Analyzing Wet-Neuromorphic Computing Using Bacterial Gene Regulatory Neural Networks

Samitha Somathilaka, *Member, IEEE,* Sasitharan Balasubramaniam, *Senior Member, IEEE*
Daniel P. Martins, *Member, IEEE,*

***Abstract*—Biocomputing envisions the development computing paradigms using biological systems, ranging from micron-level components to collections of cells, including organoids. This paradigm shift exploits hidden natural computing properties, to develop miniaturized wet-computing devices that can be deployed in harsh environments, and to explore designs of novel energy-efficient systems. In parallel, we witness the emergence of AI hardware, including neuromorphic processors with the aim of improving computational capacity. This study brings together the concept of biocomputing and neuromorphic systems by focusing on the bacterial gene regulatory networks and their transformation into Gene Regulatory Neural Networks (GRNNs). We explore the intrinsic properties of gene regulations, map this to a gene-perceptron function, and propose an application-specific sub-GRNN search algorithm that maps the network structure to match a computing problem. Focusing on the model organism Escherichia coli, the base-GRNN is initially extracted and validated for accuracy. Subsequently, a comprehensive feasibility analysis of the derived GRNN confirms its computational prowess in classification and regression tasks. Furthermore, we discuss the possibility of performing a well-known digit classification task as a use case. Our analysis and simulation experiments show promising results in the offloading of computation tasks to GRNN in bacterial cells, advancing wet-neuromorphic computing using natural cells.**

***Index Terms*—Biocomputing, Neuromorphic Computing, Bacteria, Gene Regulatory Network.**

I. Introduction

Bacterial computing is an emerging field within the broader discipline of biocomputing [1]. The inherent computing properties of bacteria enable them, in particular, to sense their environment, make decisions, and adapt to changing conditions [2], with remarkable efficiency [3]. These characteristics, along with biocompatibility, parallelism, self-sustainability [1], communication capabilities [4], and data storage [5], tend to provide bacterial computing with an edge over conventional silicon-based computing architectures. Furthermore, the concept of neuromorphic computing is gaining traction, inspired by the workings of neurons, showing promise compared to Von Neuman computing architectures [6]. By integrating natural biocomputing into neuromorphic systems, researchers are now exploring wet-neuromorphic computing, where living cells are used in tandem with silicon technology. This has resulted in new paradigms such as organoid intelligence [7], one of which is the brain organoid. The "Dishbrain" is one example of many approaches to such systems, which is a brain organoid that harnesses the inherent adaptive computation of brain neurons within a structured environment that is capable of learning and performing complex tasks, such as playing a game of Pong [8].

Past research has extensively analyzed the functional components of natural bacterial computing, revealing a complex interplay of molecular processes that results in their decision making and adaptive behaviors [9]. Signal transduction mechanisms facilitate adept extracellular information reception, followed by the complex orchestration of transcription and translation processes, resulting in a sophisticated computational architecture within bacterial cells using their Gene Regulatory Networks (GRNs). The synergy between these processes emphasizes the profound complexity of bacterial computing, underscoring its potential as a model for advanced bioinspired computing paradigms within the confines of a single bacterial cell.

Synthetic biology has enabled researchers to use conventional computing theories that are engineered into cells by precisely altering and modifying biological components [10], [11]. Although the feasibility of using biological substrates for computing has been established since Adleman's seminal work in 1994 [12], a number of proposals have been made for the use of bacterial cells for computing. An example is Levskaya et al., who proposed the bacterial cell as a programmable computational device [13]. Expanding on this concept, Baumgardner et al. successfully programmed *Escherichia coli* (*E. coli*) with a genetic circuit using DNA segments [14]. This breakthrough has resulted in bacteria solving a classical problem in artificial intelligence – the Hamiltonian problem. Theoretical models, such as the application of bacteria to solve the "*burnt pancake problem*" [15] have also reinforced the notion that bacteria possess computing capabilities that extend beyond traditional electronic systems. Furthermore, it is possible to witness biocomputing in multiple dimensions such as at the gene, metabolic and population levels. Exemplifying gene-level computation, researchers created encoding devices by altering the parameters of genes [6]. In addition, computing capabilities in metabolic circuits are demonstrated in whole-cell and cell-free environments in [16] that signify metabolic

Samitha Somathilaka is with VistaMilk Research Centre, Walton Institute for Information and Communication Systems Science, South East Technological University, Waterford, X91 P20H, Ireland and School of Computing, University of Nebraska-Lincoln, 104 Schorr Center, 1100 T Street, Lincoln, NE, 68588-0150, USA. E-mail: ssomathilaka2@unl.edu.

S. Balasubramaniam is with School of Computing, University of Nebraska-Lincoln, 104 Schorr Center, 1100 T Street, Lincoln, NE, 68588-0150, USA. E-mail:sasi@unl.edu

Daniel P. Martins is with School of Computer Science and Electronic Engineering (CSEE), University of Essex Wivenhoe Park Colchester CO4 3SQ, UK. E-mail: daniel.martins@essex.ac.uk.

This This publication has emanated from research conducted with the financial support of National Science Foundation (NSF) under Grant Number 2316960.

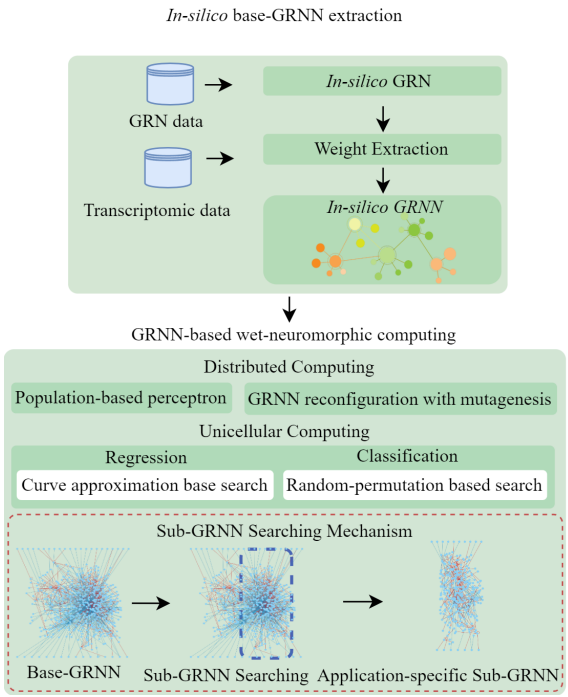


Fig. 1. Illustration of the framework associated with the novel concept of GRNN-based wet-neuromorphic computing. Our previous studies focused on revealing the concept of GRNN, the emergent property of population-based perceptron model and GRNN reconfiguration with mutagenesis. This study focuses on the components marked with the red-dashed box.

layer biocomputing, while [17] uses bacteria consortiums for pattern recognition, giving an example for population-level biocomputing. All these approaches are based on engineering the cells. However, the challenges associated with engineered bacteria limit their practical use for computing. To this day, it is burdensome to design large-scale genetic circuits without stressing the cell, negatively affecting the circuit dynamics and overall reliability [18]. Moreover, the cross-talks between signaling expression pathways also narrows down the possibility of complex genetic circuit design [19]. This is also impacted by the gene regulatory mechanism that competes for expression resources, which further creates unintended cross-talks [20]. Ensuring the long-term stability of engineered cells is also challenging under the altered gene expression pathways as the modifications may affect cell viability and growth rates [18]. This motivates us to explore the possibility of using the inherent computing functions within the cells through external chemical control, without modifying the cells' internal genetic system.

This study focuses on a specific aspect of a GRNN-based wet-neuromorphic computing framework that we introduce as illustrated in Fig. 1. In our previous work, a mechanism was introduced to quantify gene-gene interactions that returns a weighted directional network with nodes as genes and edges as expression interaction with the capability to process the incoming regulatory factors [21]. This network is analogous to a random structured Neural Network (NN); hence, we call this **Gene Regulatory Neural Network (GRNN)**. Further, previous work demonstrated the potential of using bacterial populations as single perceptrons, with a focus on Molecular

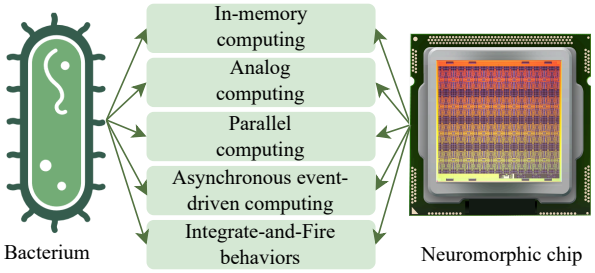


Fig. 2. Comparison of bacterial and neuromorphic computing architectures, emphasizing shared features like in-memory, analog, parallel, and asynchronous event-driven computing. This comparison lays the direction for bacterial computing as a novel wet-neuromorphic computing architecture.

Communication (MC) between GRNNs [21]. In contrast, this study examines the individual capabilities of these networks, exploring the feasibility of using non-engineered bacteria for general-purpose computing, including mathematical regression and classification tasks, as highlighted in the red dashed box in Fig. 1. Further, the current study emphasizes the necessity of searching sub-GRNN to perform application-specific computing and introduce a search mechanism. Finally, using *E. coli* as our model species, this study evaluates the feasibility of performing mathematical regression and classification tasks and elucidates the computing diversity within a GRNN of a single cell.

The core principles of neuromorphic systems, such as in-memory computing, analog computing, parallel processing, and asynchronous event-driven computing draw inspiration from brain computing architecture. While the literature demonstrates the potential to leverage inherent cellular properties to synthetically design neuromorphic architectures within bacterial cells [6], observing GRNN-based bacterial computing through a neuromorphic lens, we recognize in-memory computing in bacteria, where computational "weights" are embedded in the physical properties of gene interactions, including binding affinity, allosteric effects, and diffusion dynamics. Analog computing is reflected in the way gene expression levels respond to TF concentrations, analogous to continuous signals. Further, parallel processing in neuromorphic systems is mirrored in bacterial cells, where multiple genes are regulated simultaneously by various TFs. Additionally, asynchronous event-driven computing, a hallmark of neuromorphic systems, is evident in bacterial behavior where gene regulation and cellular responses occur in reaction to environmental changes without a global clock, ensuring efficient and timely processing. At each event, a spike in TF concentration occurs for the related gene, with its amplitude determined by the incoming signal's concentration, drawing parallels to spiking neural networks. The accumulation of TF spikes from source genes leading to expression in the target gene is analogous to the Integrate-and-Fire model in SNNs, with studies showing that the BReLU (ReLU) behavior observed in gene expression mathematically resembles both IF and Leaky Integrate-and-Fire models [22]. By leveraging these similarities as shown in Fig. 2, we propose GRNN-based bacterial computing systems as a novel wet-neuromorphic approach, utilizing the inherent biological computing properties of bacterial cells for efficient

computation. The contributions of this article are as follows.

- **Modeling the *E. coli* GRNN for *in-silico* experiments:** Our previous studies proved the existence of GRNN [21] using experimental gene expression data. Here, we extend this model by focusing on the GRNN of *E. coli* and explore their natural individual computing capabilities to solve computer science problems.
- **Introducing application-specific sub-GRNN search algorithm:** GRNNs are considered pre-trained NNs leading to significant differences in the application pipelines. Therefore, we introduce a search algorithm tailored to GRNNs that includes extractions of sub-GRNNs by mapping them to an application problem.
- **Feasibility analysis on performing computing of conventional computing tasks:** We evaluate the feasibility of performing regression (including linear, multiple variable linear, 2^{nd} and 3^{rd} degree polynomial regressions) and classification (including binary and multi-class) tasks using the extracted *E. coli* GRNN. Further, we solve a digit classification problem and evaluate its accuracy as a case study.

The remainder of the manuscript is structured as follows. Section II explores the background of GRN computing properties using the literature and, subsequently, the existence of GRNN. The next section describes the design of the *E. coli* GRNN using experimental transcriptomic data, presents a structural analysis of GRNN, and introduces the methodology applied for extracting application-specific sub-GRNN. Section IV and V present feasibility analyzes of performing regression and classification problems including a digit classification use-case problem. This is followed by discussion and conclusions in Section VI.

II. Background

We identified that the natural computation of a bacterium is manifested within the domain of gene regulation. The bacterial cell's genome serves as a repository of encoded information, and its dynamic regulation is through the transcription, translation, and post-translational modifications, which constitutes a complex computing network.

A. Gene as a Computing Unit

The gene expression process can consider intra/extra cellular information computing, resulting in functional gene products, such as proteins or non-coding RNAs [23]. Prokaryotic genes are often organized into operons (clusters of genes transcribed as a single mRNA molecule with a shared promoter), and transcription is regulated by specific DNA-binding proteins near the promoter, influencing RNA polymerase activity. The transcribed mRNA, complementary to the DNA template, undergoes translation into ribosomes, which are composed of rRNA and proteins. This tightly regulated and coordinated process is vital for the cell to execute specific functions and respond to environmental signals. The computing capabilities of genes have been investigated in recent decades, gradually paving the way toward the use of biological entities for computing. Since the pioneering work of DNA computing in 1994 [12], the field has been relentlessly advancing as an alternative to conventional computing methods. Among many diverse approaches, the use of genes as perceptrons [24] holds a special position due to the crucial role that Artificial Neural Networks (ANNs) play in today's computing world. An artificial neuron can accept multiple inputs that are then weighted and summed before passing through an activation function. Similarly, a gene can receive multiple TFs that lead to combinatorial regulation of the target gene [25]. Furthermore, the properties of TFs including the affinity of the TF binding site and mechanisms such as thermoregulators/enhancers/silencers [26], [27] and the lifetime of a DNA-TF complex [28] can determine the magnitude of the impact of the source TFs on the target gene resulting in regulation of expression level. In an ANN, the weighted summation of an artificial neuron is then passed through an activation function such as sigmoid, tanh, and ReLU, which introduces nonlinearity to the computing. Similar behavior can also be observed in genes, as they are considered switches that are in the "on" or "off" states determined by the regulatory influence of the TFs. Here, the combined influence of the TFs as a weighted summation is converted into the same two-state output ("on" or "off" states) for cell transcription [29], exhibiting sigmoid-like dynamics of gene expression.

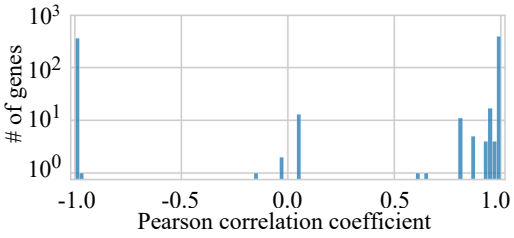


Fig. 3. Gene expression Pearson correlation coefficients between the source and target genes with only one inward and no outward edges. These results indicate the expression levels of the source-target gene pairs have linear relationships.

To observe gene expression behaviors beyond the sigmoidal properties, we performed a simple correlation analysis on temporal expression dynamics utilizing an *E. coli* dataset (accession number GSE65244) from the GEO database [30]. Here, we used 827 target genes with single inward and no outward edges to clearly observe gene correlation. Based on our analysis, 95.40% of the target genes have a correlation coefficient greater than 0.9, while 4.11% of the genes have a coefficient less than -0.9, demonstrating that there are strong linear relationships between the expressions of the source and target genes. Furthermore, only 0.49% of the expressions of the pair of source-target genes have correlation coefficients within ± 0.9. This analysis reveals that most of the source and target genes have linear relationships as shown in Fig. 3. This emphasizes that the relationships between the source and target genes can be converted to a single value (a.k.a weight) and the suitability of the ReLU activation function over a sigmoid to represent a gene's expression behavior in the time domain. However, there is a biophysical boundary for the maximum gene expression rate, which emphasizes the requirement of using a Bounded Rectify Linear (BReLU) activation function.

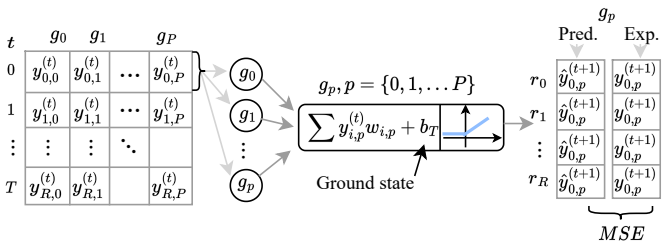


Fig. 4. Illustration of the gene-perceptron model of gene g_p, where the weighted summation of source gene (g_0, g_1,...,g_P) expression levels at time step t passes through the activation function BReLU and produces an expression level at time step $t+1$ corresponding to the input. This figure also elucidates the weight extraction mechanism, where we use temporal transcriptomic records to refine the weights between predicted and expressed genes, as well as the biases.

We prove this argument in Section III-A, where we use the accuracy of the extracted GRNN to show that the gene expression behavior follows a BReLU activation function. Moreover, a further investigation of gene expression properties explains that prokaryotic genes have "ground states" since RNA polymerase can access almost any promoter without the presence of activators or repressors [31]. Taking into account this property, we improve our previous weight extraction model by accompanying each gene perceptron with a bias that represents the ground state, as shown in Fig. 4.

B. Existence of GRNN

Gene interactions driven by functional proteins form a complex network within the GRN, involving operons, modules, and motifs for coordinated gene expression [32]. An operon is a set of co-regulated genes with a common promoter, producing a single mRNA molecule. A module is a group of operons regulated by the same TF or signal, while a motif is a recurring interaction pattern with specific functions, such as feedback loops [33]. GRNs process and transmit information through these structures, modulating downstream gene activity. Transcription patterns and GRN topologies drive bacterial decision-making, hinting at complex computing properties [34]. To reveal these properties, we introduced a framework for quantifying gene-gene interaction dynamics [21]. By perceiving the GRN as a random structured graph network, we quantified the influence of source genes on target genes using transcriptomic data. This weighted interaction GRN can be considered a gene regulation-based random structured NN, transforming it into a GRNN and reflecting the computational principles observed in ANNs.

III. Base-GRNN creation and Application-specific sub-GRNN Extraction

In this section, we detail a modified version of the GRN-to-GRNN conversion framework originally introduced in [21] which creates an *in-silico* version of the biological GRNN. Subsequently, we dive into the essential components of GRNNs, which include network structures as well as their analog and parallel computing capabilities; and present the network architectural search in the context of GRNN-based computing instead of the conventional NN learning stage as the GRNN is analogous to a pre-trained random structured NN.

A. GRN-to-GRNN Conversion

The GRN represents a unique gene-gene interaction network specific to each species. Publicly accessible GRN databases covering various species or strains, such as E. coli, can be found in [35]. Typically, these GRNs contain only data on the existence and type of interactions between static features such as genes, operons, and TFs (including Sigma Factors - SFs). In addition, both coding RNAs (cRNAs) and non-coding RNAs (ncRNAs) play critical regulatory roles in bacteria. In *E. coli*, small RNAs (sRNAs) that are typically less than 200 nucleotides, regulate gene expression by base pairing with target mRNAs to influence their stability or translation, often requiring chaperone proteins. Approximately 200 trans-acting sRNAs in *E. coli* can inhibit translation initiation by binding near the target's Shine-Dalgarno sequence, preventing ribosome binding and promoting RNA degradation. The regulatory impact of both cRNAs and ncRNAs acting as TFs, is determined by their diffusion properties and allosteric effects, which influence the interaction strength with the target genes. In addition, the concentration of these TFs serves as the input value for gene regulation, ultimately determining the magnitude of their regulatory effect [36]. The absence of quantitative properties, including the magnitude of the impact of one element on another, hinders the extraction of accurate natural computing capabilities of biological cells. To overcome such obstacles, we previously introduced a GRN-to-GRNN conversion method [21] to quantify the influence of TFs on the regulation of a target gene. The transcriptomic data used for weight extraction in this method inherently captures the combinatorial effects of various regulatory mechanisms, including the influences of non-coding and coding RNAs on TF activity. As a result, the interaction strengths derived from our framework reflect the integrated impact of these regulatory properties, including the indirect contributions of non-coding and coding RNAs on gene regulation through their modulation of TFs.

The previously proposed GRN-to-GRNN conversion method involves several stages: modeling the GRN as a graph network, dividing the GRN into gene-perceptrons, pre-processing transcriptomic data, and weight extraction [21]. Initially, the GRN is reconstructed as a directed graph where genes are nodes and their interactions are edges. This GRN is divided into sub-networks, each associated with a gene having at least one inward edge. These sub-networks, similar to single-layer perceptrons, contain a target gene and source genes regulating its expression. These are termed single-layer gene-perceptrons. Transcriptomic data, consisting of expression levels of both source and target genes in each gene-perceptron, is used to assess interaction strengths. Fig. 3 shows that gene expression has linear relationship between inputs and outputs. Additionally, the maximum biophysical expression rate suggests that gene-perceptron expressions have an upper bound. These factors indicate that gene expression exhibits BReLU properties,

and a mechanism similar to single-layer perceptron training quantifies interactions between source and target genes. Furthermore, as the prokaryotic genes have a ground state, identified as the bias of the gene-perceptron model, we improved the previously proposed GRN-to-GRNN conversion by embedding a bias in gene-perceptrons to extract gene-gene interaction dynamics. The gene-perceptron of this study is depicted in Fig. 4 and its functions are represented as,

$$\hat{y}_p^{t+1} = max\left(\sum_{i=1}^{P} y_{(i,p)}^t w_{(i,p)} + b_p, 0\right), \tag{1}$$

where $\hat{y}_p^{t+1}$ represents the predicted output of the target gene g_p at the next time step $t+1$. This prediction is influenced by the current outputs of other genes g_i at the same time step t, denoted by $y_{(i,p)}^t$. The interaction between gene g_i and the target gene g_p is quantified by the weight $w_{(i,p)}$, which determines the strength and nature (positive or negative) of this interaction. The term b_p represents the bias or baseline expression level of the target gene, independent of the interactions with other genes. In this model, the expression level of the target gene at the future time step $t+1$ is influenced not only by the direct interactions with other genes at the current time step t, but also by the inherent properties of the target gene itself (captured by the bias b_p). Initially, the weights $w_{(i,p)}$ are set to random values, reflecting the lack of prior knowledge about the interactions. These weights are then iteratively refined through a process of minimizing the Mean Squared Error (MSE), which is the difference between the predicted gene expression $\hat{y}_p^{t+1}$ (from the model) and the actual measured gene expression y_p^{t+1} (from experimental transcriptomic data). This iterative adjustment continues until the model's predictions closely match the observed data, thereby optimizing the gene-perceptron for accurate prediction of gene expression levels. Further information can be found in [21]. MSE is calculated as,

$$MSE(g_p) = \frac{1}{T}\sum_{t=0}^{T}(y_p^t - \hat{y}_p^t)^2 \tag{2}$$

where T is the last time step. This process is iterated for all the gene-perceptrons to extract the weights of all the interactions within the GRN.

B. E. coli Base-GRNN

We applied the GRN-to-GRNN conversion method explained in Section III-A to *E. coli* k-12 strain CSH50 to extract its base-GRNN and used it for all analyses of this study. In the first stage of conversion, the GRN data is obtained from [35] under multiple categories including TF - gene, TF - operon, TF - Transcription Units (TU), TF - TF, SF - Gene, SF - TU and sRNA - gene. By merging the interactions in each category, the complete GRN of *E. coli* is constructed as a directed graph network consisting of 3175 genes as nodes and 9678 interactions as edges using Python and the NetworkX library. Next, the GRN is divided into single-layered gene-perceptrons that contains corresponding source genes and initialized with random weights. For the weight/bias extraction phase, temporal transcriptomics data [30] (GEO accession number GSE65244) that contains interpolated expression records for 43 time steps are used after normalization. However, we only use 34 expression records, which is about 80% of the total records for the weight/bias extraction, while the rest of the transcription records are used to evaluate the accuracy of the extracted weights and biases. This step involves a custom Python script designed to identify each single-layered gene-perceptron and generate a corresponding randomly weighted shadow perceptron with the same structure. The transcriptomic data are then used to fine-tune the weights using equations (1) and (2). The learning rate and the number of epochs are set to 10^{-5} and 10^9, respectively. Further, we parallelized the iterative process of weights and biases extraction of all the gene-perceptrons using the CUDA platform on NVIDIA GeForce RTX 4090. This provides a weight matrix representing an *in silico* randomly structured NN that can mimic the computation based on the gene expression of the bacterial cell. The extracted weight matrix is denoted as

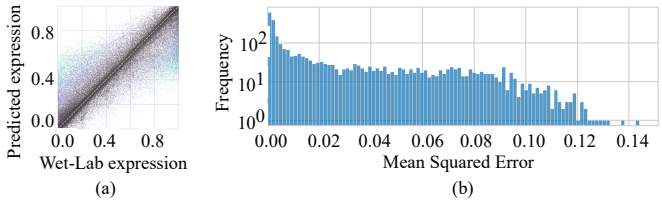


Fig. 5. Illustration of the accuracy of the extracted base-GRNN, where a) compares the predicted $\hat{y}$ and wet-lab experiment y expression levels for all the timesteps of all the gene-perceptrons and b) showes the MSE of each gene for all the timesteps.

$$\mathbf{W} = \begin{array}{c} \\ g_1 \\ g_2 \\ \vdots \\ g_P \end{array} \begin{array}{c} \begin{array}{cccc} g_1 & g_2 & \dots & g_P \end{array} \\ \begin{pmatrix} w_{(1,1)} & w_{(1,2)} & \dots & w_{(1,P)} \\ w_{(2,1)} & w_{(2,2)} & \dots & w_{(2,P)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{(P,1)} & w_{(P,2)} & \dots & w_{(P,P)} \end{pmatrix} \end{array}, \tag{3}$$

where $w_{(i,j)}$ is the weight of the interaction between i^{th} and j^{th} gene where $i:j = \{1,2,..,P\}$. The $w_{(i,j)}$ is the weight of self-regulation interaction when $i=j$. Next, we model the output of the GRNN, $\mathbf{O}^{(t+1)}$ at $t+1$ using weight $\mathbf{W}$ as,

$$\mathbf{O}^{(t+1)} = max(\mathbf{W}\cdot(\mathbf{I}^{(t)} + \widetilde{\mathbf{N}}) + \mathbf{B}), \tag{4}$$

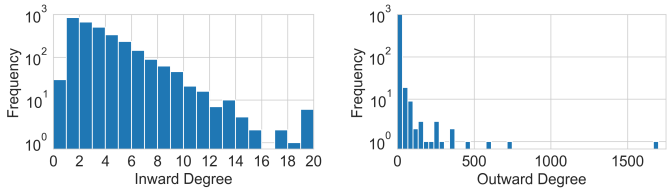
where $\mathbf{I}^t$ is the input matrix $\mathbf{B}$ is the bias matrix and $\widetilde{\mathbf{N}}$ is the Gaussian noise $\widetilde{\mathbf{N}} = N(0, 0.1)$ extracted based on the iterative experiments [30] (GEO accession number GSE215300). For the next time step, the input matrix $\mathbf{I}^{t+1} = \mathbf{O}^{t+1}$ and $\mathbf{O}^{t+2}$ is computed as,

$$\mathbf{O}^{(t+2)} = max(\mathbf{W}\cdot(\mathbf{I}^{(t+1)} + \widetilde{\mathbf{N}}) + \mathbf{B}). \tag{5}$$

The structural and algorithmic complexity behaviors of the *E. coli* base-GRNN are analyzed in the next section.

C. GRNN Structural and Algorithmic Complexity

The base-GRNN consists of a graph topology with a power-law distribution that contains a few central nodes and a



(a) (b)

Fig. 6. Network degree distribution of the *E. coli* GRNN, where a) and b) show inward and outward degree frequency, respectively.

significant number of terminal nodes, as evident in Fig. 6. Within the *E. coli* GRNN, 68.45% of the gene-perceptrons have more than one inward edge (Fig. 6a) with the ability to compute multiple inputs together. Moreover, the distribution of the outward edges as shown in Fig. 6b proves the existence of central gene-perceptrons that can influence around 92.12% of the terminal nodes. For example, the gene *b3067* has a total of 1703 outward edges, of which 91% are terminal nodes. Activation of this particular gene-perceptron in turn results in a wide range of expression values in the terminal nodes, which contributes to making the base-GRNN suitable for a range of problems. Therefore, this power-law distribution positively reflects computing diversity, allowing the base-GRNN to be recognized as an extensive repository of diverse pre-trained sub-GRNNs. Due to the presence of loops in gene expression pathways, GRNNs can be considered NNs with a theoretically unlimited number of layers, allowing complex, recurrent interactions and regulatory feedback.

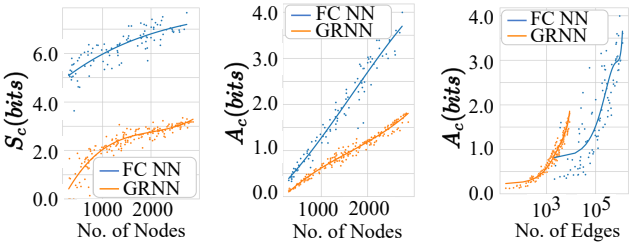
An important factor of NN is its structural and algorithmic complexities; therefore, we analyze and compare the structural and algorithmic complexities of GRNN with conventional NNs to determine its computing performance. The structural entropy of a graph neural network indicates its capacity to process complex graph-structured data by measuring the randomness and diversity in its topology, facilitating effective information integration. Kolmogorov complexity reflects the simplicity or complexity of the graph neural network's representation, with lower complexity implying more efficient learning and higher complexity indicating greater expressive power. Together, these metrics help assess the computational capabilities of a graph neural network in terms of efficiency, expressiveness, and generalization potential.

Calculation of structural entropy S_c begins by computing the betweenness centrality of all the gene-perceptrons, where we denote the betweenness of the i^{th} gene-perceptron $l(i)$ as follows

$$l(i) = \sum_{1 \leqslant i \leqslant N, s \neq i \neq t} \frac{\sigma_{s,p}(i)}{\sigma_{s,p}} \tag{6}$$

where $\sigma_{s,p}$ is the total number of shortest paths between the source g_s and target g_p genes, while $\sigma_{s,p}(i)$ is the number of shortest paths through gene g_i between the source gene g_s and target gene g_p. Further, we define the relative degree p_i of the i^{th} node as,

$$p_i = \frac{Degree(i)}{\sum_{i=1}^{N} Degree(i)}, \tag{7}$$



(a) (b) (c)

Fig. 7. A comparison of structural (S_c) and algorithmic (A_c) complexity behaviors between Fully Connected Neural Networks (FCNNs) and GRNNs. a) and b) compare the S_c and A_c variations against the number of nodes in the two types of NNs, while c) focuses on A_c against the number of edges.

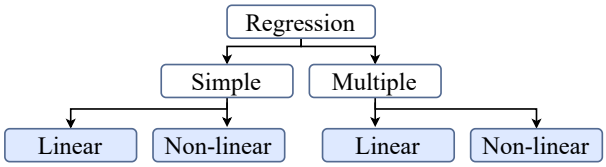


Fig. 8. Illustration of sub-categories of regression problems.

and q_i as the nonextensive parameter which is defined based on the betweenness $l(i)$ as follows,

$$q_i = 1 + (l(max) - l(i)), \tag{8}$$

where $l(max) = max[l(i), (i = 1, 2, 3, ..., N)]$. Finally, structural complexity S_c is calculated as follows

$$S_c = \sum_{i=1}^{N} \left(\frac{p_i^{q_i}}{\sum_{i=1}^{n} p_i^{q_i}} \right) log \left(\frac{p_i^{q_i}}{\sum_{i=1}^{n} p_i^{q_i}} \right). \tag{9}$$

This study uses the Kolmogorov complexity (K-complexity) approximated by the Coding Theorem Method (CTM) to determine the algorithmic complexity A_c, which is considered the basis for the network complexity [37]. CTM is calculated based on the Laplacian matrix L, and due to the large dimensions, we also employ the Block Decomposition Method (BDM) as follows

$$A_c \approx BDM(L) = \sum_{i=1}^{P} CTM(b_i) + log_2|b_i|, \tag{10}$$

where b_i is the i^{th} row of L and more information on estimating the CTM of b_i can be found in [38]. Fig. 7a compares the behavior of the structural complexity with respect to the number of nodes in a fully connected NN versus GRNN. It is evident that the power-law properties in the random structured GRNNs compared to fully connected NN result in lower structural complexity. We also found lower algorithmic complexity in GRNN structures which is shown in Fig. 7b due to a minimized number of edges compared to a fully connected NN with a similar number of nodes. Moreover, GRNNs comprising edges ranging from 2000 to 10000 exhibit greater algorithmic complexity compared to fully connected NNs as depicted in Fig. 7c. This reveals certain GRNN structures are capable of complex computing, while

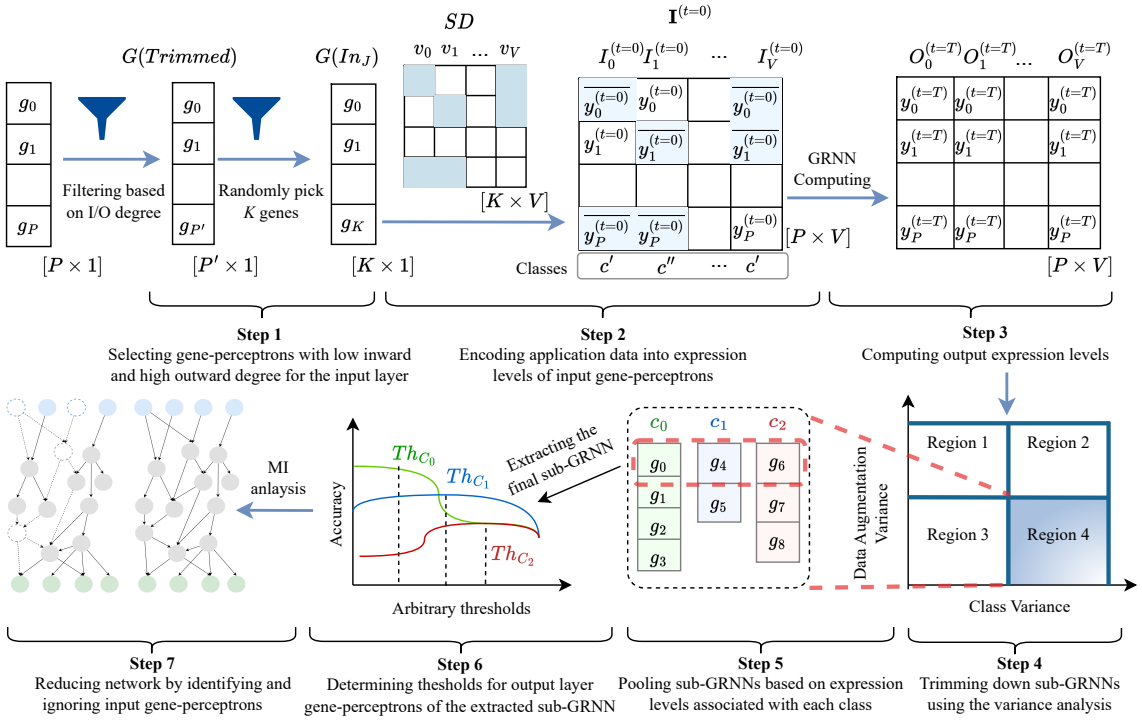


Fig. 9. Illustration of the proposed application-specific sub-GRNN search algorithm for One-vs-All classification. Step 1 focuses on selecting a set of input gene-perceptrons, $G(Trimmed)$ based on their inward/outward degree distributions, and a subset, $G(In_j)$ with K number of gene-perceptrons input features is selected from $G(Trimmed)$. In Step 2, the searching dataset SD is encoded into the expression level-based input matrix $I^{(t=0)}$. The output matrix, $O^{(t=T)}$, corresponding to $I^{(t=0)}$, is then calculated using the *in-silico* base-GRNN model as explained in Section III-B. In Step 4, a set of gene-perceptrons is identified, exhibiting higher expression variance between classes and lower expression variance within the same class. This set of gene-perceptrons is then pooled under each class in Step 5 based on their expression levels. Step 6 searches for the optimal expression thresholds for each class by maximizing accuracy. Finally, Step 7 conducts an Mutual Information (MI) analysis to identify the insignificant input gene-perceptrons and removes them from the input layer to reduce the size of the network.

maintaining an improved interpretability compared to fully connected GRNNs.

D. Application-specific sub-GRNN Search Algorithm for Regression

The weight configuration of an extracted base-GRNN depends solely on the state of the bacterial cell and the environmental conditions. Thus, the weight extraction process creates an *in-silico* twin of the cell's base-GRNN, which can be viewed as an application-agnostic, pre-trained, randomly structured physical neural network or a weight-defined neuromorphic system. To use a bacterial cell with such a GRNN for problem-solving, it is necessary to search for and extract the precise sub-GRNN from the *in-silico* version of the base-GRNN.

Mathematical regression has been widely used in data mining applications [39], [40] for many years. Therefore, as one of the key contributions of this study, we introduce and detail the extraction mechanism of the sub-GRNN that can approximate regression functions of the gene-perceptron expression profiles. In order to showcase the regression diversity illustrated in Fig. 8, we first stimulate a randomly selected gene as input and observe the resulting expression patterns of the other genes in the GRNN. Subsequently, each expression pattern is subjected to a curve approximation to determine its mathematical form. For the extracted expression levels of all the gene-perceptrons, the r^2 score ($r^2 = RSS/TSS$, where RSS is the residual sum of squares and TSS is the total sum of squares) is calculated to measure the goodness of fitness of the output gene-perceptron expression with a regression approximations. This method is used for all types of regressions. Further, in this case of multiple polynomial regression, the goodness of the curve is fitted to the following equation,

$$f(x_1, x_2) = d_1x_1^2 + d_2x_2^2 + d_3x_1x_2 + d_4x_1 + d_5x_2 + d_6, \quad (11)$$

where d_1 to d_6 are coefficients of the approximated curve for each output gene-perceptron and x_1 and x_2. When applying GRNN for regression tasks, users can select the gene whose expression pattern exhibits the desired mathematical curve for the computational task using the *in silico* GRNN. Results for the regression diversity are presented in Section IV.

E. Application-specific sub-GRNN Search Algorithm for Classification

Our next main contribution in this study is the proposal of an application-specific sub-GRNN search algorithm for classification tasks, illustrated in Fig. 9, that is executed *in silico*. The conventional approach for classification using NNs involves creating a suitable NN architecture and training the weights using a training dataset, denoted as $SD_{K\times V}$ where K is the number of input features and V is the number records with corresponding class labels. In contrast, our algorithm is based on random permutation search to select sub-GRNNs

from the extracted base-GRNN. Upon stimulation of the input gene perceptron in the *in silico* base-GRNN model with input data from the labeled dataset ($SD_{K\times V}$), the gene expression cascades through the cell mimicking the dynamics of the actual cell. This results in the computation of the output as the expression levels of the output genes. Next, the model evaluates their accuracy using $SD_{K\times V}$. This process is repeated iteratively until a sub-GRNN with higher accuracy is identified. The identified sub-GRNN reveals the relationships between input and output gene-perceptrons. Next, the input data for specific applications should be converted into the TF concentration domain for computing tasks. This involves mapping input feature value ranges to corresponding input layer gene-perceptrons TF ranges based on the transcriptomic database used for the weight extraction. While the study does not emphasize wet-lab experiments, it lays the groundwork for wet-neuromorphic computing by harnessing the computational diversity of GRNN. The following section provides an in-depth explanation of this algorithm.

In the initial step, suitable candidates for the input layer are first filtered using the characteristics of the genes, including inward/outward degree as shown in Fig. 9 (Step 1). These characteristics include gene-perceptrons with an inward degree closer to zero, which are not significantly influenced by unnecessary incoming signals except for the problem-specific inputs and input gene-perceptrons with a higher outward degree so they have variety of output combinations that can support complex computing capabilities. This stage uses a theoretical graph degree distribution to create a set of gene-perceptrons for the input layer, $G(Trimmed)$. Here, $G(Trimmed)$ contains the P' number of genes, where the selected genes, P', is less than the total number of genes, P, of GRNN, respectively. Given that $G(Trimmed)$ contains P' number of gene-perceptrons and the problem has K number of features, there are ${}_{P'}P_K$ different input layers that can be extracted (where ${}_{P'}P_K = {}^{P'!}/_{(P'-K)!}$). Due to the massive number of sub-GRNNs, a heuristic search algorithm may be more efficient. However, exploring such algorithms is beyond the scope of this study and we use only this random permutation-based algorithm. Finally (Fig. 9-Step 1), the algorithm randomly picks a set of K number of inputs denoted as $G(In_J)$, for the J^{th} permutation, where $J = \{0, 1, 2, ..., {}_{P'}P_K\}$, where K is number of input features of the problem.

Before SD is encoded into the expression levels in Fig. 9 (Step 2), a base-TF array is created using the expression levels at the zero time step of the transcriptomic data used for weight extraction [30] (GEO accession number GSE65244). This step is crucial to mimic the base behavior of the cell at $t = 0$ (starting time of the computing process), where the cell functions with respect to environmental conditions. The base-TF array is then sufficiently altered to encode the inputs of SD to create the input matrix $\mathbf{I}^{(t=0)}$ that contains the TF input arrays $I_v^{(t=0)}$, where $v = \{0, 1, 2, ..., V\}$. In this stage, if SD is considered digital, then state "1" represents the highest expression level of the corresponding gene, while state "0" represents the lowest value. However, if SD is in analogue form, the values are normalized and mapped with the concentrations based on the highest and lowest expression levels of the relevant gene. After decoding all input records with the expression levels in Step 2, using the mathematical model explained in (4) and (5), the output expression levels are computed in Step 3. This step produces an expression matrix, $O^{(t=T)}$, with output arrays, $O_v^{(t=T)}$, where $v = \{0, 1, 2, ..., V\}$ corresponds to each class.

In Step 4, we conduct a variance analysis to identify the genes-perceptron that can be used to represent each class at the output layer of the sub-GRNN. Suppose a gene-perceptron can express in a higher level for the corresponding input $I^{(t=0)}$ of a particular class, c_i, while maintaining low variance between augmentations in the same class and higher variance between different classes. In this case, that gene-perceptron is a good candidate to represent c_i. Hence, we search for gene-perceptrons for all the classes in "Region 4" (as shown in Fig. 9- Step 4), where the variance between classes is high and the variance between records of the same class is low, which we will then form a set of output gene-perceptrons.

In Step 5, if a gene-perceptron g_i from the above set satisfies the condition, $\overline{y}(g_i, c_l) > \overline{y}(g_i, c_m) : m\forall; m < |c|, m \neq l$, where $\overline{y}(g_i, c_l)$ is the mean expression level for the class c_l, then g_i is pooled under the class c_l. This process is repeated for all the gene-perceptrons in "Region 4". Following this, the gene-perceptrons that has the highest mean expression level a particular class and the lowest gene-perceptrons for other classes are selected to represent each class.

Step 6 of the algorithm is dedicated to identifying the threshold for each gene perceptron using an accuracy-maximizing approach. First, we get the *true-positives* (TP), *true-negatives* (TN), *false-positives* (FP) and *false-negatives* (FN) for each class using an arbitrary threshold value, $Th = a$. The accuracy of the classification is then calculated for class c_l, which is denoted as $ACC(c_l, Th = a)$ and represented as follows

$$Acc(c_l, Th = a) = \frac{TP + TN}{TP + TN + FN + FP}. \tag{12}$$

This calculation is repeated with various thresholds a ranging from zero to one with increments of 0.05 and the threshold for the class c_l is calculated as

$$Th = \arg\max_a Acc(c_l, Th = a). \tag{13}$$

Similarly, the thresholds for all the classes associated with the problem are extracted iteratively. This process is repeated a number of times ($< {}_{P'}P_K$), resulting in an accuracy-based ranking to support the selection of the best sub-GRNN.

After successfully selecting a suitable application-specific sub-GRNN, a perturbation-based MI analysis is performed with the objective of optimizing the network in Step 7. In this step, all inputs of the extracted sub-GRNN are subjected to signals fluctuating from zero to one, and the outputs of the network are recorded from the gene-perceptrons at the output layer. Since both inputs and outputs for this network are continuous variables, the MI between the input and output nodes is denoted as,

$$I(g_x; g_y) = f(x, y) \cdot \log\left(\frac{f(x, y)}{f(x) \cdot f(y)}\right) dxdy, \tag{14}$$

TABLE I
PARAMETERS UTILIZED FOR THE *in silico* FEASIBILITY ANALYSIS FOR REGRESSION APPLICATIONS

Parameter	Value
Input genes	*b3067* (for simple regressions), *b3067, b3357* (for multiple regressions)
Input range	0 to 0.5 (normalized concentration units)

where g_x and g_y are the input and output nodes, respectively; and $f(x, y)$ is the joint probability density function of g_x and g_y expressions. These MI values indicate the amount of information flow from input nodes to output nodes. Input nodes associated with lower MI values can be ignored, resulting in a reduced network. This reduction allows the network to compute with fewer inputs, making it more efficient and easier to use.

IV. GRNN APPLICATION IN REGRESSION

In this section, we elucidate the diversity of the search space for regression problem types, as illustrated in Fig. 8 using the method explained in Section III-D. This analysis is conducted using the *E. coli* GRNN as a use case.

A. Linear Regression Analysis

First, we identified *b3067* as the *E. coli* input gene-perceptron that has the highest outward degree and can regulate other 1703 connected gene-perceptrons. This is important for this analysis as the *b3067* stimulation cascades through a significant portion of the GRNN, leading to various regression outputs. Next, the input gene-perceptron is stimulated with 25 concentration input values ranging from 0 to 0.5 normalized concentration units (as mentioned in Table I). The initial expression values of the rest of the gene-perceptrons are kept at the minimum level based on the expression profiles in [30] (GEO accession number GSE65244). Each step of this experiment is also iterated 10 times to observe more accurate behaviors.

The variety of linear regression slopes with respect to r^2 fitness, where the gene-perceptrons with the highest r^2 values tend to have a variety of coefficients for different slopes (see Fig. 10a), reveals that the highest positive slope is estimated as 0.36 while -0.50 is the largest negative slope for *E. coli* GRNN when the input is *b3067*. Fig. 10b presents three output gene-perceptrons for different regression lines, where *b1380* and *b3293* have positive slopes of 0.29 and 0.12, respectively; while *b4435* has a negative regression slope of -0.29. Fig. 10c illustrates the sub-GRNNs associated with the three output gene-perceptrons, *b4435, b1380* and *b3293* with corresponding color codes to Fig. 10b. According to Fig. 10c, it is essential to highlight that all three linear regressions are performed in parallel, proving the parallel computing properties of the GRNN. This analysis evidently elucidates the availability of a diverse linear regression solution space, where an algorithm can search and map gene-perceptrons to applications.

B. Quadratic Polynomial Regression

The GRNN output generated in the previous section is also utilized for the matching to quadratic polynomial regressions. Fig. 11a represents the behavior of quadratic (Coef. 1) and linear (Coef. 2) coefficients for each gene-perceptron that is color-coded according to the RSS value, where the lighter color (yellow) indicates the higher goodness of fit. The quadratic coefficients of the curves with the highest RSS values range from -2 to 2, while the linear coefficient ranges from -1 to 0.5. Fig. 11b shows three example curves to emphasize the diversity of the available quadratic regression within the *E. coli* GRNN given the input gene-perceptron *b3067*. Three quadratic curves shown in Fig. 11b are for the gene *b0124*, *b2487* and *b3751* where the quadratic coefficients are -0.83, -0.63 and 1.48, respectively. Figure 11c shows the sub-GRNNs corresponding to the three output gene-perceptrons, namely *b0124*, *b2487*, and *b3751*, each represented with distinctive color codes, aligning with Fig. 11b. This shows that with the same input gene-pereceptron *b3067*,

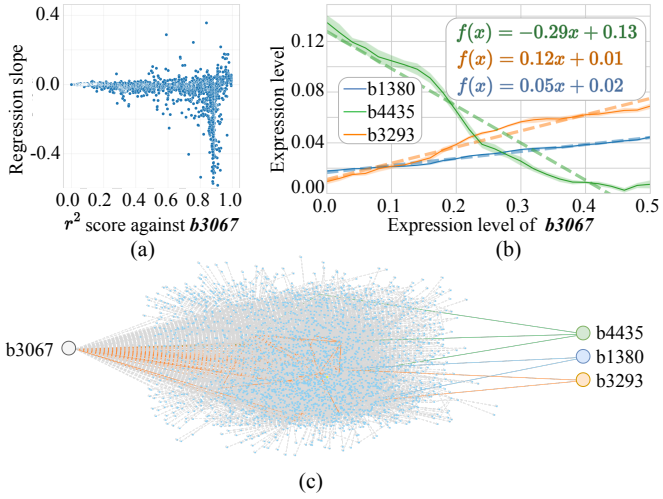


Fig. 10. Illustration of simple linear regression using *E.coli* GRNN, where a) shows the regression slope distribution of all the genes against the respective r^2 score, b) exemplifies three regression lines based on three output gene-perceptrons and c) is the sub-GRNN for the linear regressions.

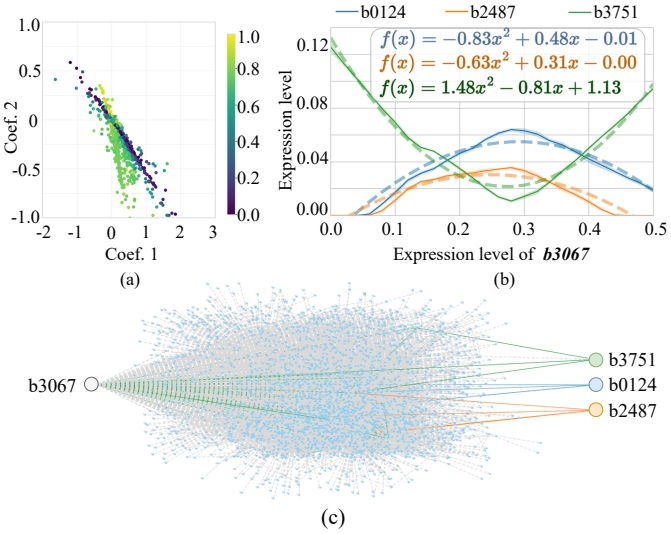


Fig. 11. Illustration of non-linear quadratic regression using *E.coli* GRNN, where a) shows the quadratic and linear coefficient distribution of all the genes that are color-coded to the RSS value, b) shows three example regression curves and c) is the sub-GRNN associated with the three example regression curves.

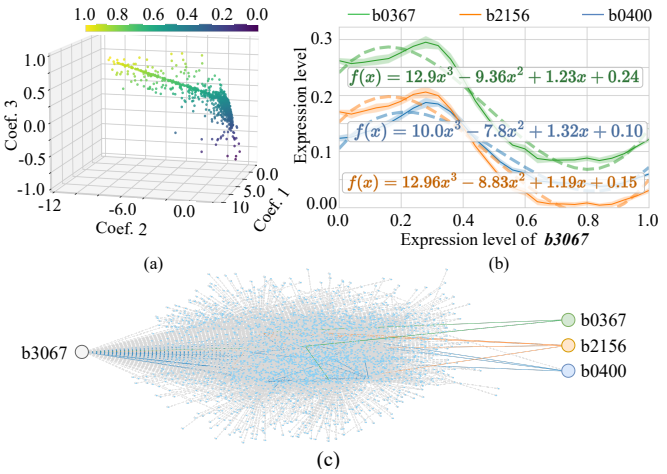


Fig. 12. Illustration of non-linear cubic regression using *E.coli* GRNN, where a) shows the cubic, quadratic and linear coefficient distribution of all the genes that are color-coded to the RSS value, b) shows three example cubic regression curves and c) illustrates the extracted sub-GRNNs of the three cubic regression curves.

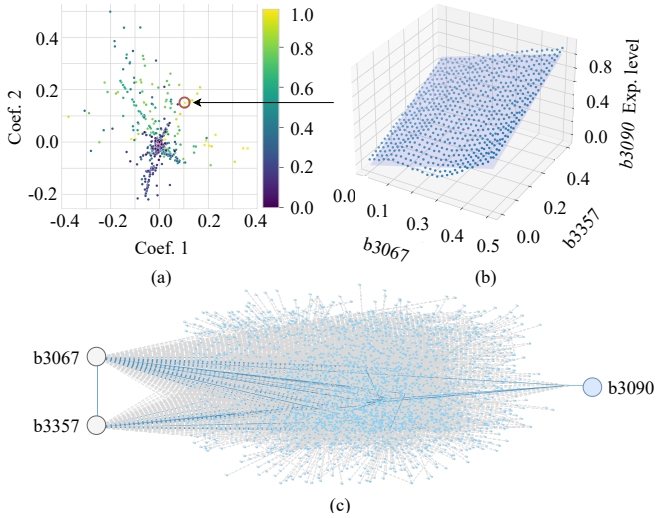


Fig. 13. Illustration of multiple-linear regression using *E.coli* GRNN by using gene-perceptrons *b3067* and *b3357* as two inputs, where a) shows the first and second coefficients distribution of all the genes that are color-coded to the *RSS* value, b) and c) shows the example plane of the output gene-perceptron *b3090* and the corresponding sub-GRNN, respectively.

we can switch from linear to quadratic polynomial regression by finding a different output gene-perceptron combination.

C. Cubic Polynomial Regression

The same data set used in Sections IV-A and IV-B is used to discover and match to the cubic polynomial regression of *E. coli* GRNN. Fig. 12a shows the coefficients of the cubic polynomials, Fig. 12b provides three example curves, while Fig. 12c illustrates the corresponding three sub-GRNNs. The cubic coefficient with $RSS > 0.7$ ranges approximately from 0 to 13, while the quadratic and linear coefficients range from -11 to 2 and -0.75 to 0.75, respectively. The ranges of the data points for the curves in Fig. 12 are not spread out, which means there is a minimized variation in the higher-degree polynomial regressions. This can be perceived as a limitation in the discovery of higher-degree functions.

However, it is important to note that these solution spaces are derived using the input gene-perceptron *b3067* as an example. Selecting a different gene could yield an entirely different solution space, further showcasing the diversity and adaptability of the computing capabilities. Conversely, selecting a gene with fewer outward edges may restrict information flow through the GRNN, leaving portions of the network underutilized. This, in turn, could result in a smaller and less dynamic solution space.

D. Multiple Linear Regression

We further investigate the feasibility of multiple regression of *E.coli* GRNN by using two input gene-perceptrons, *b3067* and *b3357* with outward degrees of 1703 and 596, respectively. Similarly to the previous setup, the two inputs are stimulated with expression levels from 0 to 0.5 with increments of 0.02 (a total of 625 input setups). In this analysis, RSS is considered the measure of variance of the regression model. Fig. 13a shows the coefficient variation, where Coef. 1 and Coef. 2 are associated with the two input gene-perceptrons, respectively. Planes with $RSS > 0.7$ have Coef. 1 ranging from -0.2 to $+0.4$ and Coef. 2 ranging from -1 to 1. Fig. 13b exemplifies the plane for the output *b3090* with the first and second coefficients of 0.10 and 0.14, respectively. Fig. 13c shows the sub-GRNN where the input layer consists of *b3067* and *b3357* and output layer with *b3090* gene-perceptrons. This example only considers the gene-perceptrons *b3067* and *b3357* as the inputs, and it is possible to explore diverse efficient spaces by selecting different combinations of inputs and output gene-perceptron.

E. Multiple Polynomial Regression

Results in Fig. 14 depict the possibility of extracting complex higher-order multivariate polynomial regression models that match our problem. The d_1 and d_2 in Fig. 14a are quadratic coefficients associated with the two inputs that govern the curvature of the model. The positive values of d_1 and d_2 result in curvature along the *b3067* and *b3357* concentration axes, respectively. It is evident that the distributions d_1 and d_2 exhibit distinct trends, characterized by positive and negative skewness, within approximate ranges of -10 to $+10$ and -3 to $+3$, respectively. Furthermore, d_3 is the cross-term coefficient that determines how the two inputs, *b3067* and *b3357* combine to affect the shape of the curve. Given that both inputs are consistently positive, the negative skewness of the d_3 distribution emphasizes that most of the resulting curves are shifted upward due to the combined influence of *b3067* and *b3357*. Linear terms d_4 and d_5 have an impact on the vertical position of the curve based on individual inputs. The analysis of the graphs in Fig. 14a reveals that *b3067* exerts a balanced effect on the curve shift, while *b3357* primarily tends to shift the curve downwards. Lastly, d_6 represents the y-intercept or the offset of the curve, which determines the vertical positioning of the curve. In particular, the distribution of d_6 is fairly symmetric around zero. Fig. 14b and Fig. 14c elucidate two significantly different multivariate polynomial

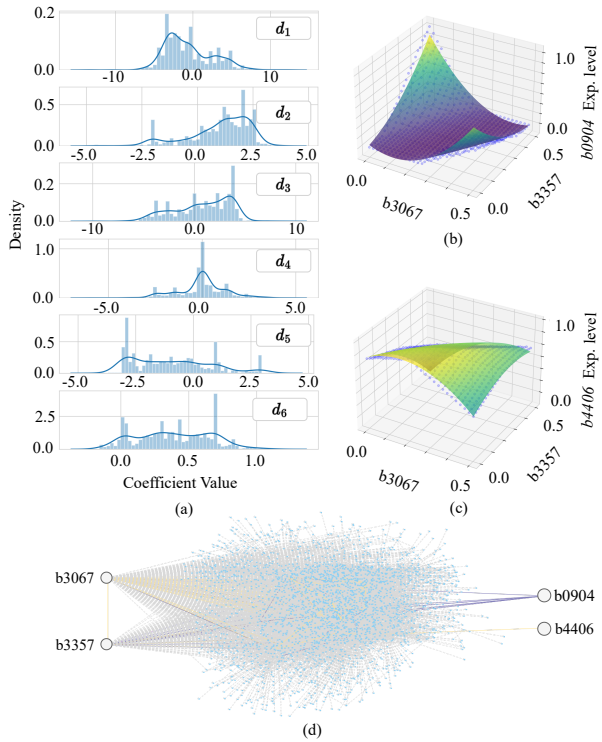


Fig. 14. Illustration of multiple non-linear regression using *E.coli* GRNN by using gene-perceptrons *b3067* and *b3357* as two inputs, where a) shows the distributions of each coefficient associated with (11) while b) and c) exemplify two curves with positive and negative coef. 1, respectively. Subsequently, d) shows the sub-GRNNs for the two examples regressions shown in b) and c).

regression examples, using gene-perceptrons *b0904* and *b4406* and this is based on the sub-GRNN presented in Fig. 14c.

V. GRNN Application in Classification

In addition to regression applications, NNs are also well known for classification tasks [41]. In this section, we explore the application of GRNN for three types of classification tasks: One-vs-All, One-vs-One, and Multiclass, as depicted in Fig. 15. These tasks necessitate more advanced and specialized layers compared to the simpler architecture used for curve approximation in regression, as discussed in Section IV.

A. BReLU as the activation function for GRNN-based classification

Gene-perceptrons can exhibit BReLU activation function behaviors (see Section III-A), which can be beneficial for GRNN-based classification. The sigmoid-like activation function produces positive outputs for negative inputs, and this can lead to noisy computing in NNs. However, this noisy behavior cannot be observed in NNs, when ReLU (BReLU) activation functions are utilized for classification tasks [42] as evident in Fig. 16a. ReLU including (BReLU) further encourages activation sparsity in computing by zeroing out negative values [43] and it further contributes to pruned networks for better computing efficiency, which we cover in this section using mutual information analysis.

Another advantage of BReLU is the increased sensitivity in classification [44]. As shown in Fig. 16b, where the upper

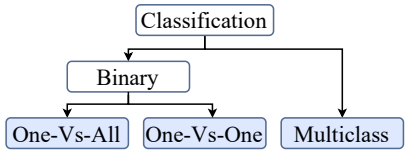


Fig. 15. Illustration of sub-categories of classification problems.

bound of BReLU is equal to that of the sigmoid function, BReLU is more sensitive within the input range of zero to one compared to the sigmoid activation function. Although the variation of the sigmoid activation function is limited to approximately 0.23 within the non-negative region ($[0, +1]$), the BReLU exhibits an unitary variation. The increased sensitivity in BReLU paves the way for multiclass classification using a single output gene-perceptron. This property is presented in Fig. 16b, where different threshold values (Th_1, Th_2 and Th_3 are set to 0.2, 0.5 and 0.8, respectively) are spaced to enable greater sensitivity. As gene expression values are modeled by the BReLU activation function, the thresholds represent varying expression levels corresponding to the input. In the multiclass classification application, the number of thresholds is determined by the number of classes and can be expressed as $|Th| = |c| - 1$, where $|Th|$ and $|c|$ are the number of thresholds and classes, respectively. Due to this continuous output of the gene-perceptrons, two types of thresholds can be further applied: $Th > 0$ for one class versus all the other classes (One-vs-All) and one class versus another class (One-vs-One); and $Th = 0$ for One-Vs-All. However, determining the threshold in the case of $Th > 0$ requires additional processes, as discussed in Section III-E and elucidated in Step 6 of Fig. 9.

Fig. 16c shows multiple expression levels corresponding to different classes (C_0, C_1, C_2 and C_3) where the expected expression level for the class c_0, $\overline{E}_{(c_0)}$ is always higher compared to other classes. Therefore, such an expression pattern of a gene-perceptron deems it suitable for One-vs-All classification output. However, determining the appropriate threshold ($Th_{(c_0)}$) requires additional steps. If a gene-perceptron is expressed in two distinguish levels, $\overline{E}_{(c_0)}$ and $\overline{E}_{(c_1)}$ as shown in Fig. 16c, it can be utilized for One-vs-One classification output with the threshold $Th_{(c_0,c_1)}$. Similarly, a gene-perceptron capable of expressing in three fixed levels ($\overline{E}_{(c_0)}$, $\overline{E}_{(c_1)}$, and $\overline{E}_{(c_2)}$; see Fig. 16e) in response to various inputs is suitable to represent the output layer node for multiclass classification applications. It is essential to emphasize that, in such situations, two thresholds ($Th_{(c_0,c_1)}$, $Th_{(c_1,c_2)}$) should be determined. The feasibility of binary and multiclass classification is proven with *in silico* experimental results in the next section.

B. Binary and Multi-Class Classification

We discuss the feasibility of gene-perceptron performing a binary classification under two subcategories, One-Vs-One (Fig. 16c) and One-Vs-All (Fig. 16d). Initially, we established a transcriptomic-level experimental setup using the *E. coli* GRNN, where we use 16 randomly selected gene perceptrons as inputs. Table II further provides detailed information on the parameters used for this analysis.

TABLE II
PARAMETERS UTILIZED FOR THE *in silico* FEASIBILITY ANALYSIS FOR CLASSIFICATION APPLICATIONS

Parameter	**Value**
Input gene	*b2664*, *b0080*, *b3060*, *b2697*, *b2220*, *b3423*, *b3743*, *b0345*, *b3481*, *b4401*, *b0357*, *b0889*, *b0817*, *b2217*, *b3905*, *b3071*
Input range	0 to 0.5 (normalized concentration units)
Iterations	10 per each input
# of classes	5
# of augmentations	10 (per class)

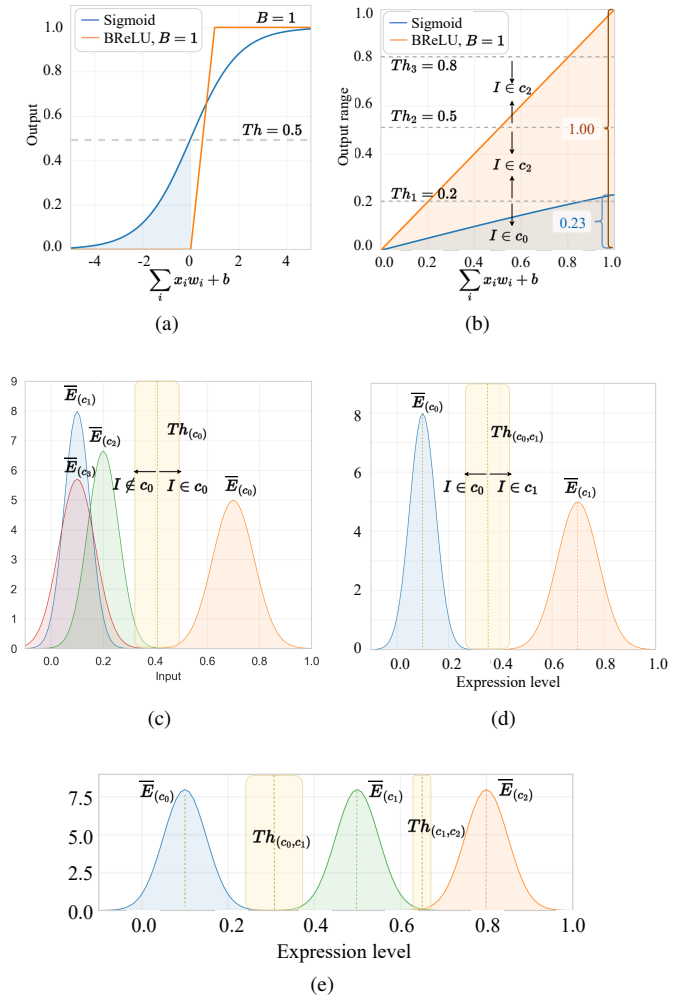


Fig. 16. Comparison of the sigmoid and inherent BReLU properties in the GRNN, where, a) highlights the shaded area in light blue that represents the region, where the sigmoid activation function outputs a positive value for negative weighted summation of the perceptron, b) compares the Sigmoid Vs BReLU output variation within the weighted summation range $[0, 1]$ and the possibility of having multiple thresholds (Th_1, Th_2 and Th_3) leading to multi-class classifications (c_0, c_1, and c_2), c) example gene expression distribution for One-vs-All classification where $Th > 0$, d) gene expression distribution for One-Vs-One binary classification, and e) gene expression distribution with multi-class classification possibilities.

For this specific network, the input layer of the GRNN is introduced with five different TF arrays (created as described in Section III-E) associated with five classes $c_i, i = \{0, 1, 2, 3, 4\}$. To capture the stochastic behavior within a cell, each simulation setup is iterated 10 times. A lower number of iterations may fail to accurately reflect these dynamics, while increasing the iterations further may not provide additional insights, as

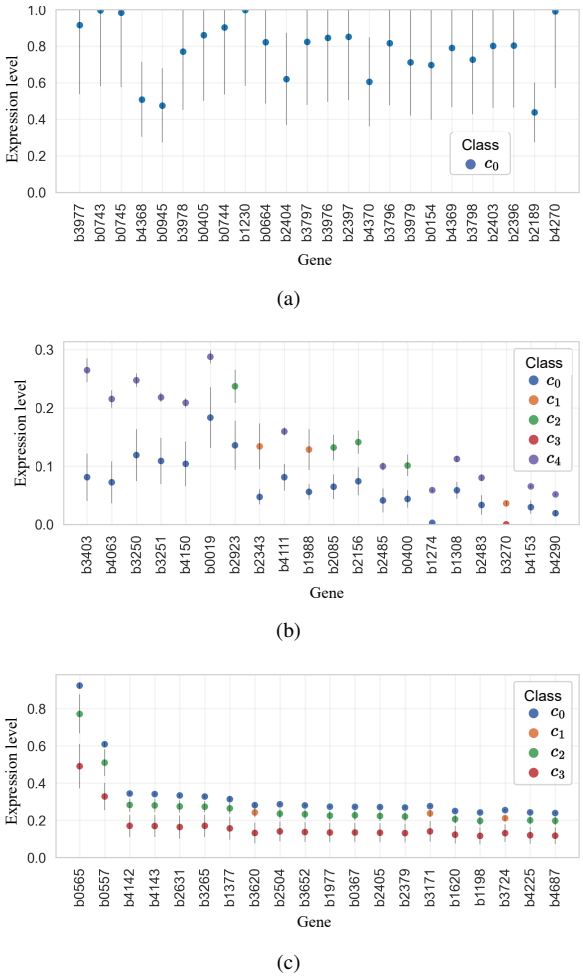


Fig. 17. Classification using the GRNN under three methods, where a) shows the One-vs-All, b) shows the One-vs-One, where the aim is to show different expression levels can be achieved for each gene indicating it can be used to classify more than one class and c) illustrates the multi-class classification, where we can see certain genes have separations that are high to support classification of up to three classes.

the system's behavioral variance remains consistent. Next, the expression levels of each gene-perceptron are recorded and filtered using the search algorithm proposed in Section III-E to identify the suitable gene-perceptron perform three subcategories of classification, one-vs-all, one-vs-one and multi-class.

For the one-vs-all subcategory of binary classifications that uses the threshold as $Th = 0$, the algorithm seeks genes-perceptrons that express for one class with a minimized variance, while the expression levels for the other classes remain equal to zero. Fig. 17a shows results for one-vs-all classification for class c_0 where gene-perceptrons on the x-axis are an example set of suitable candidates that can be considered output nodes. Selecting one of the gene-perceptrons on the x-axis as the output node, it is possible to classify inputs into class c_0, proving the possibility of using the GRNN for One-vs-All classification tasks. Moreover, the One-vs-All method can be used for multiclass classification by selecting gene-perceprons that are suitable for other classes.

Fig. 17b shows the feasibility of using GRNN for One-vs-One classification, in which the objective is to search genes that have low variance in expression levels within the same

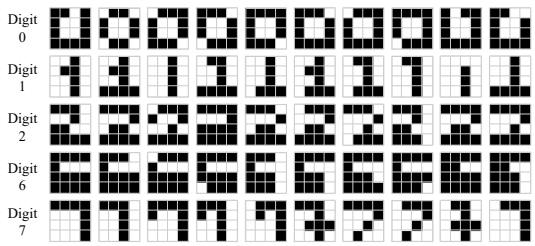


Fig. 18. 4×4 images under the five classes of digits and the associated augmentations.

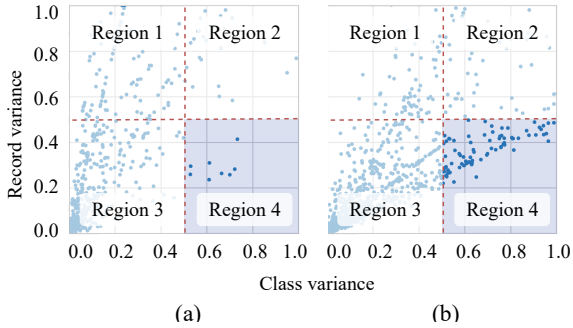


Fig. 19. Illustration of variance analysis results of two input permutations $J = 23$ and $J = 29$ in a) and b) respectively, where the x-axis is the expression variance between classes and the y-axis is the variance for different augmentations within the same class.

class while showing higher variance between classes. Hence, these genes have different fixed expression levels for each class. This plot provides evidence for the existence of the gene-perceptrons that can be expressed in two distinct levels by representing two classes. The x-axis is sorted based on the mean expression distance between the two classes. The gene-perceptrons *b3403, b4063, b3250, b3251,* and *b4150* clearly differentiate between classes c_0 and c_4 with an expression difference of approximately 0.1. Further, gene-perceptrons *b2923* and *b2343* can express at various levels to differentiate between classes c_0 and c_2, and c_0 and c_1, respectively.

Finally, the possibility of using gene-perceptrons for multiclass classification is discussed here. The extended output range due to BReLU allows multiple thresholds that allow multiclass classification, and Fig. 17c presents a series of genes that can be used to classify three classes. The gene *b0565* has the largest distribution of mean expression levels associated with each class. Note that, except for the genes-perceptrons *b0565* and *b0557*, the deviation between multiple classes is low (differences between classes are more difficult to observe). However, it is important to mention that these results are extracted using one set of random inputs, and various input gene-perceptrons enable finding more output gene-perceptrons that will have sparse expression levels for multiple classes.

C. Digit Classification Use Case

This study focuses on using *E. coli* GRNN, discovered through our search algorithm, in a classic use case (digit classification task). The primary goal of this investigation is to systematically analyze each step of the application-specific sub-GRNN search algorithm and the accuracy of the computing required to classify digit images. This section first discusses the experimental setup, the utilization of the proposed sub-GRNN search algorithm, and finally the performance of the GRNN computing for digit classification.

The complexity of the problem is kept low to make the analysis more explicable. We only use 4×4 images with 16 pixels. We used a search data set SD with five classes of digits ("0","1", "2", "6", and "7") and 10 augmentations with significant pattern differences as shown in Fig. 18. Consequently, this SD has the dimensions of 50×16 with an accompanying label matrix of 50×1. Following the proposed search algorithm, first, a pool of 128 ($P' = 128$) gene-perceptrons as suitable candidates are selected for the input layer based on their inward/outward edges' degree distributions. As the images contain ($K = 16$) pixels, the total number of input layer permutations is equal to ${}_{128}P_{16} \approx 1.95 \times 10^{33}$ (Fig. 9, Step 1). Next, SD is encoded in $\mathbf{I}^{(t=0)}$ as explained in Section III-E (Step 2) taking into account the binary properties of pixels. We only use $J < 150$ permutations to find the most suitable sub-GRNN for our digit classification scenario. However, using a smaller number of permutations limits the solution space. While this reduces search time, it also decreases the probability of identifying a more accurate sub-network. Using the GRNN computing model explained in Section III-B (Step 3), the output expression levels for all genes for each of these input permutations are recorded after 43 time steps (where $T = 42$). As the next stage of the search algorithm, the variance analysis is performed. Here we extract a pool of gene-perceptrons for the output layer (Fig. III-E, Step 4), with higher expression variation between classes and low expression variations for different augmentation within the same class, which results in steady expression rates for the corresponding class. The results of the variance analysis for two input permutations with significantly high and low gene quantities in "Region 4" (higher class variance and lower record variance) are shown in Fig. 19. Fig. 19a shows the variance behavior of a random permutation $J = 23$ of inputs where a minimal number of candidates for output genes-perceptrons can be observed in "Region 4". In contrast, Fig. 19b associated with the permutation $J = 29$ has a significant number of candidates in "Region 4", increasing the likelihood of selecting the most accurate output gene-perceptrons.

After ordering the gene-perceptrons by the difference between the first and second highest mean expression values, according to Step 5 in Fig. 9, the output gene-perceptrons are categorized under each digit as shown in Fig. 20. This categorization will allow us to determine which output gene corresponds to which digit. Evidently, Fig. 20a depicts a low variation between the expression levels for each digit class, that is resulted from the low number of gene-perceptron candidates in Fig. 19a for $J = 23$ permutation. However, contrasting results are observed for the expression patterns in Fig. 20b due to the evidently increased number of candidate gene-perceptrons in "Region 4" from Fig. 19b for the $J = 29$ permutation. Hence, we select the input permutation $J = 29$ as the most suitable input layer for this particular problem, as shown in Fig. 22.

The search algorithm can be employed to extract sub-GRNNs for One-vs-One, One-vs-All and multiclass classifi-

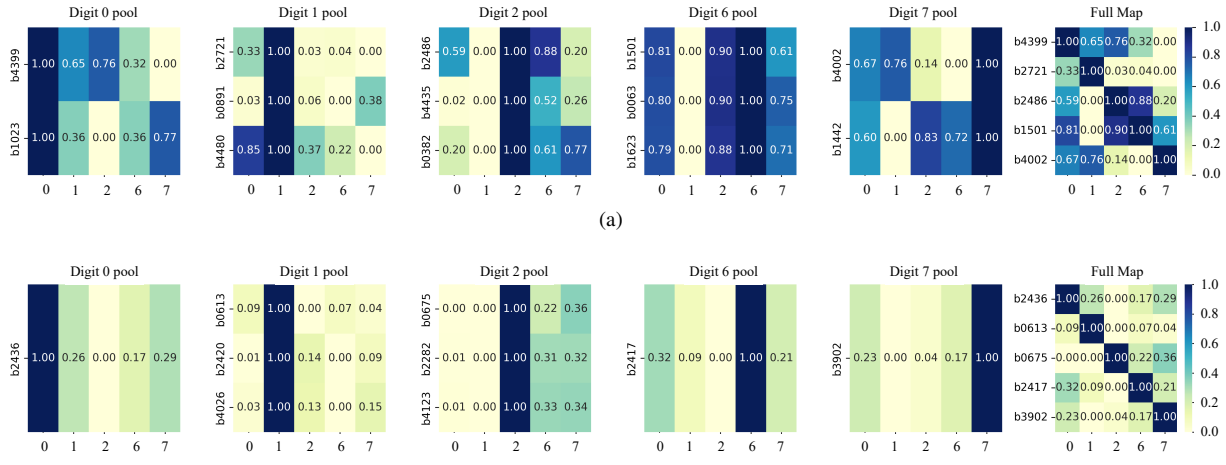


Fig. 20. Heatmaps of normalized gene-perceptron pools for each digit class, where a) shows the expression behaviors for the permutation, $J = 23$, that is associated with Fig. 19a and b) represents the results for permutation, $J = 29$ that is associated with Fig. 19b.

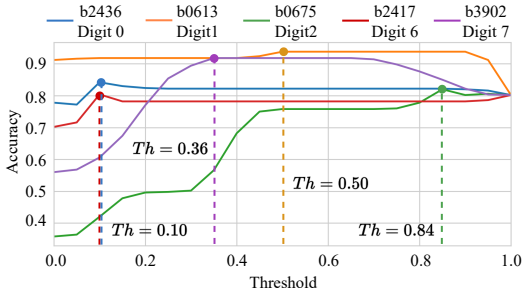


Fig. 21. Determining the thresholds for each class by maximizing the accuracy of each output gene-perceptron. These thresholds are then used to classify the expression levels of the corresponding output gene-perceptrons.

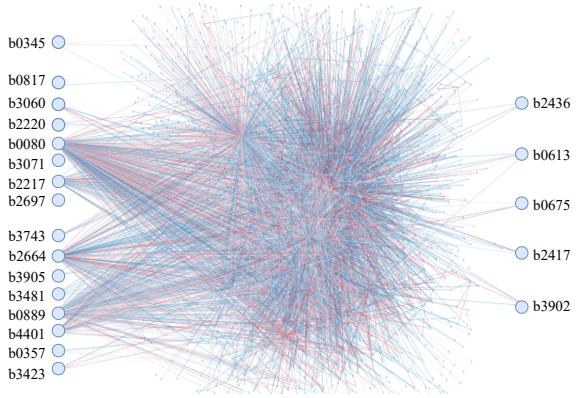


Fig. 22. Illustration of perturbation-based MI analysis on the input and output layer of the extracted sub-GRNN.

cation. However, as the GRNN contains a large number of genes compared to the number of classes in this use case, we only exemplify the method for One-vs-All classification, where each class is assigned with corresponding output gene-perceptrons. Based on the statistical distance relative gene expression levels corresponding to each class of digits, the algorithm then selects five output gene-perceptrons (*b2436, b0613, b0675, b2417* and *b3902*) from each group of digits as shown in Fig. 20b. Subsequently, the algorithm searches for the appropriate expression thresholds (Step 6) which are selected based on the maximum classification accuracy. The classification accuracy, for the five selected output gene-perceptrons, is calculated using (12) for a threshold range (from zero to one with increments of 0.05), and the consequent accuracy variation is shown in Fig. 21. In this case, the accuracy maximization method determines 0.10, 0.50, 0.85, 0.1, and 0.35 as threshold values, and the resulting accuracy for digit classes 0, 1, 2, 6 and 7 was 0.842, 0.938, 0.820, 0.804, and 0.918, respectively.

After the successful extraction of the application-specific sub-GRNN, we performed a perturbation-based MI analysis (Step 7). The results of this MI analysis are presented in Fig. 23 distinctively proving that only the input gene-perceptrons, *b0080*, *b4401*, *b0889*, *b2217* and *b3905* contribute to the decision in the output layer gene-perceptrons. It is understood that the shutdown of gene expression pathways as a result of BReLU causes the disconnection of the information flow between the input and output layer nodes (this lack of connection is evident in Fig. 23, where MI becomes zero). We then extract an optimized network based on these results by reducing the input layer to only have the five investigated gene-perceptrons. As the last step, we compare the decision-making accuracy of the extracted sub-GRNN, before and after condensing the network based on minimizing the number of inputs; see Fig. 24. These results suggest that the optimized sub-GRNN can make decisions close to the previous version of the network, despite the reduced structural complexity due to the low number of input nodes that are stimulated. This reduction in complexity can result in lower ATP energy to fuel gene-perceptrons for computing [45], lower the amount of noise to maximize reliability, and improve the explainability and reproducibility of the network.

VI. Discussion

GRNNs introduced in [21], represent a distinctive form of neural networks naturally embedded within GRNs. These networks can be conceptualized as extensive repositories of pre-trained NNs, at the biological hardware layer, capable of executing intricate and diverse computing tasks. Hence, GRNNs can be regarded as a wet-neuromorphic systems. However, harnessing the potential of GRNNs for computing demands a spe-

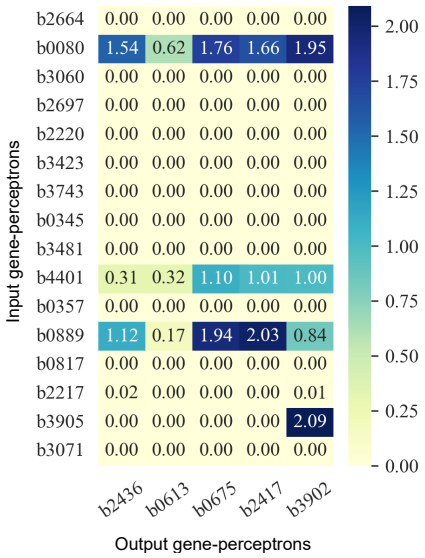


Fig. 23. Illustration of perturbation-based MI analysis on the input and output layer of the extracted sub-GRNN.

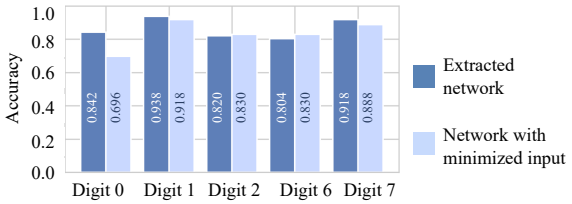


Fig. 24. A comparison of the accuracy of the extracted sub-GRNN before and after minimizing the number of inputs, where the darker columns represent the accuracies of each class before and the lighter columns represent the accuracy values of the network after minimizing the number of inputs.

cialized set of mechanisms, including GRNN extraction and an algorithm for searching application-specific sub-GRNNs. Therefore, this study improved the GRNN extraction method in [21] and introduced a sub-GRNN search algorithm based on random permutation for specific applications. Considering *E. coli* as the model species, we extracted the base-GRNN and proved its accuracy indicating the reliability of the proposed extraction method, the innate computing and the possibility of converging the multidimensional gene-to-gene interaction to a single weight. Subsequently, a feasibility analysis on the extracted *E. coli* GRNN proved its computing capability in classification and regression problems.

The feasibility analyses exhibit the computing power embedded in a single cell and the possibility of mapping sub-GRNNs for wide range of applications. The classification and regression results highlight two noteworthy attributes of GRNN-based computing: analog and parallel computing capabilities. Both analyses used continuous inputs for One-vs-All and One-vs-One classifications, revealing the potential for analog-to-digital computing. Furthermore, in multiclass classification, it is evident that GRNN can be utilized for analog-to-multilevel computing. In addition, we conducted all the analyzes on three datasets for classification, simple regression, and multiple regression. All sub-analyses on classification, simple regression, and multiple regression are done in parallel on the corresponding dataset, highlighting the possibility of using GRNNs for parallel computing, which can be significantly efficient.

Here, we like to highlight the prospective research areas associated with the concept of GRNN as it is still in its infancy. Integrating reporter genes as the output layer of the application-specific sub-GRNN for conveniently observable outputs can be a promising research topic. Similarly, we believe that exploring the possibility of utilizing synthetic proteins as inputs can also be an avenue for further research. Moreover, embedding the metabolomic layer in the GRNN can lead to the incorporation of molecular inputs, which enhances both convenience and practicality in this domain. This study indicates that in the future, GRNN-based biocomputing can be an alternative to silicon-based computing. However, cells exhibit plasticity, dynamically adapting their internal network structures in response to environmental or genetic changes. This flexibility supports GRNN-based dynamic computing, a promising area for future research.Although this study proposes an application-specific sub-GRNN search algorithm, we emphasize that further research is vital for developing more efficient and effective search algorithms. Such advances are crucial to fully exploit the inherent general-purpose computing capabilities of bacterial GRNNs.

REFERENCES

[1] J. L. Poet, A. M. Campbell, T. T. Eckdahl, and L. J. Heyer, "Bacterial computing," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 17, no. 1, pp. 10–15, 2010.

[2] R. Lahoz-Beltra, J. Navarro, and P. C. Marijuán, "Bacterial computing: a form of natural computing and its applications," *Frontiers in Microbiology*, vol. 5, p. 101, 2014.

[3] M. D. Carroll R, "Twenty years of amos research," *Currents in Biblical Research*, vol. 18, no. 1, pp. 32–58, 2019.

[4] M. Kuscu, E. Dinc, B. A. Bilgin, H. Ramezani, and O. B. Akan, "Transmitter and receiver architectures for molecular communications: A survey on physical design with modulation, coding, and detection techniques," *Proceedings of the IEEE*, vol. 107, no. 7, pp. 1302–1341, 2019.

[5] L. Ceze, J. Nivala, and K. Strauss, "Molecular digital data storage using dna," *Nature Reviews Genetics*, vol. 20, no. 8, pp. 456–466, 2019.

[6] L. Rizik, L. Danial, M. Habib, R. Weiss, and R. Daniel, "Synthetic neuromorphic computing in living cells," *Nature communications*, vol. 13, no. 1, p. 5602, 2022.

[7] L. Smirnova, B. S. Caffo, D. H. Gracias, Q. Huang, I. E. Morales Pantoja, B. Tang, D. J. Zack, C. A. Berlinicke, J. L. Boyd, T. D. Harris, *et al.*, "Organoid intelligence (oi): the new frontier in biocomputing and intelligence-in-a-dish," *Frontiers in Science*, p. 0, 2023.

[8] B. J. Kagan, A. C. Kitchen, N. T. Tran, F. Habibollahi, M. Khajehnejad, B. J. Parker, A. Bhat, B. Rollo, A. Razi, and K. J. Friston, "In vitro neurons learn and exhibit sentience when embodied in a simulated game-world," *Neuron*, vol. 110, no. 23, pp. 3952–3969, 2022.

[9] S. Balasubramaniam, S. Somathilaka, S. Sun, A. Ratwatte, and M. Pierobon, "Realizing molecular machine learning through communications for biological ai," *IEEE Nanotechnology Magazine*, vol. 17, no. 3, pp. 10–20, 2023.

[10] S. Angerbauer, F. Enzenhofer, T. Pankratz, M. Hamidović, A. Springer, and W. Haselmayr, "Novel nano-scale computing unit for the iobnt: Concept and practical considerations," *TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.24167163.v1*, Sept. 2023.

[11] O. B. Akan and S. Balasubramaniam, "Body area nanonetworks with molecular communications in nanomedicine," *IEEE Communications Magazine*, vol. 50, no. 1, pp. 28–34, 2012.

[12] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *science*, vol. 266, no. 5187, pp. 1021–1024, 1994.

[13] A. Levskaya, A. A. Chevalier, J. J. Tabor, Z. B. Simpson, L. A. Lavery, M. Levy, E. A. Davidson, A. Scouras, A. D. Ellington, E. M. Marcotte, *et al.*, "Engineering escherichia coli to see light," *Nature*, vol. 438, no. 7067, pp. 441–442, 2005.

[14] J. Baumgardner, K. Acker, O. Adefuye, S. T. Crowley, W. DeLoache, J. O. Dickson, L. Heard, A. T. Martens, N. Morton, M. Ritter, *et al.*, "Solving a hamiltonian path problem with a bacterial computer," *Journal of biological engineering*, vol. 3, pp. 1–11, 2009.
[15] L. J. Heyer, J. L. Poet, M. L. Broderick, P. E. Compeau, J. O. Dickson, and W. L. Harden, "Bacterial computing: using e. coli to solve the burnt pancake problem," *Math Horizons*, vol. 17, no. 3, pp. 5–10, 2010.
[16] A. Pandi, M. Koch, P. L. Voyvodic, P. Soudier, J. Bonnet, M. Kushwaha, and J.-L. Faulon, "Metabolic perceptrons for neural computing in biological systems," *Nature communications*, vol. 10, no. 1, p. 3880, 2019.
[17] X. Li, L. Rizik, V. Kravchik, M. Khoury, N. Korin, and R. Daniel, "Synthetic neural-like computing in microbial consortia for pattern recognition," *Nature communications*, vol. 12, no. 1, p. 3139, 2021.
[18] K. E. Duncker, Z. A. Holmes, and L. You, "Engineered microbial consortia: strategies and applications," *Microbial Cell Factories*, vol. 20, no. 1, pp. 1–13, 2021.
[19] W. Jiang, X. He, Y. Luo, Y. Mu, F. Gu, Q. Liang, and Q. Qi, "Two completely orthogonal quorum sensing systems with self-produced autoinducers enable automatic delayed cascade control," *ACS Synthetic Biology*, vol. 9, no. 9, pp. 2588–2599, 2020.
[20] R. Zhang, H. Goetz, J. Melendez-Alvarez, J. Li, T. Ding, X. Wang, and X.-J. Tian, "Winner-takes-all resource competition redirects cascading cell fate transitions," *Nature communications*, vol. 12, no. 1, p. 853, 2021.
[21] S. S. Somathilaka, S. Balasubramaniam, D. P. Martins, and X. Li, "Revealing gene regulation-based neural network computing in bacteria," *Biophysical Reports*, vol. 3, no. 3, 2023.
[22] S. Lu and F. Xu, "Linear leaky-integrate-and-fire neuron model based spiking neural networks and its mapping relationship to deep neural networks," *Frontiers in neuroscience*, vol. 16, p. 857513, 2022.
[23] S. Cussat-Blanc, K. Harrington, and W. Banzhaf, "Artificial gene regulatory networks—a review," *Artificial life*, vol. 24, no. 4, pp. 296–328, 2019.
[24] J. VOHRADSKY, "Neural network model of gene expression," *the FASEB journal*, vol. 15, no. 3, pp. 846–854, 2001.
[25] C. Scholes, A. H. DePace, and Á. Sánchez, "Combinatorial gene regulation through kinetic control of the transcription cycle," *Cell systems*, vol. 4, no. 1, pp. 97–108, 2017.
[26] A. Ishihama, "Prokaryotic genome regulation: a revolutionary paradigm," *Proceedings of the Japan Academy, Series B*, vol. 88, no. 9, pp. 485–508, 2012.
[27] M. V. Grosso-Becerra, G. Croda-García, E. Merino, L. Servín-González, R. Mojica-Espinosa, and G. Soberón-Chávez, "Regulation of pseudomonas aeruginosa virulence factors by two novel rna thermometers," *Proceedings of the National Academy of Sciences*, vol. 111, no. 43, pp. 15562–15567, 2014.
[28] Z. Koşar and A. Erbaş, "Can the concentration of a transcription factor affect gene expression?," *Frontiers in Soft Matter*, vol. 2, p. 914494, 2022.
[29] A. Ishihama, "Prokaryotic genome regulation: multifactor promoters, multitarget regulators and hierarchic networks," *FEMS microbiology reviews*, vol. 34, no. 5, pp. 628–645, 2010.
[30] T. Barrett, S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, K. H. Phillippy, P. M. Sherman, M. Holko, *et al.*, "Ncbi geo: archive for functional genomics data sets—update," *Nucleic acids research*, vol. 41, no. D1, pp. D991–D995, 2012.
[31] K. Struhl, "Fundamentally different logic of gene regulation in eukaryotes and prokaryotes," *Cell*, vol. 98, no. 1, pp. 1–4, 1999.
[32] X. Huang, C. Song, G. Zhang, Y. Li, Y. Zhao, Q. Zhang, Y. Zhang, S. Fan, J. Zhao, L. Xie, *et al.*, "scgrn: a comprehensive single-cell gene regulatory network platform of human and mouse," *Nucleic Acids Research*, p. gkad885, 2023.
[33] F. Fioravanti, M. Helmer-Citterich, and E. Nardelli, "Modeling gene regulatory network motifs using statecharts," *BMC bioinformatics*, vol. 13, pp. 1–12, 2012.
[34] G. Xue, X. Zhang, W. Li, L. Zhang, Z. Zhang, X. Zhou, D. Zhang, L. Zhang, and Z. Li, "A logic-incorporated gene regulatory network deciphers principles in cell fate decisions," *bioRxiv*, pp. 2023–04, 2023.
[35] V. H. Tierrafría, C. Rioualen, H. Salgado, P. Lara, S. Gama-Castro, P. Lally, L. Gómez-Romero, P. Peña-Loredo, A. G. López-Almazo, G. Alarcón-Carranza, *et al.*, "Regulondb 11.0: Comprehensive high-throughput datasets on transcriptional regulation in escherichia coli k-12," *Microbial Genomics*, vol. 8, no. 5, 2022.
[36] A. Chauvier and N. G. Walter, "Regulation of bacterial gene expression by non-coding rna: It is all about time!," *Cell Chemical Biology*, 2024.
[37] M. Morzy, T. Kajdanowicz, P. Kazienko, *et al.*, "On measuring the complexity of networks: Kolmogorov complexity versus entropy," *Complexity*, vol. 2017, 2017.
[38] H. Zenil, S. Hernández-Orozco, N. A. Kiani, F. Soler-Toscano, A. Rueda-Toicen, and J. Tegnér, "A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity," *Entropy*, vol. 20, no. 8, p. 605, 2018.
[39] C.-E. Yin and G. Qu, "Obtaining statistically random information from silicon physical unclonable functions," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 96–106, 2014.
[40] J. Rojo, L. G. de Pinho, C. Fonseca, M. J. Lopes, S. Helal, J. Hernández, J. Garcia-Alonso, and J. M. Murillo, "Analyzing the performance of feature selection on regression problems: A case study on older adults' functional profile," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 137–152, 2023.
[41] T. Wei, W.-L. Liu, J. Zhong, and Y.-J. Gong, "Multiclass classification on high dimension and low sample size data using genetic programming," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 704–718, 2022.
[42] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.
[43] X. Zhou, Z. Du, S. Zhang, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Addressing sparsity in deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1858–1871, 2019.
[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[45] Q. Qi, Y. Lu, J. Li, J. Wang, H. Sun, and J. Liao, "Learning low resource consumption cnn through pruning and quantization," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 886–903, 2022.

SAMITHA SOMATHILAKA (Member, IEEE) is a postdoctoral researcher at the University of Nebraska-Lincoln, NE, USA. Somathilaka received his PhD in Biocomputing from South East Technological University, Ireland. His research interests include biocomputing, bacterial computing, and gene regulatory neural networks. Contact him at ssomathilaka2@unl.edu.

SASITHARAN BALASUBRAMANIAM (Senior Member, IEEE) is currently an Associate Professor with the School of Computing, University of Nebraska-Lincoln, USA. His research interests include molecular/nano communications, Internet of Bio-Nano Things, and 5G/6G networks. He is currently the Editor-in-Chief of IEEE TRANSACTIONS ON MOLECULAR, BIOLOGICAL AND MULTI-SCALE COMMUNICATIONS and an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING. He was an IEEE Distinguished Lecturer for the IEEE Nanotechnology Council in 2018.

Daniel P. Martins (Member, IEEE) is currently an Assistant Professor and Curriculum Development Director with the School of Computer Science and Electronic Engineering, University of Essex, UK. Daniel received his PhD from Waterford Institute of Technology (2019), and his research interests include cellular signal processing, biosensing, molecular/nano communications and agent-based simulations of biological systems.