approach, achieves convergence within a finite number of steps, generating a GRN that integrates all the information contained in the three input matrices with a certain grade of concordance;
- ***Designed parallelism***: PANDA's parallelism can be exploited through two different hardware setups [36]:
    - *using a multi-core CPU approach*: the time required for the PANDA loop computation by 10 cores of the considered CPU is half of the time required by the single-core CPU. This cannot be considered a significant improvement because of the proportion between the increment of resources and the time efficiency obtained;

    - *using a GPU approach*: using these specific hardware setups, the time complexity of the PANDA's loop can be reduced to one-tenth of the one registered to compute the same process using a sequential approach;
- ***Partial Task Parallelization***: the only task optimized in the parallel version is the execution of PANDA loop, while tasks such as the loading of the expression matrix and the computation of the correlation matrix are not, becoming a limiting factor as the dimensions of the data increase;
- ***Stability of PANDA loop execution time:*** the time required for this task depends solely on the size of the correlation matrix created from the expression matrix passed as input. Since this is a square matrix with dimensions equal to the number of genes considered, the time complexity of this task is independent of the cell count variations;
- ***Independence from the sparsity of the coregulation matrices:*** the set of tests executed for the comparisons between the performance related to input with motif and PPI data and their half versions, confirm the independence of the algorithm's time complexity from their dimensions;
- ***Intrinsic parallelization in correlation matrix computation***: Although not explicitly optimized for parallel processing, this task benefits the use of a package in which the parallel computation is designed, achieving, in the case of multi-core hardware setup, time performance from 3 to 7 times lower than the ones reached by all the other setups;
- ***Memory usage:*** there's a general trend, presented in Plot 4, for all the statistics related to memory occupied by the computation of the simulated dataset. However, this trend seems not to be followed by the real dataset, raising the possibility of not being able, in the simulation phase, to fully capture all memory-relevant features;
- ***Sequential Nature of the PANDA loop***: Although the message-passing approach used in the PANDA loop produces biologically meaningful results, it is inherently sequential, with the number of loop iterations representing a lower bound on the algorithm's critical length.

Based on these considerations, this study has explored ways to increase this tool's scalability in terms of computational time. All the limitations identified by this analysis presented in this chapter have been considered as a starting point in developing modified versions of specific parts of PANDA. It's crucial to underline that not all the considerations done in this paragraph have led to modifications; some features have been considered too relevant to modify,

maintaining them in the form in which they have been proposed originally, for instance, the loop structure on which the message passing approach has been exploited. Others, regarding memory occupancy improvement or implementing a multi-GPU approach, were not pursued during the development of this work but can represent promising directions on which there is room for improvement.

4.4 LIONESS

The second algorithm considered for the analysis of GRN inference performance is LIONESS. It, as described in the earlier chapters, utilizes as input the PANDA object produced by the omnibus software, and for each cell (or, more generally, sample) within the expression matrix creates a specific GRN. It's clear that some of the memory-saving options used by PANDA, cannot be applied to this program because require the deletion of the original expression matrix, which LIONESS requires at the start of each computation.

The structure of this program has features more adaptable to parallelization compared to PANDA's: the computation of each GRN sample-specific can be executed in parallel, not depending on the results of the other steps, relying only on the aggregate network produced by the initial PANDA execution. The primary limit on the parallelization of this algorithm appears to be the hardware setup used, offering also a multi-GPU procedure to decrease further the time performance. These premises have not been followed by encouraging results: the time complexity of the computation executed on a sequential approach using the "1k Dataset" required over 2 hours, while the memory requested by the use of a GPU exceeded the already large amount of memory allowed to the computation (>200GB). Consequently, these results have forced the stop of the testing on all the larger datasets, not producing data on which analysis can be exploited.

The code simplicity, along with various attempts to mitigate time and memory complexity (such as limiting output to two sample-specific GRNs instead of generating one for each sample), suggest that the tested package version (netZooPy 0.8) may not be optimized for large-scale or high-sample computations.

While the tests and the analysis on this second algorithm have not produced the expected results, it's essential to signal that all the limitations presented for PANDA are equally pertinent for LIONESS. These represent important points on which all the improvements can generate a significant impact also on LIONESS, due to the high number of calls of PANDA required by this algorithm. Therefore, while the implementation of LIONESS seems to be the most concrete and relevant problem, an optimization of the code and a parallelization of some

PANDA tasks can lead to a couple of usable (in terms of time and space required) tools, able to generate sample-specific GRNs starting from expression matrices with a high number of samples.

5. Improvements to the PANDA Algorithm

The preceding analysis highlighted the intrinsic limitations of the PANDA algorithm, which can have scalability issues considering datasets with biologically relevant and realistic cell counts. Consequently, the next phase of this study involves identifying specific sections of the code to modify, focusing on the most time-intensive tasks that are likely to become critical bottlenecks in single-cell applications. These tasks can be identified with whose performance directly depends on the number of cells in the expression matrix, specifically: the loading of the expression matrix and the computation of the correlation matrix.

In contrast, the main PANDA loop is excluded from this consideration due to its consistent time performance, which is influenced solely by the number of genes in the dataset. Furthermore, the loop has already exploited parallelization by the NetworkZoo programmer team, using GPU architectures for Tanimoto similarity computation and the update function.

Finally, this chapter will present a comparative analysis of the execution times for these two tasks—matrix loading and correlation computation—between the improved implementation and the original version, providing insights into the results achieved through GPU parallelization of these tasks.

5.1 Loading the Expression Matrix

The loading of the expression matrix, identified as one of the most time-intensive tasks in the earlier analyses, requires a deeper examination of the implemented code. Specifically, the results from tests conducted across various hardware configurations reveal no substantial or consistent performance differences between the sequential approach and the use of parallel architecture: multi-core CPU and GPUs load the expression matrix at the same time required by a sequential approach. This observation indicates that the current function used for this task — read_csv from the PANDAs library— lacks optimization for parallel processing.

```
FUNCTION ImprovedLoadingExprMatrix

    IMPORT necessary libraries

    CLASS Timer

        ENTER: Start timer

        EXIT: Print elapsed time
```

```
    FUNCTION rows_counter(filename)
        OPEN filename
        COUNT newlines in filename
        RETURN total count
    FUNCTION load_chunk(interval, filename)
        READ file chunk based on interval with pandas
        RETURN chunk as DataFrame

    SET filename, cpus
    CALCULATE nrows using rows_counter(filename)
    DETERMINE chunk_size (nrows / cpus)
    CREATE intervals list: [(start, end) for each CPU]

    WITH Timer("Parallel Import"):
        WITH Pool(cpus):
            PARALLEL MAP load_chunk over intervals
            MERGE results into data_parallel DataFrame
END FUNCTION
```

Pseudo-code 1: Improved version of the loading of the expression matrix

To address this limitation, it is necessary to design an improved implementation capable of simultaneously loading distinct sections of rows (referred to as "chunks") and subsequently merging them into a unique data frame. Initially, the developed solution, shown in Pseudo-code 1, determines the total number of rows in the input text file containing the expression matrix, counting the number of row delimitators. The program divides the dataset into evenly sized chunks based on the number of processors available for the parallel computation. Each chunk is then read simultaneously by individual processors, using ‘pandas.read_csv’ function, the same employed in the original implementation. Finally, the chunks are concatenated to

reconstruct the complete expression matrix, which is required for the PANDA algorithm input.

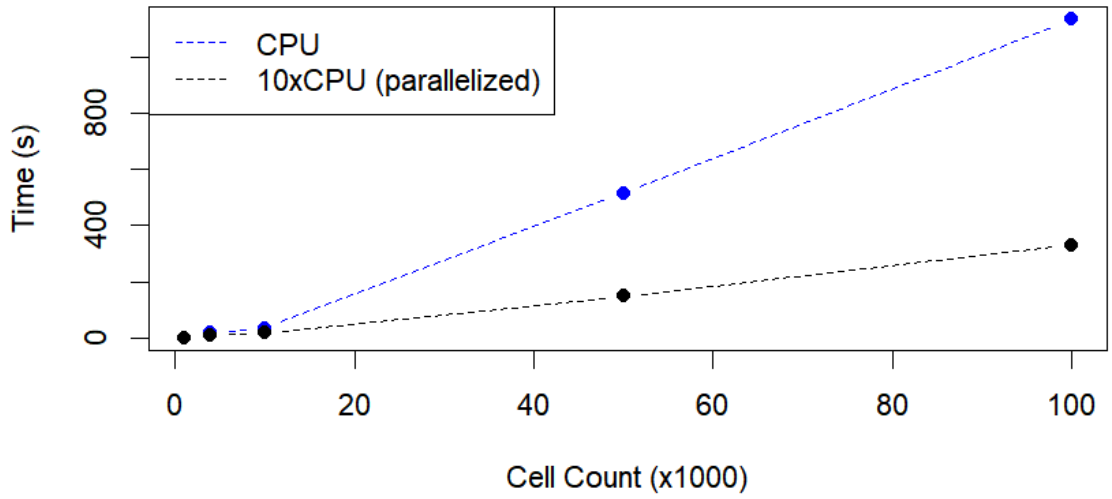


Figure 13: Comparison between the time performance, across all the datasets, achieved using a sequential approach (blue) and a parallel approach using a multi-core CPU (black)

Hardware setup	Dataset				
	1k	4k	10k	50k	100k
CPU	1.32 s	17.61 s	34.5 s	514 s	1133.78 s
10xCPU	0.76 s	8.7 s	17.12 s	148.09 s	329.43 s

Table 8: Time performance, across all the datasets, of loading expression matrix task using CPU and multi-core CPU.

Figure 13 and Table 8 demonstrate that the improved implementation for loading the expression matrix achieves significant time efficiency compared to the sequential algorithm. While the performance of the two approaches is comparable for datasets with fewer than 10,000 cells, the time-saving advantage of the improved program becomes increasingly significant as the cell count rises. For datasets such as ‘50k’ and ‘100k’, the parallel approach reduces execution time to approximately one-fourth or one-fifth of that required by the sequential method. This different time complexity is given by the use of simultaneous processors and is mitigated by the time needed for the concatenation of all the chunks, a task not required by the original code.

In this loading optimization, the GPU architectures achieve results comparable to the ones obtained by the multi-core CPU approach and, therefore, are not presented in Figure 13 and Table 7. Indeed, the parallelization of this code section is implemented by dividing the data frame into chunks and employing a sequential function to read each of them simultaneously, unable to support a GPU architecture and leading to the conduction of this task by CPUs with the number of cores specified by the programmer. As a result, the performance differences are due to the different hardware setups and are not consistent and substantial.

The improvement produced by this different approach changes drastically the impact of this task on the general amount of time consumed and, considering also that: (1) the tests here presented indagate only parallelism exploited by a multi-core CPU, (2) in all the LIONESS workflow this task is executed once; this reduction of the time consumed to one-fourth of the original version can be considered a substantial improvement.

5.2 Calculating the Correlation Matrix

The computation of the correlation matrix has been identified as another key task with important room for improvement. This consideration is given, not only by the large times registered and by their general increasing trend, but also by the type of calculation needed to generate the correlation matrix. Each cell of this output matrix is produced by the calculation of the Pearson correlation coefficient, ρ_{XY} , of the two genes, X and Y, taken as coordinates of the new matrix, expressed by this formula:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

where σ_X is the standard deviation of the row associated with the gene X in the expression matrix, and σ_{XY} is the covariance between the genes X and Y. A notable feature of this computation is the independence between coefficients. For instance, calculating ρ_{XY} does not depend on the computation of ρ_{XZ}, with all required standard deviations precomputed. This independence underscores the high degree of parallelizability inherent in this task. In the PANDA algorithm, the correlation matrix is computed using the corrcoef function from the NumPy library. This function, as shown in the plot presented in Figure 14 and from data in Table 9, already exploits some level of parallelism. These observations indicate that, while the original implementation benefits from parallel computation, further optimization could

yield even more substantial efficiency improvements using a parallel approach able to exploit the GPU potentialities in an improved version.

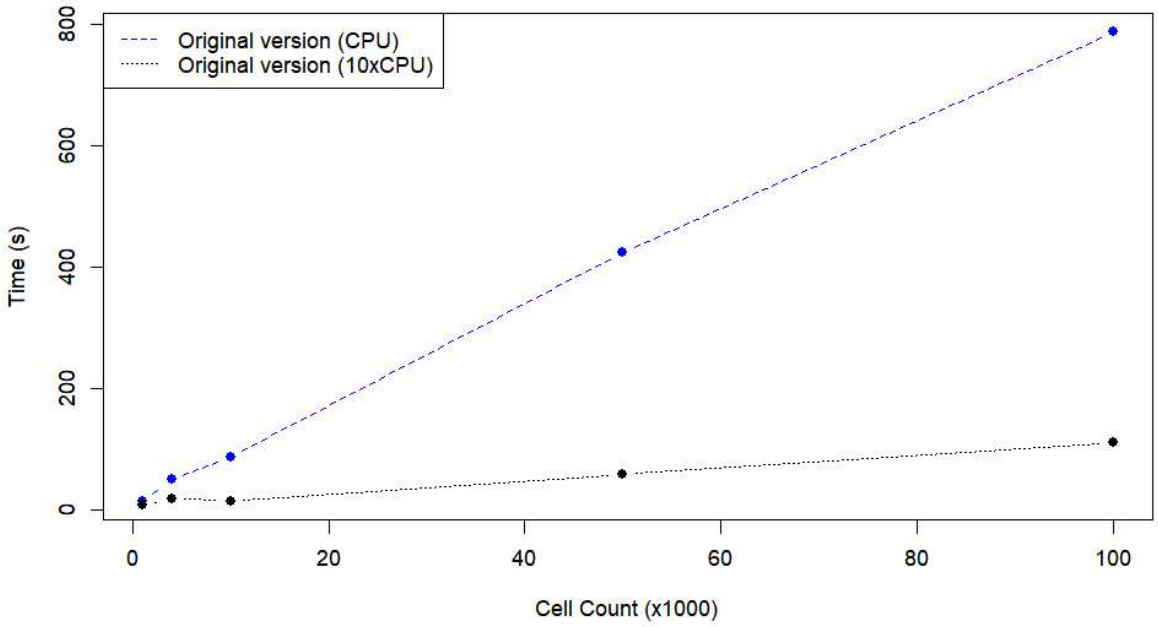


Figure 14: Comparison across time performance of correlation matrix computation of the original version using: CPU (sequential approach), and 10 cores of CPU (parallel approach).

Hardware setup	Dataset				
	1k	4k	10k	50k	100k
CPU	14,5 s	50,6 s	87,6 s	424,9 s	787,88 s
10xCPU	7,87 s	18,6 s	15,76 s	59,21 s	111 s

Table 9: Time performance achieved by CPU and multi-core CPU structures across all the datasets for the correlation matrix computation task with the original implementation.

The modifications implemented focused on the application of the GPU computation on this task. Specifically, the NumPy library, traditionally used for this task, was replaced with CuPy, a GPU-accelerated library from the Rapids ecosystem. This change abled the shift of the correlation computation job from the CPU to the GPU, with tests conducted using two hardware configurations: NVIDIA RTX and A40 GPUs.

The results, presented in the plot shown in Figure 15 and in Table 10, highlight the comparison of the performance between the improved version (GPU-based computation) and the original version (best-performing CPU-based setup, the multi-core CPU). For datasets with smaller cell counts, the execution times across all implementations, original and

improved, can be considered equivalent. However, as the dataset size increases, the advantages of GPU acceleration become increasingly significant, with both the RTX and A40 allowing the improved configuration to achieve lower execution times than the original version. Notably, there is no substantial performance difference between the RTX and A40 GPUs, as both achieve comparable speedups over the multi-core CPU approach.

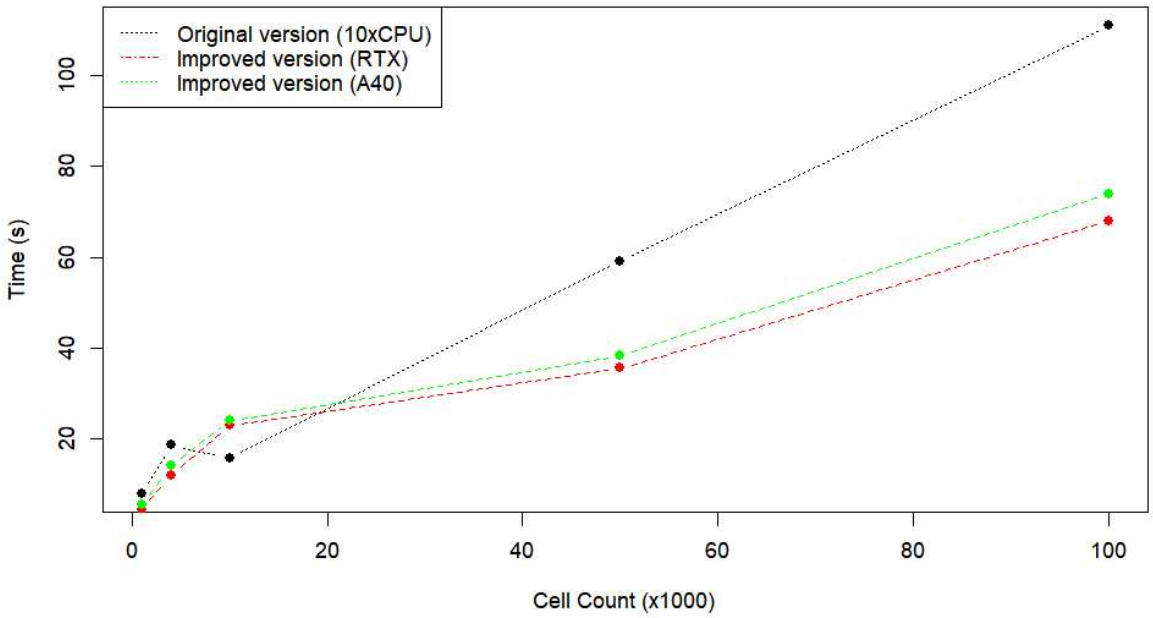


Figure 15: Comparison across the time performance achieved by the original implementation using 10 cores of the CPU(black line) and the improved implementation using RTX (red line), A40 (green line) GPUs for the correlation matrix computation task.

Hardware setup	Dataset				
	1k	4k	10k	50k	100k
10xCPU	7,87 s	18,16 s	15,76 s	59,21 s	111 s
RTX	4,4 s	12 s	23 s	35,54 s	68 s
A40	5,49 s	14,1 s	24 s	38,34 s	74 s

Table 10: Times performed by 10xCPU, RTX, and A40 architecture across all the datasets to compute the correlation matrix, using the two different implementations, the original and the improved.

The modifications introduced to improve this code section provide significant results, lowering notably its time consumption. Specifically, these results assume privileged importance not only because of the demonstrative potentialities, underling effectively the

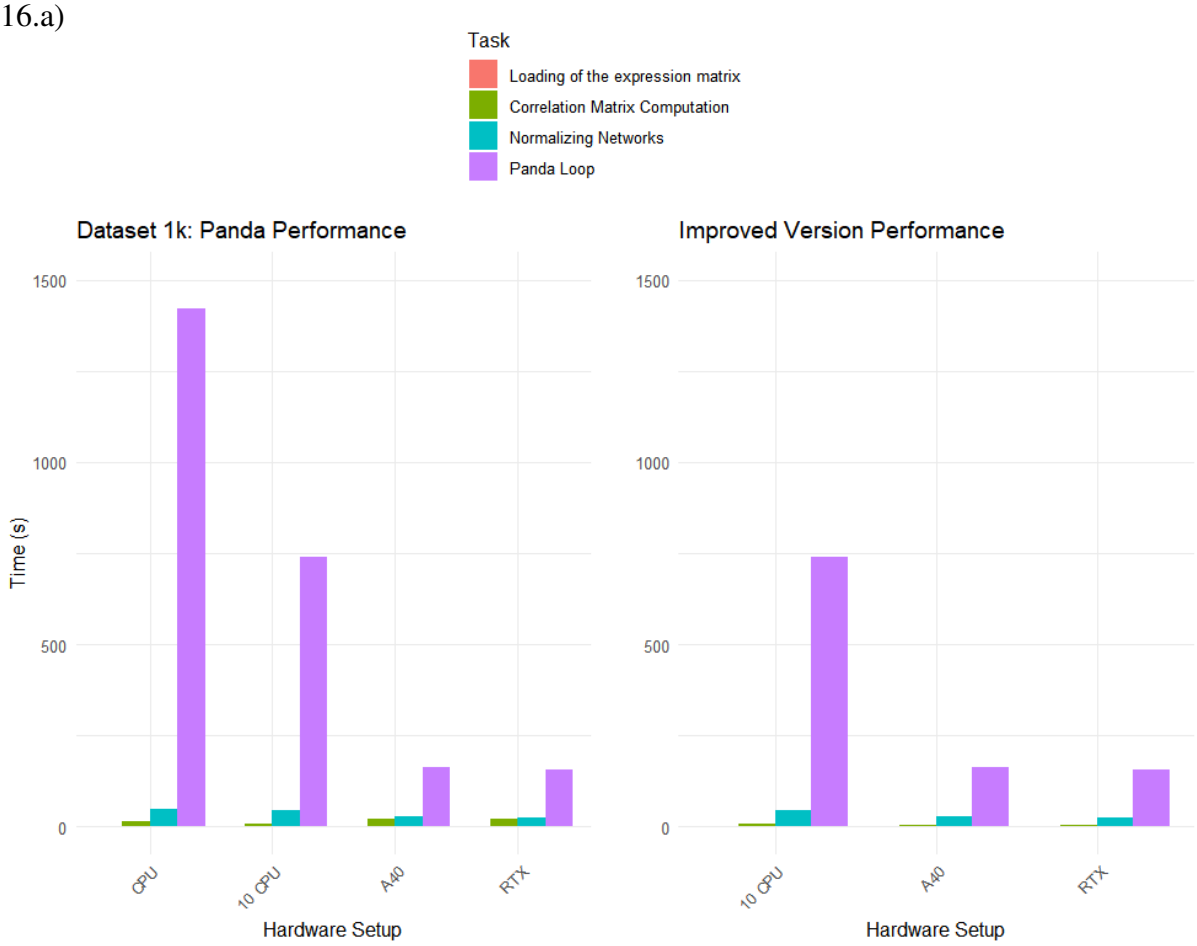
power of a parallel approach compared to the traditional computational way of solving problems, but also for the consequences on the two GRN inference algorithms, PANDA and LIONESS. For PANDA, these are relevant updates because allow the use of the GPU setup without time efficiency problems related to preliminary tasks; for LIONESS, the improvements are even more critical, as the algorithm requires the computation of a correlation matrix for each individual sample in the dataset.

In conclusion, these modifications, regarding only targeted sections of code enable the parallelization of the two most time-consuming tasks identified by the analysis. This not only reduces the time required for each computation but also establishes a parallelized framework that can support further optimization efforts, leading to inference sample-specific GRN algorithms able to elaborate realistic single-cell matrices in an acceptable amount of time.

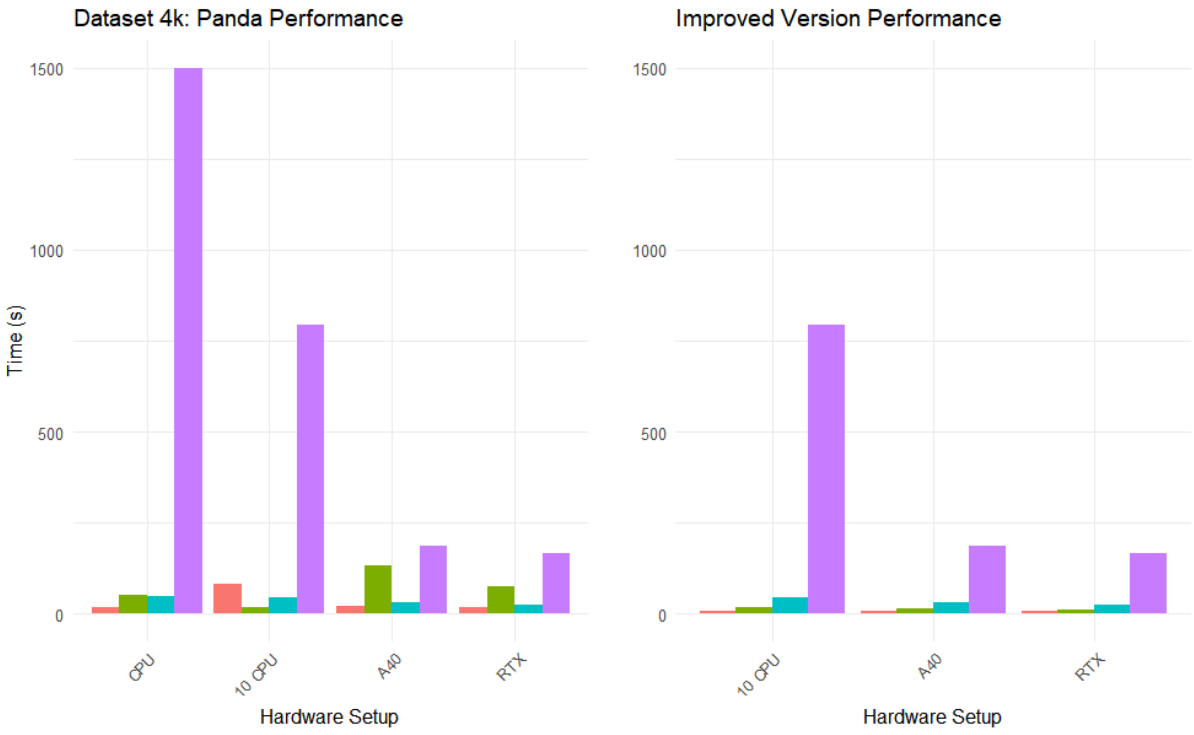
5.3 Results Achieved

This paragraph presents the difference in terms of time consumed in the PANDA workflow between the original GPU implementation of this inference GRN algorithm and the proposed improved version.

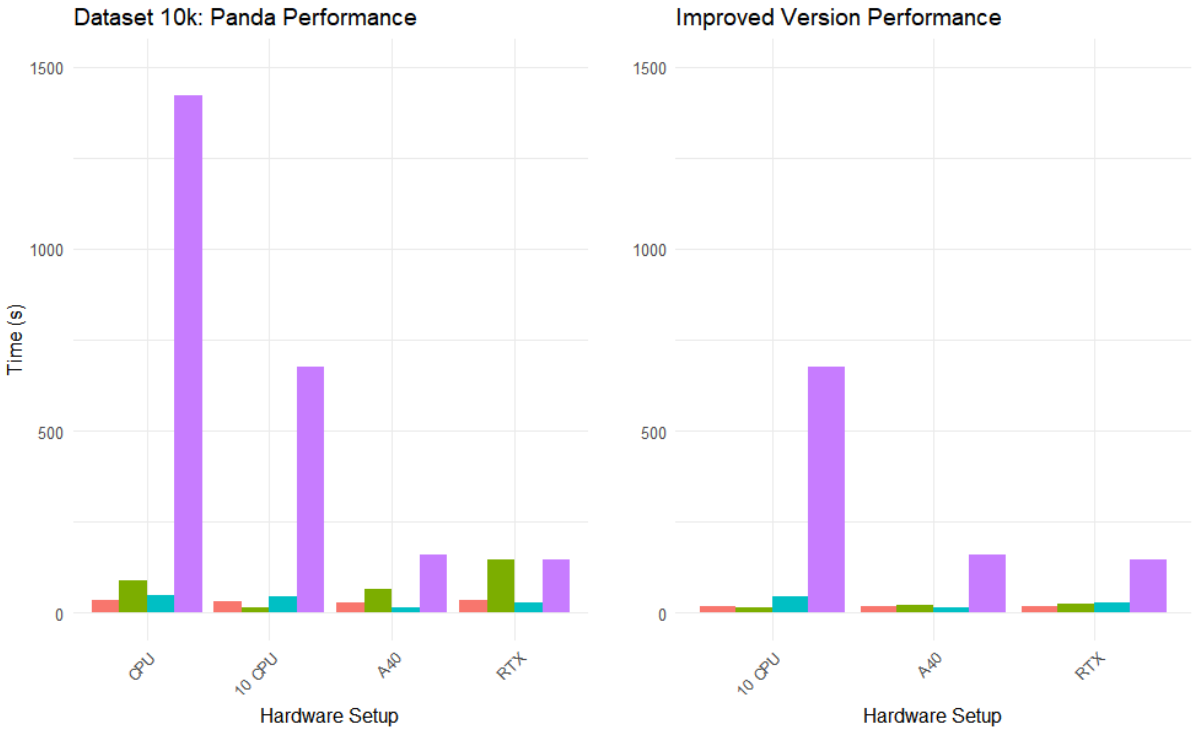
16.a)



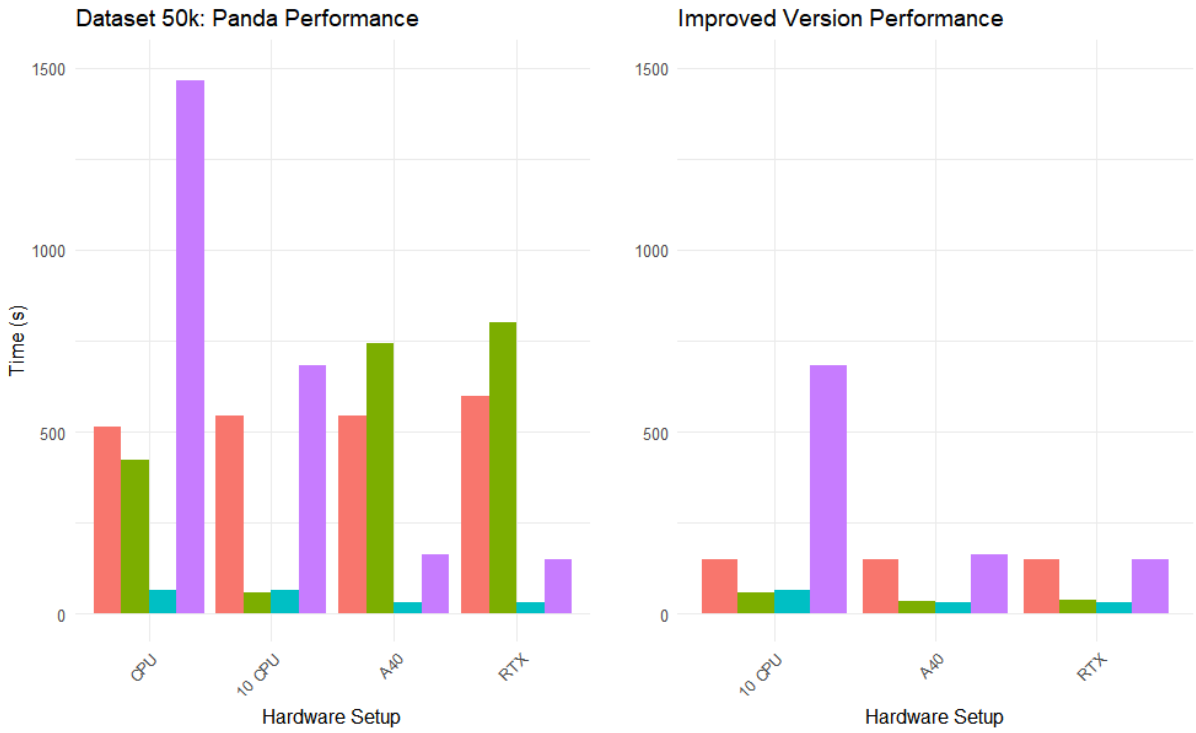
16.b)


Dataset 4k: Panda Performance
Improved Version Performance
Time (s)
1500
1000
500
0
CPU
10 CPU
A40
RTX
Hardware Setup

16.c)


Dataset 10k: Panda Performance
Improved Version Performance
Time (s)
1500
1000
500
0
CPU
10 CPU
A40
RTX
Hardware Setup

16.d)



16.e)

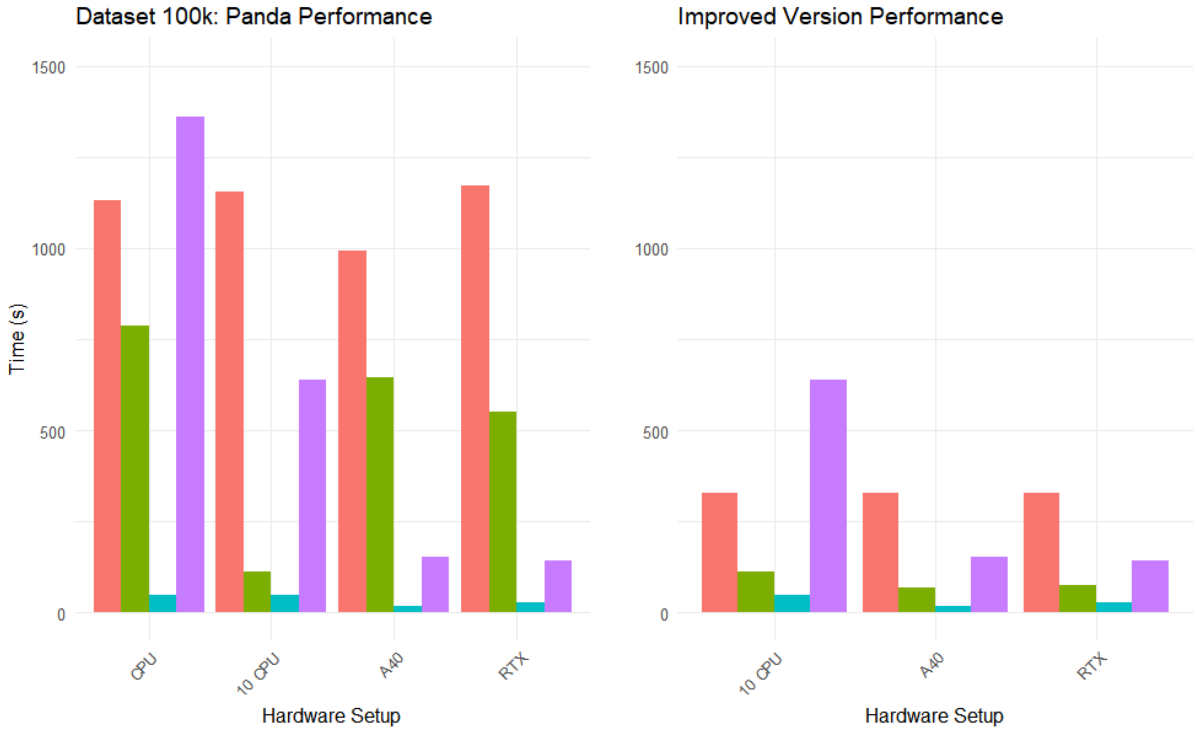


Figure 16 (a,b,c,d,e): Comparison between the time consumption of the PANDA algorithm and of the modified version.

Figure 16 compares the performance achieved through the parallelization of the 2 preliminary tasks with the time consumption of the PANDA original algorithm. While in the case of a

dataset with a cell count equal or lower than 10 thousand (Figure 16 a,b,c) the differences are not significant, for the largest expression matrices (Figure 16 d, e) considered the time gains reach the following results:

- ***Loading expression matrix***: The improved version is at least three times faster than the sequential approach in the 50k and 100k Datasets, respectively;
- ***Correlation matrix computation***: the use of the GPU architecture can lead to the execution of this task 2 times faster than the multi-core CPU setting and from 7 to over 10 times faster than the sequential approach, using as reference the two most large datasets and depending on the hardware setup considered.

In conclusion, the obtained results demonstrate how the GRN inference workflow can be analyzed and improved in different tasks, such as the ones modified in this version, achieving a more scalable and efficient product. In particular, this study proves the potentialities and the applicability of the state-of-the-art general-purpose GPU software applied to the bioinformatic field, enabling the handling of multi-omic data with reasonable time consumption.

6. Conclusion

This study aims to analyze the advantages and limitations of an inference GRN algorithm, PANDA, and a sample-specific GRN inference software, LIONESS, evaluating the possible applications of parallel computing and GPU optimization to enhance their computational performance. Initially, there is a presentation of the state-of-the-art GPU technology applications in the bioinformatic field: they can be considered pivotal in the future development of bioinformatic pipelines. These suites of software can change drastically the process of analyzing NGS data in each step (secondary and third analysis) of the genomic pipeline, allowing the computation of fundamental biological tasks, such as the Whole Exome Analysis, in a fraction of the time normally required by the sequential approach.

Once this parallel ecosystem excursus is completed, the focus is maintained on the two core algorithms, PANDA and LIONESS. This couple of software is based on the message-passing approach: an iterative process having the goal of finding a network with the highest possible accordance among the data matrices used as input, representing the relations between TFs and regulated genes.

This explanation is followed by the performance analysis of PANDA in terms of time consumed and memory occupied. All the tests have been conducted on a set of datasets simulated from a real one, abstracting its main characteristics, such as the sparsity of the matrix and the sequencing depth. The variables whose impact has been evaluated in this analysis are (1) the hardware setup, using CPU, multi-core CPU, and two different GPUs; (2) the cell count of the expression matrix; (3) the sparsity of regulation between a specific TF and the genes, and among TFs.

Memory usage follows an increasing trend as the cell count increases; it's relevant to underlining the tendency of the only real dataset, 4k, to require a value of memory higher than the one expected by the trend of the other datasets, behavior confirmed among all the hardware setups tested. The time analysis has revealed how the co-regulation matrix dimensions don't affect the execution time, and how the total time required by PANDA is subdivided among all the relevant tasks that compose it. In particular, the loading of the expression matrix, the correlation matrix computation, and the PANDA loop dominate the time complexity, presenting designed parallel optimization only for this last task.

For the first two tasks, some parallel improvements have been proposed considering, using parallel computing strategies exploiting multi-core CPU and GPU. Finally, in the following

paragraphs, the results obtained are compared with the original performance, and the potential future improvements are described.

6.1 Limitations and potential future improvements

While the modifications previously discussed have led to significant improvements, several aspects represent relevant limitations for the real applicability of these couple of tools. At the same time, these can be considered as opportunities for future improvements to optimize further the scalability of this software. These aspects, divided by argument, are:

- ***Optimization of the expression matrix loading***: the modifications have reduced the time consumption of the parallel approaches. However, from the previous comparison, this task results still dominate, in terms of time, among all the other tasks as the cell count increases. Potential future improvements are the evaluation, implementation, and test of the state-of-the-art GPU-optimized I/O functions for the loading of matrices;
- ***Optimization of the correlation matrix computation***: this task is directly dependent on the cell count, therefore can be useful to consider:
    - Testing specialized specific bioinformatic packages, such as Rapids-singlecell, different from the general-purpose ones used in this study, to evaluate theoretically and practically the time and memory performance;
    - A multi-GPU approach, evaluating which packages can store and operate on large matrices;
- ***Redesign of the PANDA loop structure***: Evaluate how this typically sequential strategy can be rethought as a parallel process, aiming to design a more efficient way of representing this workflow using a parallel architecture;
- ***Optimization of the memory usage***: this aspect represents the main limitation after the modifications provided to alleviate the time consumption problems, therefore it would be important to:
    - Establish the dependence between the memory usage and the dimensions of the input matrices;
    - Evaluate how the internal variables and matrices are stored and distributed in a GPU;
- ***LIONESS analysis***:
    - Evaluate and solve the problems that have hindered the testing of this algorithm, in particular, the motivation behind the high time and space consumption associated with a not significantly large expression matrix;

- Execute a performance analysis among different hardware setups looking for the software limitations and the potential optimizations;
- Manage efficiently the allocations of the variables and matrices, guaranteeing continuity with the memory architecture used for the PANDA algorithm.

With these analyses and this set of potentialities concretized, PANDA and LIONESS will be finally able to perform GRN inference on large single cell RNA-seq data with limited computational burden.

Bibliography

[1] Sumida et al., *The regulation and differentiation of regulatory T cells and their dysfunction in autoimmune diseases,* Nature, 2024.

[2] Zhu et al., *Cell signaling and transcriptional regulation of osteoblast lineage commitment, differentiation, bone formation, and homeostasis*, Nature, 2024.

[3] Yokota, *Osteoclast differentiation in rheumatoid arthritis*, Immunological Medicine, 47(1), 6–11, 2023.

[4] Haseltine et al., *The RNA Revolution in the Central Molecular Biology Dogma Evolution,* Molecular Sciences, 2024.

[5] Koonin, *Why the Central Dogma: on the nature of the great biological exclusion principle*, SpringerNature, 2015.

[6] Querido et al., *The molecular basis of translation initiation and its regulation in eukaryotes*, Nature, 2024

[7] Goldberger, *Biological Development and Regulation – Gene Expression*, National Cancer Institute, 2012.

[8] Yuan et al., *Inferring gene regulatory networks from single-cell multiome data using atlas-scale external data*, Nature, 2024.

[9] Phan et al., *Studying temporal dynamics of single cells: expression, lineage and regulatory networks*, Springer Nature, 2024.

[10] Hunyuh-Thu et al., *Gene Regulatory Network Inference: An Introductory Survey*, SpringerNature, 2018

[11] Ozsolak et al., *RNA sequencing: advances, challenges and opportunities*, Nature Review Genetics, 2011.

[12] Kumar et al., *Next-Generation Sequencing and Emerging Technologies*, Thieme, 2024.

[13] Wang et al., *Single cell analysis: the new frontier in 'omics', Trends in Biotechnology*, 2010.

[14] Potter, *Single-cell RNA sequencing for the study of development, physiology and disease*, Nature, 2018.

[15] Stegle et al., *Computational and analytical challenges in single-cell transcriptomics*, Nature Review Genetics, 2015.

[16] Mercatelli et al., *Gene regulatory network inference resources: A practical overview*, ScienceDirect, 2020

[17] Zhao et al., *A comprehensive overview and critical evaluation of gene regulatory network inference technologies*, Briefing in Bioinformatics, 2021.

[18] Salleh et al., *Reconstructing gene regulatory networks from knock-out data using Gaussian Noise Model and Pearson Correlation Coefficient*, ScienceDirect, 2015.

[19] Elembaby et al., *Comparing gene regulatory inferring algorithms with different perspective*, Instrumentation, Mesure, Métrologie, 2018.

[20] Pirgazi et al., *A robust gene regulatory network inference method base on Kalman filter and linear regression*, Plos one, 2018.

[21] Liu et al., *Inference of Gene Regulatory Network Based on Local Bayesian Networks*, Plos Computational Biology, 2016.

[22] Akers et al., *Gene regulatory network inference in single-cell biology*, ScienceDirect, 2021.

[23] Aibar et al., *SCENIC: single-cell regulatory network inference and clustering,* Nature, 2017.

[24] Moerman et al., *GRNBoost2 and Arboreto: efficient and scalable inference of gene regulatory networks*, Bioinformatics, 2019.

[25] Paul M. et al., *Reconstruction of gene regulatory networks using graph neural network,* ScienceDirect, 2024.

[26] Kharchenko, *The triumphs and limitations of computational methods for scRNA-seq,* Nature Methods, 2021.

[27] Lythal et al., *Normalization Methods on Single-Cell RNA-seq Data: An Empirical Survey*, Frontiers, 2020.

[28] Bhardwaj et al., *Limitations of scRNA-seq Zero-Imputation Methods for Network Inference*, OpenReview, 2024.

[29] Badia-i-Mompel et al., *Gene regulatory network inference in the era of single-cell multi-omics*, Nature, 2023.

[30] Glass et al., *Passing Messages between Biological Networks to Refine Predicted Interactions*, Plos One, 2013.

[31] Kuijjer et al., *Estimating Sample-Specific Regulatory Networks*, iScience, 2019.

[32] Web Site: Parabricks Documentation, https://docs.nvidia.com/clara/parabricks/latest/index.html

[33] O'Connell et al., *Accelerating genomic workflows using NVIDIA Parabricks*, SpringerNature, 2023.

[34] Web Site: Nvidia Parabricks Documentation, "Welcome to Nvidia Parabricks v4.3.1 – Software Overview – Software Tools", https://docs.nvidia.com/clara/parabricks/latest/documentation/tooldocs/standalonetools.html

[35] H. Clifford & P.Vats, *Accelerate Whole Exome Analysis with Deep Learning at 70% Cost Reduction Using NVIDIA Parabricks*, Nvidia Developer Blog, 2023.

[36] Web Site: Nvidia RAPIDS, https://www.nvidia.com/it-it/deep-learning-ai/software/rapids/

[37] Hricik et al., *Using RAPIDS AI to Accelerate Graph Data Science Workflows*, IEEE, 2020.

[38] Web Site: Rapids-singlecell GitHub repository, https://github.com/NVIDIA-Genomics-Research/rapids-single-cell-examples

[39] Web Site: scverse, https://scverse.org/

[40] Web Site: Anndata, https://anndata.readthedocs.io/en/stable/

[41] S.Dicks & C.Nolet, *GPU-Accelerated Single-Cell RNA Analysis with RAPIDS-singlecell*, Nvidia Developer Blog, 2023.

[42] Web Site: Netzoopy, GitHub repository, https://github.com/netZoo/netZooPy

[43] Guebila et al., *gpuZoo: Cost-effective estimation of gene regulatory networks using the Graphics Processing Unit*, NAR Genomics and Bioinformatics, 2022.

Acknowledgments

I would like to warmly thank all the people who made this important journey and project possible.

A special thanks to Prof. Baruzzo for serving as the supervisor of this thesis, for his availability, and for the attention he dedicated to me over the past months. My gratitude also goes to Prof. Di Camillo for her valuable guidance during the development of this project, and to Dott. Cesaro for her assistance in implementing the tests presented and for her readiness to help.

I am also grateful to Dr. Loris Bertoldi for the support and advice on these topics, as well as to the entire BMR Genomics team for allowing me to concretely apply my studies in such a welcoming work environment.