Genetic Programming and Evolvable Machines (2025) 26:2
https://doi.org/10.1007/s10710-024-09501-6

REVIEW

A survey on batch training in genetic programming

Liah Rosenfeld[1] · **Leonardo Vanneschi**[1]

Received: 20 May 2024 / Revised: 11 November 2024 / Accepted: 13 November 2024

Abstract

In Machine Learning (ML), the use of subsets of training data, referred to as batches, rather than the entire dataset, has been extensively researched to reduce computational costs, improve model efficiency, and enhance algorithm generalization. Despite extensive research, a clear definition and consensus on what constitutes batch training have yet to be reached, leading to a fragmented body of literature that could otherwise be seen as different facets of a unified methodology. To address this gap, we propose a theoretical redefinition of batch training, creating a clearer and broader overview that integrates diverse perspectives. We then apply this refined concept specifically to Genetic Programming (GP). Although batch training techniques have been explored in GP, the term itself is seldom used, resulting in ambiguity regarding its application in this area. This review seeks to clarify the existing literature on batch training by presenting a new and practical classification system, which we further explore within the specific context of GP. We also investigate the use of dynamic batch sizes in ML, emphasizing the relatively limited research on dynamic or adaptive batch sizes in GP compared to other ML algorithms. By bringing greater coherence to previously disjointed research efforts, we aim to foster further scientific exploration and development. Our work highlights key considerations for researchers designing batch training applications in GP and offers an in-depth discussion of future research directions, challenges, and opportunities for advancement.

Keywords Genetic programming · Batch training · Sampling methods · Generalization · Overfitting

✉ Liah Rosenfeld
lrosenfeld@novaims.unl.pt

Leonardo Vanneschi
lvanneschi@novaims.unl.pt

[1] NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal

Published online: 29 November 2024

1 Introduction

Over the years, researchers in Genetic Programming (GP) [1] have dedicated significant efforts to refining and extending Koza's original formulation. These efforts have focused on better leveraging the algorithm's potential and addressing its inherent limitations, such as loss of diversity [2], computational expensiveness [3], bloat [4] and tendency to overfit [5]. At the same time, within the field of Machine Learning (ML), the usage of parts of the training data (i.e., batches), rather than its entirety, has been extensively studied as a form of not only reducing computational costs [6] but also improving models' performance [7] and enhancing models' generalization abilities [8]. Considering this, learning systems that operate on batches instead of the entire training set introduce an extra parameter known as batch size [9]. As the name indicates, batch size (generally) refers to the number of training instances that are used during the training process or that are grouped together in the same batch. While there are several forms and types of batch training (as will be described in Sect. 2), it is known that the manipulation of this parameter has proven to yield the aforementioned benefits regarding the algorithms' performance, generalization abilities and computation costs [6–8]. However, batch training can be interpreted from various perspectives and there is no consensus or universally accepted algorithm/implementation-independent definition of what constitutes batch training.

One of our objectives in this paper is to explore and clarify the different forms of batch training that can be employed and how existing research can be considered a form of batch training, even if that specific term is not explicitly used. We therefore propose a theoretical redefinition of batch training, broadening its interpretation to be seen as a more comprehensive concept beyond its conventional use in Neural Networks (NNs) and related algorithms. Over the years a wide variety of studies that use a dynamic or adaptive batch size rather than a pre-determined static one, have come to light with the goal of capitalizing on the previously discussed benefits regarding the models' performance and generalization abilities, while still reducing computational costs [7, 10–12]. We believe that GP could benefit from this adaptive/dynamic batch size approach, especially considering its other aforementioned known limitations. As such, our paper reviews the progress made so far on dynamic/adaptive batch size approaches in GP. While this strategy has been extensively studied in other ML algorithms, its application in GP remains underexplored. We believe that this gap in the literature warrants further investigation, presenting potential for future research.

Interestingly, despite existing research on the use of various forms of batch training during the selection or evolutionary GP process [13–15], the term 'batch training' itself is seldom used. This suggests that, even with ongoing research, there remains a significant gap in our comprehensive understanding and exploration of the concept in GP, even more than the previously mentioned general lack of consensus in the ML community.

The absence of a comprehensive survey that summarizes, classifies, explores, and structures the existing batch training methods in GP has left the concept

underdeveloped and vaguely outlined. This survey, therefore, is of significant importance as it fills a critical research gap, providing much-needed clarity and organization, especially to the field of batch training in GP. This clarity is deemed of high importance especially when we consider that the aforementioned lack of clear definition results in a large body of literature that could be seen as a unified entity, or as parts of the same approach viewed from different angles (i.e., different types of batch training), but instead remains unlinked. The need for this clear definition and the integration of previously unrelated data becomes even more crucial when we consider the practical impact of the wide variety of batch training methodologies. These methodologies have significantly improved the ability of complex ML algorithms to handle more amounts of data than ever before [16], which generalized their ability to handle real-world data. For instance, they have contributed to advancements in fields such as healthcare [17], pharmacokinetics [18], astronomy [19], forest fire spread prediction [20] and more.

Even when considering batch training in its current "traditional" form, it is clear that these applications have attracted significant attention from researchers in recent years due to their practical applications and implications for ML as a whole. To better understand and quantify this growing interest, we aimed to visualize and analyze research trends from 2000 to 2023. For this purpose, we used the Scopus Search Engine to conduct a query that captures the literature on batch training in its current, simplified "traditional" view. This query was:

(TITLE-ABS-KEY("batch training") OR TITLE-ABS-KEY("batch-training") OR TITLE-ABS-KEY("mini-batch")) AND PUBYEAR > 1999 AND PUBYEAR < 2024

As shown in Fig. 1, this growth in attention towards batch training applications over the years becomes noticeable, even when considered under the current vaguely

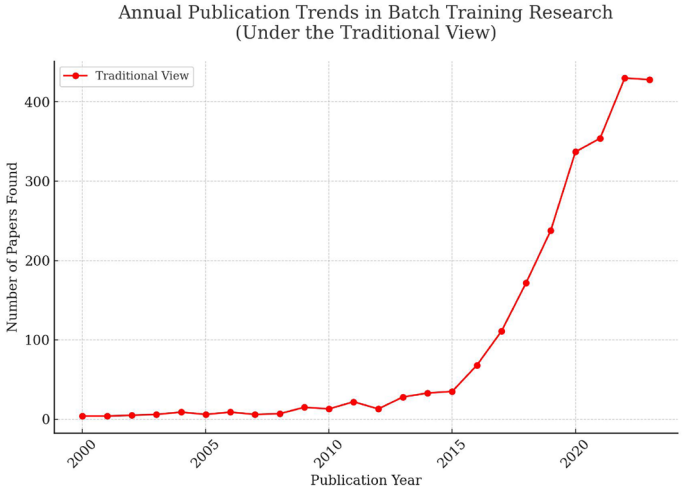

Fig. 1 Number of papers on batch training under its vaguely defined traditional perspective over the years

defined, oversimplified view. This is unsurprising, given the previously mentioned potential advantages of these approaches.

Given the vast amount of literature on batch training, even in its traditional sense, all existing studies could not be covered within this paper. Instead, we selected references to avoid redundancy, focusing on the most relevant and impactful works that would enhance readers' understanding by supporting the narrative and clarifying the proposed redefinition of batch training.

With this in mind, our goal is to further encourage and guide future research in the area of batch training by proposing a theoretical reformulation that encompasses a broader range of literature than the one seen in Fig. 1, thereby connecting research that was previously considered unrelated. To achieve this, we propose a novel categorization for batch training methods, highlight key considerations for researchers to consider when developing batch learning applications and provide an in-depth discussion on potential future applications and research directions.

This paper is structured as follows: Sect. 2 introduces the fundamentals of batch training, along with a proposed theoretical categorization that distinguishes between different types of batch training techniques. It also provides an overview of dynamic/adaptive batch training methodologies. Section 3 presents the current state of the art in batch training within GP, showing how current approaches and studies fit within the aforementioned novel proposed categorization. Section 4 highlights what are the key factors that we consider researchers should take into account when designing future methodologies for batch training. Section 5 discusses the current challenges in batch training applications and explores potential future directions and research opportunities. Finally, Sect. 6 offers an overview of the goals and conclusions drawn from the analyses conducted in this review.

1.1 Notations and terminology

Before moving forward, we offer a brief overview, definition, and clarification of some of the key concepts that will be discussed throughout this paper, providing this summary in Table 1. Although each term will be elaborated upon in detail later,

Table 1 Short overview of key terms and concepts used in this survey

Term	Definition
Batch	Group of training instances (i.e., fitness cases) that are processed together at a given time
Batch size	Parameter referring to the number of training instances that are grouped together in the same batch
Batch training	Any training method that does not entail processing the entire set of training data at once
Dynamic batch size training	Batch training techniques where the batch size parameter is modified according to a predetermined schedule throughout the learning process
Adaptive batch size training	Batch training techniques where the batch size is adaptively adjusted during the learning process based on the model's performance or other criteria

this initial overview is intended to facilitate understanding and enhance the reading experience.

2 Fundamentals of batch training

This section provides a brief introduction to the concept of batch training, while also suggesting a novel classification of the different types of batching approaches found in the literature.

In general, we consider batch training as any form of training that does not use the entirety of training data at a single time with the goal of tackling the previously described challenges. With that in mind, several forms of batch training can be employed. Here, we propose a taxonomy that broadens the concept to include a wider range of methodologies and objectives. Our aim is to unify diverse approaches under a common term, recognizing their shared (theoretical or methodological) characteristics when viewed from a higher-level perspective. This reformulation encourages the use of "batch training" to describe more algorithms than currently acknowledged, allowing previously disconnected bodies of research to be seen as part of a larger, integrated field of study. With this goal in mind, as can be observed in Fig. 2, we identify three different batch training strategies: (a) Data Parallelism, (b) Horizontal batching and (c) Vertical batching. These different strategies are briefly explained in this section.

2.1 Data parallelism

In order to improve training efficiency, in previous years ML researchers have been giving attention to the parallelism and scalability (i.e., the extent to which the training process can be conveniently spread across multiple devices) capabilities that algorithms can have [21, 22]. With that in mind, data parallelism consists in splitting the data into several batches that are spread across multiple devices or cores, while aggregating the different results at a later stage [21]. As such, the training occurs simultaneously across different machines (or cores), each one operating on a different batch of the overall training data. This data parallelism approach is capable of

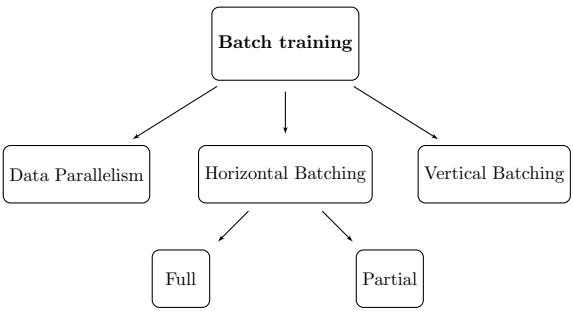

Fig. 2 Categorization of the different batch training types

reducing the communication costs while also optimizing the ML algorithms' execution run-times [22].

2.2 Horizontal batching

Horizontal batching is a different and simpler form of optimizing algorithms' execution time, while also reducing the computational evaluations required. It entails using a (typically) random batch of the training set rather than its entirety, without necessarily resorting to any parallelization. For instance, a Neural Network (NN) that uses horizontal batching would go through the training dataset by using a different batch of the training data for the weights adjustment at each epoch [23], a Random Forest (RF) would be built by the agglomeration of different Decision Trees (DTs), trained on their own sub-set of the training data [24], while an evolutionary algorithm such as GP could evaluate its individuals on a random batch of the training set at each generation [25].

We choose the term horizontal batching due to the fact that training instances (i.e., fitness cases) are organized as rows in the dataset, resembling "horizontal portions". Therefore, randomly selecting a batch of training instances entails selecting rows (i.e., horizontal lines) from the training set in some manner.

One might look at data parallelism as a form of horizontal batching, given how the training instances are not all processed as a whole, considering that they are divided across different machines or cores. However, we suggest a distinction between these two forms of batching as not all algorithms can easily be parallelized. Additionally, the focus of the two forms of batching seems to be different: while data parallelism seeks to speed the execution time by separating the processes across different machines or cores, horizontal batching simply seeks to save on computational efforts or potentially even improve algorithms' performance (e.g., [24, 26, 27]) by not processing all the training data or, at least, not processing all of it at the same time.

Finally, we suggest a further distinction between (a) full and (b) partial horizontal batching. Full horizontal batching occurs when the training instances are separated into different batches, yet all of the batches are used by the algorithm at each iteration or training iteration. In other words, in a full horizontal batching approach, all the training instances are still processed at each iteration, but that processing occurs in batches. On the other hand, in partial horizontal batching only part of the training instances at each iteration are being used. This means that, depending on the strategy for the selection of the training instances that take part of the used batch, there usually is no guarantee that all the training instances are used at least once.

When discussing the proposed concept of horizontal batch training, it is natural for readers to draw comparisons with sub-sampling techniques. In fact, sub-sampling can be viewed as a specific form of horizontal batch training, as both involve selecting a subset (or batch) of training instances from the full dataset. The key distinction is that we consider that sub-sampling involves reducing the original dataset to a smaller sample or using a smaller sample from the original dataset, while horizontal batch training refers to the underlying methodology of how this subset

is utilized during the training process, not merely the use of the subset itself. Thus, horizontal batch training, as we propose it, is a broader theoretical framework that encompasses any method where not all training instances (or their features) are processed together at a single time. Within this framework, a method can be classified as full or partial horizontal batch training, depending on whether all subsets are used in each iteration or only a portion is selected.

Sub-sampling strategies, on the other hand, are more narrowly focused on selecting a representative sub-sample to reduce the dataset size for computational efficiency [28–30], without necessarily dictating how the processing should be organized or repeated over time. Sub-sampling also differs from our broader definition of batch training as it does not inherently align with data parallelism, which focuses on distributing training instances across multiple devices without reducing their total number, nor does it align with vertical batch training, which deals with selecting a subset of features rather than instances. In essence, while sub-sampling methods fall under the umbrella of horizontal batch training, batch training as we define it refers to the broader methodological approach of not processing the entire dataset at once, whether by instance or feature. Sub-sampling is simply one way to achieve this by drawing a representative subset, whereas batch training involves the specific strategy of how these subsets are processed throughout the learning process.

2.3 Vertical batching

While horizontal batching uses a batch of the training instances (i.e., rows) in the data, vertical batching consists on using a batch of vertical columns in the data (i.e., variables). Not using all the variables in the dataset reduces the data-dimensionality, relieving the system of part of the weight of its memory requirements [31]. It is important to note that usually the usage of a subset of the variables in the dataset is not referred to as batch training but as feature selection [32]. In our redefinition of batch training, we propose a broader framework that includes any training approach where the entire dataset is not processed at once. Within this broader context, feature selection naturally fits as a form of batch training.

We choose not to delve deeply into feature selection because we consider it to be a well-established, popular and widely studied area, with many recent discussions and reviews providing comprehensive overviews of its methods and applications (see [33–35]). Given the wealth of recent research, we believe it would be redundant to explore it in detail in this paper, allowing us instead to focus on other, less-discussed aspects of batch training that contribute to our proposed broader framework. Additionally, we chose to not include vertical batch training approaches but still explore data parallelism methodologies because we believe that investigating data parallelism within the context of our batch training definition is particularly relevant, as it supports our goal of encouraging future research in batch training. This is because data parallelism plays a crucial role in both the choice of software and the design of new implementations.

Figure 3 offers a visual summary of the categorization of the proposed various types of batch training strategies.

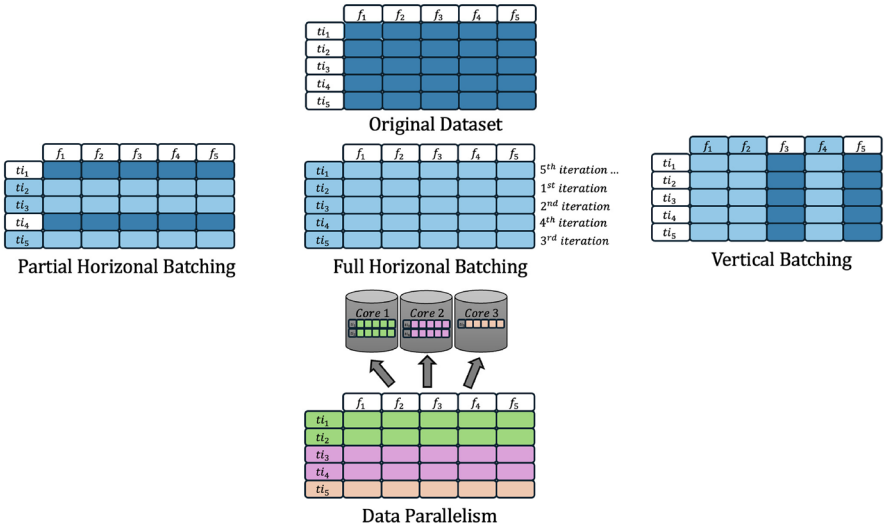

Fig. 3 Visual summary of the proposed categorization for the different batch training strategy types

2.4 Dynamic batch size training

As previously discussed, in recent years, extensive research has focused on developing implementations that utilize dynamic or adaptive values for the batch size parameter instead of fixed, static ones. This approach aims to further optimize the advantages of using batches rather than the entirety of the dataset at a given point, thereby reducing computational resource usage and/or enhancing the overall generalization capabilities of models [7, 10–12].

Before delving further into the concept of dynamic batch size methods and reviewing the relevant literature, it is important to expand on the previously introduced terminology. This clarification will help to more precisely distinguish between the two key terms that are used throughout this paper and are pertinent to these methods:

Dynamic Batch Size Training This term refers to strategies in which the batch size changes throughout the training process in a predefined manner. In this context, we use the term dynamic batch size training to refer to literature where the batch size is modified according to a predetermined plan that remains consistent throughout the training, regardless of any events or behaviors that occur during the learning process.

Adaptive Batch Size Training This term describes methods where the batch size is adjusted dynamically in response to specific triggers or conditions that arise during the learning process. Here, the batch size is modified based on the "needs" of the algorithm, which are identified through certain measures or statistics indicating that a change is necessary. As such, all bodies of research in which the batch size is not static are considered dynamic but not all dynamic batch size training methods are considered adaptive.

An intriguing example of dynamic batch training is provided by the work of Smith and colleagues [36], which proposed gradually increasing the batch size parameter during Stochastic Gradient Descent (SGD) instead of the most popular approach of decaying the learning rate. By reducing the frequency of parameter updates, this approach shortens training times and even improves parallelization without sacrificing performance. We consider this particularly relevant as it suggests that dynamic batch size adjustments can optimize training by influencing various aspects of the algorithm beyond just the ones regarding the training instances. Subsequent and parallel research to that of Smith and colleagues has shifted focus towards adaptively adjusting batch sizes in response to the model's changing needs during training, rather than adhering to a fixed schedule.

In fact, one of the earliest adaptive batch size methods, the AdaBatch algorithm, was introduced by Devarakonda, Naumov, and Garland [10]. This approach gradually increases the batch size during DNN training when learning stagnation is detected through gradient noise, aiming to combine the fast convergence of smaller batches with the efficiency of larger ones. The results showed speed improvements of up to six times while maintaining accuracy, suggesting that there may still be unexplored advantages in optimizing batch training for even greater performance gains.

Alsadi and collaborations [37] proposed an alternative approach to accelerate SGD, distinct from the previously explored gradual increase in batch size. Their method introduces a "switching criterion" that dynamically adjusts the batch size based on the current model's output (through the gradient variance). This criterion allows the learning process to predominantly use smaller batch sizes, increasing to a larger size only when necessary, and then switching back to a smaller batch size whenever possible, thereby alternating between the two in a more responsive manner. This adaptive approach enabled a substantial reduction in computational resources while preserving the models' predictive performance [37].

Building on Alsadi and collaborator's work, Takase [7] proposed a dynamic batch size strategy using generalization loss as a "switching criterion" to enhance model performance. Instead of halting training, Takase's method reduces the batch size to help the model escape local minima and improve both training speed and generalization, thereby reducing overfitting. These findings are promising, as they underscore that a "working smarter, not harder" perspective can additionally be applied to batch training applications in order to even further enhance both the quality of solutions and computational efficiency. Furthermore, they emphasize that adopting an adaptive approach that responds to the model's needs, rather than relying on a predefined dynamic change, appears to be a more promising and learning-specific strategy.

Another notable adaptive batch training strategy is presented in the work of Ye and colleagues [38], where they explore using an adaptive batch size within a distributed Deep Neural Networks (DNNs) framework. Their approach integrates data parallelism with an adaptive batch size that adjusts according to hardware constraints, effectively tackling synchronization issues by preventing faster workers from being delayed by slower ones. This method achieved a smoother training process, enhancing speed without sacrificing performance. Their work highlights an interesting application of a dynamic batch size approach for data parallelization.

The benefits of adopting an adaptive or dynamic approach during the training process have demonstrated significant potential over recent years. Shifting away from a static batch size has been proving to be advantageous across various applications, such as optimizing hardware-related efficiency [20, 39–41], improving non-hardware-related computational performance and convergence [10, 42, 43], and enhancing models' generalization capabilities [7].

3 Batch training in GP

If we consider the categorizations and definitions of batch training detailed in Sect. 2, it is possible to find a significant amount of literature reporting on research that applies batch training in GP.

3.1 GP data parallelism techniques

Within the aforementioned batch training strategies, data parallelism (as defined in Sect. 2.1) has understandably accumulated significant interest within the GP community due to its capacity to mitigate the intensive computational demands of the algorithm. Within this realm of GP, parallelization techniques have also been extensively investigated due to the algorithm's inherently "embarrassingly parallel" nature [44, 45]. These techniques can be broadly categorized into two types: population-based parallelization and data-based parallelization [46]. The former involves independently evaluating the individuals that are within GP's population, while the latter refers to the parallel evaluation of individuals across the different training instances [46]. Note that in the context of this survey, data-based parallelization stands out as the most relevant approach, as the training cases are not used together in a single batch.

The parallelization of the fitness evaluation stage, which is widely recognized as being the most time consuming aspect of the GP process has continuously been studied as a form of generalizing the use of algorithms like GP to larger datasets [47, 48]. The exploration of data parallelism in GP has a long history, tracing back to Tufts' Single Instruction Multiple Data (SIMD) parallel computing architecture work in 1993 [49]. In recent years, the spotlight of the research community has especially been on frameworks that are designed to exploit computers' multi-core GPUs, aiming to boost parallelism beyond what is commonly achieved with conventional CPU implementations [46–48]. This many-core GPU approaches emerged in the past decade, with initial contributions made by Banzaf [47, 50] and Chitty [51].

The work of Harding and Banzaf [47] proposed using the Accelerator library [52] in order to execute the fitness evaluation process in a parallelised fashion on the GPU, while running the population and algorithm on the CPU. Their $C^\sharp$ implementation obtained results that are several hundred times faster than a typical CPU approach. Not long after, Langdon and Harrison [53] aimed to leverage the widely-used general-purpose graphics processing units (GPGPU) to illustrate how parallelizing the evaluation of GP individuals enables GP to handle

large-scale datasets effectively. More specifically, the $C++$ implementation featured in [53], revealed that executing the same GP individual on different fitness cases across numerous processors at the same time, enabled GP to efficiently learn from a real-life breast cancer dataset comprising a million inputs. These findings have further demonstrated the potential of GPU based parallelization shown by Harding and Banzaf [47], demonstrating that GPU parallelization can overcome the computational expensiveness limitation of GP and its previously believed confinement to smaller datasets.

The survey conducted by Langdon in 2011 [54] on the adoption of GPU parallelization in GP sheds more light on the scientific community's awareness of the potential of such implementations. It not only showcases the vast body of literature that was already available over a decade ago, but also reveals that the fastest GP applications rely on interpretation rather than compilation. In the years that followed, scholars concentrated on further exploring GP's parallelization, utilizing emerging technologies or libraries (e.g., [19, 51]). They also explored the combination of data parallelism techniques with other approaches like population parallelism (as seen for example in [46, 55]). The study into emerging technologies led to the findings that multi-core CPUs could deliver results at a speed that was either faster or equivalent to those of many-core GPU applications [51]. In light of this, the research conducted by Chitty [55] put forth a two-dimensional stack GPU-based GP, which was capable of executing more than 55 billion GP operations per second. This reaffirmed the capability of GPU-based GP parallelization to drastically boost the computational speed of the algorithm, thereby enhancing its usability.

More recently, Staats and collaborators [56] introduced an open source Python library called KarooGP. This library was developed making use of the application of vectorized data and TensorFlow for both CPU and GPU architectures. It was evaluated on datasets ranging from fewer than 20 training instances to a substantial, real-world dataset with over 5 million rows. Their findings aligned with much of the literature cited in this paper, demonstrating that on the largest dataset, the GPU configuration was, on average, 1.3 times faster than CPU configurations. This library laid a "modern" foundation for the development of open-source GP libraries with TensorFlow in the widely popular programming language, Python.

A few years later, in 2021, the foundation set by KarooGP inspired further research, leading to the development and implementation of GP libraries that leverage Python's TensorFlow capabilities. This resulted in the work of Baeta and collaborators [19], who introduced the TensorGP library. TensorGP distinguished itself from KarooGP by utilizing more recent TensorFlow executions and incorporating the ability to process images with image-specific operators. This use of updated TensorFlow executions proved advantageous for TensorGP, as it was shown to outperform KarooGP in terms of speed [57].

In the same year, Sathia and collaborators [58] introduced an innovative GP implementation utilizing CUDA for GPU parallelization during the selection and evaluation phases of the algorithm. Although this GP implementation is not a standalone library, it can be employed as an algorithm within CuML [59]. The results demonstrated that this CUDA-based method was able to achieve better speeds than

TensorGP and KarooGP on both regression and classification datasets, with the largest dataset containing around 16 million training instances.

The comprehensive research explored in this section enables us to comprehend that the current forefront in GP science is marked by a wide spectrum of frameworks that leverage data parallelism to improve the algorithm's computational efficiency. This includes everything from the $C++$ Operon framework [45] to Python's TensorGP [19]. Finally, it is important to note that data parallelism employs the full dataset with an emphasis on optimizing computational efficiency based on hardware. As such, this form of batch training in GP can be viewed as a more pragmatic approach, which typically divides the training instances into separate batches based on hardware constraints [38] rather than their inherent characteristics. Consequently, this form of batch training is primarily concerned with computational limitations and does not aim to utilize batch training as a significant influence on GP's performance or generalization capability.

3.2 GP horizontal batching techniques

Once the explanation of horizontal batching given in Sect. 2.2 is understood, it becomes clear how various research schools within the GP community can be seen as aligned with this form of batch training.

Contrary to the data parallelism strategies discussed earlier, strategies involving horizontal batching typically employ different batches of training data with the objective to not only cut down on computational costs (which may not always be possible), but also to enhance the generalization capabilities of algorithms (e.g., [25, 60]), increase population diversity (e.g., [61]), and improve models' problem-solving skills (e.g., [62]). These strategies are therefore not focused on optimizing the learning process in an hardware-like fashion.

Horizontal batch training techniques in GP are commonly referred to in literature as sampling methods or fitness-case sampling methods. These methodologies have been thoroughly investigated and are known to exhibit beneficial attributes, especially in symbolic regression problems. They not only enhance generalization and performance, but also reduce individuals' growth [63].

Lexicase selection [61], which has been extensively researched, can be regarded as one of the most prevalent methods of horizontal batch training in GP. Instead of evaluating GP individuals based on their performance across all data, Lexicase selection assesses them on random sequences of training cases [64]. As explained in the work of Spector [61], in its traditional form, Lexicase selection begins by randomly arranging the training data. The GP individual that performs best on the first training instance is chosen to be a parent. If there's a tie for the best performance, the process is repeated with the next training instance, and so on, until only one top-performing individual remains. This individual is then selected as the parent. Thus, GP individuals' performance is measured on individual fitness cases rather than aggregated fitness metric [15]. Given how all individuals are evaluated on all training instances, we consider Lexicase selection as an example of full horizontal batch training method in its conventional form, as it

considers batches of training instances rather than the entire dataset for each parent selection moment. This semantically aware form of selection [65] has proven to positively impact GP populations' behavioural diversity [15, 66, 67] while also outperforming other selection methods on both classification [15] and regression [68] problems. Even though we classify Lexicase selection as a horizontal batch training method, it encounters the problem of being computationally demanding due to its requirement of evaluating every individual on all training cases [13, 69]. In light of this, researchers have dedicated their work towards developing more computationally efficient versions of Lexicase selection. These versions not only retain the previously mentioned advantages but also aim to enhance them, all while reducing computational expenses [13, 62, 69].

For instance, researchers have more recently been able to show that the aforementioned benefits of Lexicase selection can be enhanced by the down-sampling (i.e., random sub-sampling) of training instances [64]. Originally introduced by Hernandez and collaborators [70], this down-sampling Lexicase selection strategy can be considered a partial horizontal batch training technique as only a random batch of the training dataset at each generation is used. Thus, this strategy only uses *batchsize* number of training instances, reducing the number of fitness cases that are considered at each generation. This reduction in the computational effort required for the evaluation of each GP individual reduces the algorithm's runtime or, alternatively, can allow for an increase in the number of generations executed or the population size used [64].

This latest variation of Lexicase selection retains the semantic awareness characteristic of traditional Lexicase selection. It continues to enhance solution robustness by specializing in various training instances while considering computational resource constraints. This method reduces the training dataset size to conserve resources, yet still improves generalization abilities, showcasing the potential and power of partial horizontal batch training approaches in GP.

The more recent study of Boldi and collaborators [69] has introduced the concept of informed down-sampled Lexicase selection. As the name so indicates, this method seeks to obtain an "informed" batch of training instances rather than a random one. Using population statistics, the authors were able to form training batches that contain fitness cases that are more distinct, therefore reducing redundancy. Their results concluded that this informed strategy is able to equally spare on computational resources while even outperforming traditional down-sampled Lexicase selection techniques.

These recent advancements offer a new strategy for horizontal batch training applications, specifically in how training instances are selected for each batch, addressing a significant challenge in these applications: the reliance on randomness for selecting training instances.

Another Lexicase variant that is relevant to this survey is Batch Tournament Selection (BTS) [13], developed by Melo and collaborators. In each generation, BTS separates all the training instances into different batches based on their error. This is done so that instances with the same difficulty level end up in the same batch. Then a tournament selection is performed with random individuals of the population for each one of the batches. According to the results found in [13], using this strategy

leads to a selection method that is approximately one order of magnitude faster than Lexicase selection.

This variant emphasizes not only achieving faster results without sacrificing generalization gains but also highlights the importance of carefully considering how batches are formed and which instances are included alongside one another. This shows once again that this aspect is worthy of attention and optimization.

It is important to note that these applications are not the only form of partial horizontal batch training approaches in GP. In fact, sampling methods in GP date all the way back to the previous century, with earlier studies such as the work of Gathercole and Ross [71] in 1994 which proposed three different strategies that can be employed for the sampling of a small subset of the training data that is to be used during the training process of GP, rather than its entirety.

Later on, the work of Curry and Heywood [72] presented a novel approach to sub-sample the training data in a way that guides GP's learning process towards the training instances that are deemed most difficult to learn or that have been least recently visited. The obtained results showed that this approach was able to obtain similar performance to the traditional approaches while significantly saving on computation time. This enhancement allows us to see, once more, how these techniques can help the application of GP to larger datasets rather than its confinement to smaller ones [72]. On the same note, the work of Gonçalves and collaborators [18] concluded that using batches of the training data during the learning stage not only leads to a better computational efficiency but also to a significant improvement in GP's generalization ability. These results are of high importance as they show that following a "work smarter not harder" approach can be beneficial for computationally complex algorithms such as GP.

Another interesting study in GP that can be classified as a full horizontal batch training is the multi-island approach proposed by Kommenda [73]. In this study, a multi-island population strategy was used, where each island worked with a different partition of the training data. This approach fostered genetic diversity among individuals and improved GP's ability to generalize. It not only created specialized individuals optimized for specific data subsets but also enhanced their robustness, as they were re-evaluated on new data subsets when migrating to different populations. The results showed that this method led to better overall performance, improved generalization, and significantly reduced computational costs.

Returning to the theme of partial horizontal batch training applications, the work of Zoppi et al. [74] aimed to minimize the training dataset size to the smallest necessary amount. They followed a mathematical rule-of-thumb suggesting that training is efficient only if the number of training instances is at least equal to the entropy of the target function. Consequently, their proposed model randomly discarded training instances in each generation, retaining only enough to match such entropy. The findings indicated that this method not only intuitively reduced computational costs significantly but also enhanced the models' generalization abilities. This recent approach offers valuable insight into the potential of using only what is "needed" during the learning stage of GP, rather than everything that is available.

Finally, as highlighted earlier, the field of GP boasts a rich variety of research on sub-sampling methods, which fall under horizontal batch training methodologies.

For a more in-depth comprehension of these horizontal batch training approaches in GP, the reader is referred to the comparative analysis undertaken by Martínez and collaborators [63] or the one conducted by Hmida and collaborators [75] shortly after.

3.3 Dynamic batch size training in GP

As previously mentioned, there is a noticeable research gap with regards to the exploration of the potential and impact of an adaptive batch size methodology in GP. Despite a few studies delving into this area, this adaptive batch size methodology in GP remains significantly uncharted. The existing studies are either briefly exploratory or exceptionally specific, which leaves a scientific gap in the scientific community's understanding of how such methodologies can be exploited for the benefit of the GP algorithm.

The earliest significant work on this topic in GP seems to date back a PhD thesis published in 2016 by Gonçalves [25]. This earlier exploration, that is perhaps the most relevant thus far in the realm of adaptive or dynamic batch size in GP, explored the usage of small dynamic subsets of the training data with the goal of reducing overfitting. Initial results concluded that, as expected, a large batch size resulted in better training fitness (as it allows the model to learn faster), at the cost of model generalization. Thus, Gonçalves explored the interchanged usage between a single training instance and all the available training data with the goal of keeping overfitting low (by using a single training instance) while still allowing the model to find the underlying pattern (by providing it with enough training data). Several strategies concerning which of the options between the usage of all training data or just a single instance should be chosen more often were explored. Results show that using a single training instance more often than most of the data was the most effective in reducing overfitting.

These results are interesting as they show how dynamically switching between underfitting and overfitting tending approaches can aid the overall model's generalization. Additionally, these results reveal once more that using a smaller batch size allows both for a better computational efficiency as well as model generalization.

However, Gonçalves' [25] work applied a deterministic dynamic batch sizing approach as the probabilities of using either a single training instance or the full training data were established a-priori. Thus, while this work is dynamic, it is not an adaptive batch sizing.

The only adaptive batch size study in the field of GP (at least to our knowledge) was performed very recently in a very specific sub-field of GP called Cartesian Genetic Programming (CGP). This work, developed by Lima and collaborators [76], proposed an adaptive batch size approach for CGP where the fitness estimation of the candidate solutions was incrementally improved by using more training instances (i.e., terms of the truth table). This approach used a simple moving average of the model's accuracy as a criterion for detecting evolution stagnation. When such stagnation occurred, the batch size was increased. Results indicated that using an adaptive batch size led to better results and a reduction of the needed evaluations. These

results are not unexpected, especially considering the positive outcomes observed in other ML algorithms such as NNs, which have been extensively researched using adaptive batch sizing techniques, showing significant improvements in execution speed and generalization (see for example [7, 77, 78]).

Future work should focus on extending this methodology to standard GP, while also testing other stagnation criterion such as other known early stopping criterion, as seen in [7]. Indeed, upon discussing the literature presented in this section, the literature gap that was previously mentioned throughout this review appears to be evident. The scientific state of the art presented in this survey suggests that there might be a need for the development of a dynamic batch sizing method that does not use a pre-determined probability, like in the work of Gonçalves [25] nor is it applied for such a specific application, like the work of Lima and collaborators [76].

Table 2 briefly summarizes some of the relevant literature presented in this survey, with regards to its authors, year of publication and suggested batching category approach. Note that rows that are marked with (*) consist in a framework publication while rows that are marked with (△) consist on dynamic batch size GP applications.

Table 2 Relevant literature on GP batch training applications

Paper	Year	Batching category
Curry and Heywood [72]	2004	Partial horizontal batching
Harding and Banzhaf [47]	2007	Data parallelism
Chitty [51]	2007	Data parallelism
Langdon and Harrison [53]	2008	Data parallelism
Langdon [54]	2011	Data parallelism
Spector [61]	2012	Full horizontal batching
Gonçalves et al. [18]	2012	Partial horizontal batching
Helmuth et al. [15]	2014	Full horizontal batching
Kommenda et al. [73]	2014	Full Horizontal Batching
Gonçalves [25]	2016	Partial horizontal batching (△)
Cava et al. [68]	2016	Full horizontal batching
Helmuth et al. [66]	2016	Full horizontal batching
Chitty [55]	2017	Data Parallelism
Staats et al. [56]	2017	Data parallelism (*)
De Melo et al. [13]	2019	Full horizontal batching
Hernandez et al. [70]	2019	Partial horizontal batching
Burlacu et al. [45]	2020	Data parallelism (*)
Baeta et al. [19]	2021	Data parallelism (*)
Sathia et al. [58]	2021	Data parallelism (*)
Helmuth and Spector [64]	2022	Partial horizontal batching
Zoppi et al. [74]	2022	Partial horizontal batching
Lima et al., [76]	2023	Partial horizontal batching (△)
Ding et al. [62]	2023	Partial horizontal batching
Boldi et al. [69]	2024	Partial horizontal batching

4 Key considerations for batch training designs in GP

This section is intended to guide future research on batch training in GP. We aim to assist researchers by delineating key considerations for designing batch training applications in GP, based on the extensive literature reviewed in this survey and the novel categorization introduced in Sect. 2. It is worth mentioning that most of the considerations outlined in this section are applicable to batch training applications in general, not just for GP. However, in keeping with the objectives of this paper, we have chosen to focus on the GP community and discuss these general guidelines along with specific details relevant to GP.

4.1 Data quality

Naturally, as with all ML applications, the quality of your data greatly influences your ability to effectively utilize batch training in its various forms. Over the years, significant research has been conducted to quantify dataset quality based on its attributes and to identify which attributes determine this quality. These studies are vital because the true value of data can only be unlocked with high-quality datasets [79].

The quality of a dataset can be evaluated from a more "technical" perspective by examining the presence of issues such as labeling errors [80, 81], duplicate data [79], missing data [82] and others. Alternatively, it can be assessed from a "higher-level" perspective, considering factors such as dataset size [83], number of attributes [83], complexity [84], and target distribution or imbalance [85].

Understanding the complexities and nuances of data enables researchers to steer their implementation designs, leveraging this knowledge to better exploit their models' capabilities. For example, Bacardit and colleagues [86] developed an efficient Pittsburgh-style genetic-based learning system that utilized partial horizontal batch training to reduce domain dimensionality. This approach minimized the number of training instances considered for fitness evaluation in each generation. Additionally, their work produced theoretical models capable of selecting an appropriate batch size based on problem complexity. This study demonstrates how dataset quality attributes and characteristics, such as problem complexity, can not only complement batch training but also guide the process, aiding in determining the crucial batch size parameter.

Additionally, when deciding on a batch training design, researchers should take into consideration that horizontal batching strategies might not yield good results in datasets that are too small or are imbalanced. Overall, ML algorithms typically exhibit improved performance with larger datasets [87]. The increased volume of training instances generally provides a wealth of information, which enhances algorithms' detection of patterns abilities and subsequently improves the their generalization capabilities [87]. Thus, researchers must take into account that larger datasets are deemed to have greater statistical power than smaller ones who might struggle to find patterns in the data [88].

Considering this, when the dataset size is small, depending solely on batches from the original dataset can be disadvantageous for researchers. The dataset is more likely to lack sufficient diversity, which could lead to batches that do not contain enough information for effective pattern detection.

Grasping the importance of dataset size for result quality raises a crucial question: How can one determine if ones dataset is sufficiently large? This topic has been extensively researched to establish guidelines that help researchers assess dataset quality from a size perspective. While other ML algorithms, such as NNs, have developed effective rules of thumb for determining an adequate training dataset size (e.g., there should be at least 10 times more training instances than the number of weights in the network [89]), specific guidelines for GP are still lacking, as far as we know.

However, when assessing the feasibility of horizontal batching strategies, researchers in GP can utilize general rules of thumb from various ML fields. For instance, it is recommended that a the training dataset should have at least 10 to 100 times more training instances than the number of features [89] to consider the dataset to have an adequate size. Another valuable guideline involves the Entropy of the target function [74]. This principle suggests that the training dataset should include a number of instances at least equal to the target function's entropy (rounded up). For regression problems, this requires calculating the entropy by creating bins to group training instances into the same class. As highlighted in Sect. 3.2, Zoppi et al. [74] specifically applied this rule-of-thumb to reduce the number of training instances in GP.

Another important aspect to consider is determining at what point a dataset is regarded as "too" large, requiring batch training techniques to make GP feasible for real-world applications. While there is no definitive answer to this question, if we refer to the commonly accepted rule of thumb mentioned earlier, a dataset with a number of training instances exceeding 100 times the number of features could be considered large. This does not necessarily mean it cannot be processed without batch training, but given the previously discussed benefits, it may be worthwhile to explore such an approach. On a more quantifiable perspective, some of the works discussed in Sect. 3.1, regarding data parallelism in GP, considered datasets with more than several hundred training instances as large datasets (e.g., [72]). However, recent frameworks that take pride in their real-life data applicability [53, 56, 58], mention datasets a million or more instances. It is worth noting that a dataset does not need to be "large" for batch training applications to be considered. As previously mentioned, horizontal batch training methods not only meet computational demands but might also improve generalization by following a "less is more" approach (e.g., [7, 12]).

Finally, as previously mentioned, the number of attributes in a dataset significantly influences its quality [83]. It is well-known that reducing the number of features can enhance model performance by removing redundant or irrelevant data [33]. Therefore, researchers should consider making use of vertical batching techniques, especially when dealing with datasets that have a large feature space. The determination of whether a dataset possesses a large feature space can be linked to the previously mentioned rule-of-thumb. If the dataset contains fewer than 10

times the number of training instances compared to the number of features, it can be assumed that the dataset has a large feature space or high dimensionality. However, there are additional methods to ascertain the presence of the "curse of dimensionality" in the dataset and to understand its implications [90]. As previously mentioned, vertical batching, which involves feature selection methods, is outside the scope of this paper. For further details on these techniques, please refer to the work of Theng and Bhoyar [35].

4.2 Parallelizability of the desired implementation

As previously discussed, GP's "embarrassingly parallel" nature gave rise to a wide body of research focused developing data parallelism applications in GP [44, 45]. Despite their immense power, these applications face certain limitations in their use and development. Therefore, researchers must consider several critical factors that could limit their potential.

When utilizing parallelization, researchers need to account for the unique characteristics of their novel implementations, as these will significantly influence design and application requirements. For example, if the fitness evaluations of individuals within a population are interdependent, frequent synchronization of parallel processes might become necessary [91, 92]. This could reduce the benefits of parallelism or even cause synchronization issues and inconsistent results. Thus, researchers must be highly knowledgeable in this field and capable of developing or adapting advanced tools. When developing their applications, researchers must take into account the overall architecture of their implementations to ensure compatibility with parallelization. For instance, complex tree manipulations that necessitate deep recursive structures can be challenging to parallelize due to the potential need for sequential processing, which can introduce significant overhead when executed in parallel [93]. Recent applications have managed to address these challenges (e.g., [94–96]), although they still demand a high level of expertise and skills to effectively utilize parallelization in such scenarios.

4.3 Software availability

Considering the factors discussed in Sect. 4.2, researchers aiming to leverage data parallelism in GP must decide on the appropriate software. Developing new software tailored to a specific application demands significant expertise and thorough research to optimize the use of existing hardware. Conversely, selecting existing software necessitates careful evaluation to ensure compatibility with the application's design requirements and constraints, as well as assessing the software's modifiability and adaptability.

The recent study by Zeng and colleagues [97] offers a clear, comprehensive, and practical comparison of various parallelized GP computing platforms, evaluating them based on their features, designs, and performance. Additionally, the authors present a valuable summary of key considerations for selecting a parallelized GP computing platform, providing guidance for researchers in their future endeavors.

We aim to build upon the thorough research presented in [97] by providing readers with links to GP open source frameworks that can be used, adapted or taken into consideration when designing future research.

- GPOCL (C++), as discussed in [46]: https://gpocl.sourceforge.net/.
- TensorGP (Python), as presented in [19]: https://github.com/cdvetal/TensorGP.
- Operon (C++), as introduced in [45]: https://github.com/heal-research/operon.
- KarooGP (Python), proposed in [56]: https://github.com/kstaats/karoo_gp.
- CuML (Python), library that contains the GP CUDA based implementation proposed in [58]: https://github.com/rapidsai/cuml.

Besides the work by Zeng and colleagues [97], it is strongly recommended that readers review the recent GP framework speed benchmark comparison conducted by Baeta and collaborators [57]. This review will not only provide insights into the performance, structure, and composition of most of the open-source libraries discussed in this paper but also introduce some additional libraries, significantly aiding in software-related decisions.

4.4 Overall considerations

When designing batch training processes, researchers should consider their ability to effectively utilize data parallelism, their capacity to develop or adapt existing software, and the constraints of the datasets they aim to explore. We consider it important for researchers to remember that data parallelism does not always lead to faster execution times. We recommend that researchers try to be mindful of their computational resource limitations and ensure their implementations are compatible with parallel computation. Given the extensive literature supporting the potential of horizontal batch training methods, researchers should consider incorporating these strategies into their new implementations. However, they must ensure their datasets are sufficiently robust to prevent any loss of crucial information.

It is also important to note researchers seeking to design batch training applications in GP need not consider these decisions as independent. In fact, researchers can employ a "multi-level" approach, which involves integrating various batch training strategies to harness their combined benefits effectively. For example, the work of Franco and Bacardit [98] combined GPU parallelism alongside partial horizontal batching, yielding combined speedups about a thousand fold.

In conclusion, when implementing batch training in GP, researchers need to consider several key factors:

- The choice of software to be used based on its characteristics, adaptability and performance.
- The type of horizontal batch training to be employed, if applicable. Partial horizontal batch training is implementation-independent and can complement novel implementations, whereas full horizontal batch training typically results from novel algorithm design and strategy.

- The strategy for creating batches in horizontal batch training (i.e., the method for selecting which training instances are going to be used for each batch). We recommend consulting research on sampling techniques in GP, such as [18, 28]. Furthermore, we suggest looking into more recent dynamic approaches, such as those that aim to better manage the selection of training instances to preserve or enhance crucial evolutionary components, like the diversity or dissimilarity of the instances [99], or those that focus on retaining the most challenging cases in the batches [72] while also optimizing how frequently these batches are updated by adaptively adjusting the change rate based on the model's needs during learning [100].
- The choice between using multi-level batch training (i.e., more than one type of batch training approach) or a single-level approach. As noted earlier, integrating various forms of batch training appears promising in leveraging the advantages of different application types.
- Considerations regarding the batch size parameter. Researchers must decide on the nature of the batch size (static or dynamic) and its dimensions (small vs. large). Given the promising potential of dynamic batch size applications in GP and the relatively limited existing research, we suggest attempting to incorporate dynamic batch sizes. Dynamic batch sizes can explore a wider range of sizes, potentially reducing the pressure on this critical parameter. However, if a static batch size is preferred, based on the results from [25], we recommend using smaller batch sizes for better generalization ability.

5 Discussion and challenges

The previously discussed literature has brought to light an issue in the understanding and application of batch training within the realms of ML and GP, which may not have been thoroughly considered in this manner before.

As we transition towards the conclusion of this paper, it is crucial to revisit the challenges currently confronting this field as addressing these challenges is vital for the progression of research and the development of more effective and generalizable models. Additionally, proposing potential avenues for future research will not only highlight the unresolved issues but also pave the way for innovative solutions and advancements.

As mentioned earlier, the benefits of batch training have been extensively studied and widely adopted over the years. However, there hasn't been a clear, universal definition of what batch training actually means. This definition is important because it helps researchers connect related bodies of work. With this paper, we hope to address this challenge by offering a broader theoretical encapsulation of the term.

In our rapidly advancing technological world, data volumes are increasing, and models are becoming more complex to keep up with these changes. These issues have been discussed for decades and remain relevant today. For example, the 2024 overview by Guan et al. [101] on pipeline model parallelism (covering both data and model parallelization) for DNNs - one of the most prominent ML algorithms - illustrates the ongoing efforts to summarize, utilize, and update these bodies of work.

While this is true for other ML fields, GP research typically doesn't use the term batch training, making the term more associated with NNs and DNNs.

In this paper, we outline the extensive literature on batch training in GP. Given the potential of GP and its relatively recent emergence compared to NNs and DNNs, aligning GP with these bodies of literature and examining how similar concepts have been or could be applied is a significant challenge that still needs to be addressed. We believe this paper marks the beginning of that effort.

Beyond the recommendations and guidelines for designing batch training applications presented in Sect. 4, this survey aims to facilitate further research by proposing future directions. We suggest that researchers in the field of GP consider adapting techniques from other ML domains. For example, incorporating adaptive batch size strategies, as utilized in other ML algorithms, could be beneficial for batch training in GP. Additionally, the implementation of early stopping criteria to determine when to adjust batch sizes, as seen in CGP [76] and NNs [7, 11], is worth exploring.

Moreover, exploring the application of data complexity measurement for determining the batch size parameter, as demonstrated by Bacardit and collaborators [86], in GP applications could be highly insightful. Comparing these results with the findings of Zoppi, Vanneschi, and Giacobini [74] may reveal whether dataset complexity offers a more nuanced understanding of batch size requirements than merely considering the entropy of the target function. Additionally, future work comparing these approaches that aim for the "bare minimum" with more adaptive batch sizing methods seems particularly interesting and relevant.

In addition to adapting, exploring, and drawing inspiration from existing literature on other ML algorithms, we recommend that researchers leverage the unique intricacies and characteristics of GP to develop new batch training applications in this field. For instance, we believe that researchers in GP often do not fully leverage the potential of having a population at their disposal. One innovative way to utilize populations more effectively for GP-specific horizontal batching training could be to encode the information about the portion of the training set used for evaluating an individual directly within the individual's chromosome. In this approach, an individual could be viewed as a pair consisting of a program (the traditional representation in GP) and a subset of the training data. This allows each individual to be evaluated on a different segment of the data, effectively transforming the population into something that more closely resembles an ensemble algorithm. Moreover, by making these training set portions part of the individual's genome, they could evolve under the principles of artificial selection, potentially leading to the emergence of more specialized patterns for our solutions. This approach could offer a significant improvement over the more conventional use of random data batches. However, it also opens up a range of compelling research questions that future work should address. For instance, how should the selection algorithm compare individuals that have been evaluated on different subsets of data? How can crossover and mutation operators be extended to act on this new part of the genome, that encodes a portion of the training set? These questions present exciting opportunities for a new and potentially impactful research direction in GP.

Another promising research avenue involves leveraging the intrinsic parallelism of GP, such as through the multi-island GP model. While this concept is not entirely

new [73], the idea of utilizing different parts of the training set across various populations remains much open to further exploration and development. For example, new and more effective selection algorithms could be designed specifically for this type of multi-island model. Additionally, there is significant potential in developing improved mechanisms for the migration of individuals between islands, creating novel representations tailored to the multi-island framework, and exploring a range of other variations. These could even include asynchronous updates of the different partitions within each island based on predefined rules or evolving patterns. Such research directions could lead to substantial advancements in the effectiveness and adaptability of the multi-island GP model.

6 Conclusions

This literature review examined batch training applications in Genetic Programming (GP). We proposed a theoretical reformulation of the concept beyond the term "batch training" and introduced a new classification of batch training methods, framing the existing GP literature within this revised perspective. The studies reviewed show that batch training can greatly enhance both computational efficiency and algorithm performance in GP by enabling the handling of larger datasets and improving model generalization, depending on the specific batch training approach used. Our exploration reveals that while batch training is relatively well-studied in GP, the term itself is rarely used, leading to an unclear definition and a fragmented body of research. This fragmentation suggests that various studies, which could be viewed as parts of a unified methodology, remain disconnected. Notably, there is a lack of research on adaptive batch size training techniques in GP compared to other Machine Learning (ML) algorithms. This review also explores key factors for designing new batch training strategies in GP, such as data quality, implementation parallelizability, software decisions, and methods for selecting training cases for batches.

Future research should focus on refining the definition of batch training in GP to better align it with broader ML practices. It should prioritize the adaptation of techniques from other fields, such as dynamic batch sizing and early stopping criteria, to improve batch training methods in GP. Investigating data complexity measures for determining batch sizes could provide a more sophisticated approach than traditional methods. Additionally, novel approaches like encoding portions of the training set within individual genomes could lead to more specialized GP solutions. Enhancing the multi-island GP model with new selection algorithms, migration mechanisms, and other improvements could also increase its effectiveness and adaptability.

In conclusion, while GP research has not been isolated from the broader ML field, it has often used different terminology, leaving some ML batch learning techniques under-explored in the GP context. Integrating these techniques could bring GP closer to the standards of other ML algorithms. However, GP's unique characteristics offer opportunities to develop new, specialized batch training methods, as illustrated in works such as [73] and approaches like Lexicase selection [61].

Author contributions All authors contributed equally in conceptualization, methodology, investigation and writing.

Funding Open access funding provided by FCT|FCCN (b-on). This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia), under the project - UIDB/04152/2020 - Centro de Investigação em Gestão de Informação (MagIC)/NOVA IMS (https://doi.org/10.54499/UIDB/04152/2020).

Data availability Non applicable.

Declarations

Conflict of interest The authors have no conflict of interest as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

Ethical approval Non applicable.

References

1. J.R. Koza, *Genetic programming: on the programming of computers by means of natural selection* (MIT Press, Cambridge, MA, USA, 1992)
2. W.B. Langdon, Genetic programming convergence. Genet. Program. Evolvable Mach. **23**(1), 71–104 (2022)
3. M.E. Roberts, The effectiveness of cost based subtree caching mechanisms in typed genetic programming for image segmentation, in *Workshops on applications of evolutionary computation* (Springer, 2003), pp. 444–454
4. S. Luke, L. Panait, A comparison of bloat control methods for genetic programming. Evolut. Comput. **14**(3), 309–344 (2006)
5. I. Gonçalves, S. Silva, Balancing learning and overfitting in genetic programming with interleaved sampling of training data. in *European conference on genetic programming* (Springer, 2013), pp. 73–84
6. C.J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, G.E. Dahl, Measuring the effects of data parallelism on neural network training. J. Mach. Learn. Res. **20**, 1–49 (2019)
7. T. Takase, Dynamic batch size tuning based on stopping criterion for neural network training. Neurocomputing **429**, 1–11 (2021)
8. N.S. Keskar , D. Mudigere , J. Nocedal, M. Smelyanskiy, P.T.P. Tang, On large-batch training for deep learning: Generalization gap and sharp minima. in *International conference on learning representations* (2017)
9. P.M. Radiuk, Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. Inf. Technol. Manag. Sci. **20**(1), 20–24 (2017)
10. A. Devarakonda , M. Naumov, M. Garland, Adabatch, adaptive batch sizes for training deep neural networks. in *International conference on learning representations* (2017)
11. L. Balles, J. Romero, P. Hennig, Coupling adaptive batch sizes with learning rates. ArXiv http://arxiv.org/abs/1612.05086 (2016)

12. M. Papini, M. Pirotta, M. Restelli, Adaptive batch size for safe policy gradients. Adv. Neural Inf. Process. Syst. **30** (2017)
13. V.V. De Melo, D.V. Vargas, W. Banzhaf, Batch tournament selection for genetic programming: the quality of lexicase, the speed of tournament. in *Proceedings of the genetic and evolutionary computation conference*, pp. 994–1002 (2019)
14. L. Spector, Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. in *Proceedings of the 14th annual conference companion on genetic and evolutionary computation*, pp. 401–408 (2012)
15. T. Helmuth, L. Spector, J. Matheson, Solving uncompromising problems with lexicase selection. IEEE Trans. Evolut. Comput. **19**(5), 630–643 (2014)
16. D. Song, M.I. Heywood, A.N. Zincir-Heywood, Training genetic programming on half a million patterns: an example from anomaly detection. IEEE Trans. Evolut. Comput. **9**(3), 225–239 (2005)
17. L. Bote-Curiel, S. Munoz-Romero, A. Gerrero-Curieses, J.L. Rojo-Álvarez, Deep learning and big data in healthcare: a double review for critical beginners. Appl. Sci. **9**(11), 2331 (2019)
18. I. Gonçalves, S. Silva, J.B. Melo, J.M. Carreiras, Random sampling technique for overfitting control in genetic programming. in *Genetic programming: 15th european conference, EuroGP 2012, Málaga, Spain, April 11–13, 2012*, (Springer, 2012), pp. 218–229
19. F. Baeta, J. Correia, T. Martins, P. Machado, Tensorgp–genetic programming engine in tensorflow. In: *applications of evolutionary computation: 24th international conference, evoapplications 2021, held as part of EvoStar 2021, virtual event, April 7–9, 2021, Proceedings 24*, pp. 763–778 (2021)
20. Y. Ma, F. Rusu, K. Wu, A. Sim Adaptive stochastic gradient descent for deep learning on heterogeneous cpu+ gpu architectures. in *2021 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pp. 6–15 (2021)
21. S. McCandlish, J. Kaplan, D. Amodei, O.D. Team, An empirical model of large-batch training. ArXiv http://arxiv.org/abs/1812.06162 (2018)
22. M. Li, T. Zhang, Y. Chen, A.J. Smola, Efficient mini-batch training for stochastic optimization. in *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 661–670 (2014)
23. K. Do, D. Nguyen, H. Nguyen, L. Tran-Thanh, Q.-V. Pham, Revisiting lars for large batch training generalization of neural networks. arXiv preprint arXiv:2309.14053 (2023)
24. L. Breiman, Random forests. Mach. Learn. **45**, 5–32 (2001)
25. I.C.P. Gonçalves, An exploration of generalization and overfitting in genetic programming: standard and geometric semantic approaches. PhD thesis, Universidade de Coimbra (Portugal) (2016)
26. P. Okanovic, R. Waleffe, V. Mageirakos, K.E. Nikolakakis, A. Karbasi, D. Kalogerias, N.M. Gürel, T. Rekatsinas, Repeated random sampling for minimizing the time-to-accuracy of learning. arXiv preprint arXiv:2305.18424 (2023)
27. P. Birzhandi, K.T. Kim, H.Y. Youn, Reduction of training data for support vector machine: a survey. Soft Comput. **26**(8), 3729–3742 (2022)
28. H. Hmida, S.B. Hamida, A. Borgi, M. Rukoz, Sampling methods in genetic programming learners from large datasets: a comparative study. in *Advances in big data: proceedings of the 2nd INNS conference on big data, October 23–25, 2016*, (Springer, Thessaloniki, Greece, 2017), pp. 50–60
29. J. Yu, M. Ai, Z. Ye, A review on design inspired subsampling for big data. Stat. Pap. **65**(2), 467–510 (2024)
30. Y. Yao, H. Wang, A review on optimal subsampling methods for massive datasets. J. Data Sci. **19**(1), 151–172 (2021)
31. H. Avron, V. Sindhwani, D. Woodruff, Sketching structured matrices for faster nonlinear regression. Adv. Neural Inf. Process. Syst. **26** (2013)
32. J. Li, K. Cheng, S. Wang, F. Morstatter, R.P. Trevino, J. Tang, H. Liu, Feature selection: a data perspective. ACM Comput. Surv. (CSUR) **50**(6), 1–45 (2017)
33. B. Venkatesh, J. Anuradha, A review of feature selection and its methods. Cybernetics and information technologies **19**(1), 3–26 (2019)
34. S.S. Subbiah, J. Chinnappan, Opportunities and challenges of feature selection methods for high dimensional data: a review. Ingénierie des Systèmes d'Information **26**(1), 67–77 (2021)
35. D. Theng, K.K. Bhoyar, Feature selection techniques for machine learning: a survey of more than two decades of research. Knowl. Inf. Syst. **66**(3), 1575–1637 (2024)

36. S.L. Smith, P.-J. Kindermans, Q.V. Le, Don't decay the learning rate, increase the batch size. in *International conference on learning representations* (2018)
37. M.S. Alsadi, R. Ghnemat, A. Awajan, Accelerating stochastic gradient descent using adaptive mini-batch size. in *2019 2nd International Conference on New Trends in Computing Sciences (ICTCS)*, pp. 1–7 (2019)
38. Q. Ye, Y. Zhou, M. Shi, Y. Sun, J. Lv, Dbs: Dynamic batch size for distributed deep neural network training. arXiv preprint arXiv:2007.11831 (2020)
39. C. Chen, Q. Weng, W. Wang, B. Li, Fast distributed deep learning via worker-adaptive batch sizing. in *Proceedings of the ACM symposium on cloud computing*, pp. 521–521 (2018)
40. K.A. Alnowibet, I. Khan, K.M. Sallam, A.W. Mohamed, An efficient algorithm for data parallelism based on stochastic optimization. Alex. Eng. J. **61**(12), 12005–12017 (2022)
41. Q. Ye, Y. Zhou, M. Shi, J. Lv, FLSGD: free local SGD with parallel synchronization. J. Supercomput. **78**(10), 12410–12433 (2022)
42. H. Yu, R. Jin, On the computation and communication complexity of parallel sgd with dynamic batch sizes for stochastic non-convex optimization. in *International conference on machine learning*, pp. 7174–7183 (2019)
43. S. Lee, Q. Kang, S. Madireddy, P. Balaprakash, A. Agrawal, A. Choudhary, R. Archibald, W.-k. Liao, Improving scalability of parallel cnn training by adjusting mini-batch size at run-time. in *2019 IEEE international conference on big data (big data)*, pp. 830–839 (2019)
44. D. Andre, J.R. Koza, Parallel genetic programming: a scalable implementation using the transputer network architecture. Adv. Genet. Program. **2**, 317–338 (1996)
45. B. Burlacu, G. Kronberger, M. Kommenda, Operon c++ an efficient genetic programming framework for symbolic regression. in *Proceedings of the 2020 genetic and evolutionary computation conference companion*, pp. 1562–1570 (2020)
46. A. Cano, S. Ventura, Gpu-parallel subtree interpreter for genetic programming. in *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pp. 887–894 (2014)
47. S. Harding, W. Banzhaf, Fast genetic programming on gpus. in *Genetic programming: 10th european conference, EuroGP 2007, Valencia, Spain, April 11–13, 2007. Proceedings 10*, (Springer, 2007), pp. 90–101
48. D.A. Augusto, H.J. Barbosa, Accelerated parallel genetic programming tree evaluation with opencl. J. Parallel Distrib. Comput. **73**(1), 86–100 (2013)
49. P. Tufts, Parallel case evaluation for genetic programming. (1993)
50. W.B. Langdon, W. Banzhaf, A simd interpreter for genetic programming on gpu graphics cards. in *European conference on genetic programming*, (Springer, 2008), pp. 73–85
51. D.M. Chitty, A data parallel approach to genetic programming using programmable graphics hardware. in *Proceedings of the 9th annual conference on genetic and evolutionary computation*, pp. 1566–1573 (2007)
52. D. Tarditi, S. Puri, J. Oglesby, Accelerator: using data parallelism to program gpus for general-purpose uses. ACM SIGPLAN Not. **41**(11), 325–335 (2006)
53. W.B. Langdon, A.P. Harrison, GP on SPMD parallel graphics hardware for mega bioinformatics data mining. Soft Comput. **12**, 1169–1183 (2008)
54. W.B. Langdon, Graphics processing units and genetic programming: an overview. Soft Comput. **15**, 1657–1669 (2011)
55. D.M. Chitty, Faster GPU-based genetic programming using a two-dimensional stack. Soft Comput. **21**, 3859–3878 (2017)
56. K. Staats, E. Pantridge, M. Cavaglia, I. Milovanov, A. Aniyan, Tensorflow enabled genetic programming. in *Proceedings of the genetic and evolutionary computation conference companion*, pp. 1872–1879 (2017)
57. F. Baeta, J. Correia, T. Martins, P. Machado, Speed benchmarking of genetic programming frameworks. in *Proceedings of the Genetic and evolutionary computation conference*, pp. 768–775 (2021)
58. V. Sathia, V. Ganesh, S.R.T.Nanditale, Accelerating genetic programming using gpus. arXiv preprint arXiv:2110.11226 (2021)
59. S. Raschka, J. Patterson, C. Nolet, Machine learning in python: main developments and technology trends in data science, machine learning, and artificial intelligence. Information **11**(4), 193 (2020)
60. S. Aenugu, L. Spector, Lexicase selection in learning classifier systems. in *Proceedings of the genetic and evolutionary computation conference*, pp. 356–364 (2019)

61. L. Spector Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. in *Proceedings of the 14th annual conference companion on genetic and evolutionary computation*, pp. 401–408 (2012)
62. L. Ding, E. Pantridge, L. Spector, Probabilistic lexicase selection. in *Proceedings of the genetic and evolutionary computation conference*, pp. 1073–1081 (2023)
63. Y. Martínez, E. Naredo, L. Trujillo, P. Legrand, U. Lopez, A comparison of fitness-case sampling methods for genetic programming. J. Exp. Theor. Artif. Intell. **29**(6), 1203–1224 (2017)
64. T. Helmuth, L. Spector, Problem-solving benefits of down-sampled lexicase selection. Artif. Life **27**(3–4), 183–203 (2022)
65. P. Liskowski, K. Krawiec, T. Helmuth, L. Spector, Comparison of semantic-aware selection methods in genetic programming. in *Proceedings of the Companion publication of the 2015 annual conference on genetic and evolutionary computation*, pp. 1301–1307 (2015)
66. T. Helmuth, N.F. McPhee, L. Spector, Lexicase selection for program synthesis: a diversity analysis. Genet. Program. Theory Pract. **XIII**, 151–167 (2016)
67. J.M. Moore, A. Stanton, Tiebreaks and diversity: isolating effects in lexicase selection, in *Artificial life conference proceedings*. (MIT Press, Cambridge, MA, USA, 2018), pp.590–597
68. W. La Cava, L. Spector, K. Danai, Epsilon-lexicase selection for regression. in *Proceedings of the genetic and evolutionary computation conference*, pp. 741–748 (2016)
69. R. Boldi, M. Briesch, D. Sobania, A. Lalejini, T.Helmuth, F. Rothlauf, C. Ofria, L. Spector, Informed down-sampled lexicase selection: identifying productive training cases for efficient problem solving. Evolut. Comput., 1–32 (2024)
70. J.G. Hernandez, A. Lalejini, E. Dolson, C. Ofria, Random subsampling improves performance in lexicase selection. in *Proceedings of the genetic and evolutionary computation conference companion*, pp. 2028–2031 (2019)
71. C. Gathercole, P. Ross, Dynamic training subset selection for supervised learning in genetic programming. in *Parallel problem solving from nature-PPSN III: international conference on evolutionary computation the third conference on parallel problem solving from nature Jerusalem, Israel, October 9–14, 1994 Proceedings 3*, (Springer, 1994), pp. 312–321
72. R., Curry, M. Heywood, Towards efficient training on large datasets for genetic programming. in *Advances in artificial intelligence: 17th conference of the canadian society for computational studies of intelligence, Canadian AI 2004, London, Ontario, Canada, May 17–19, 2004. Proceedings 17*, (Springer, 2004), pp. 161–174
73. M. Kommenda, M. Affenzeller, B. Burlacu, G. Kronberger, S.M. Winkler, Genetic programming with data migration for symbolic regression. in *Proceedings of the companion publication of the 2014 annual conference on genetic and evolutionary computation*, pp. 1361–1366 (2014)
74. G. Zoppi, L. Vanneschi, M. Giacobini, Reducing the number of training cases in genetic programming. in *2022 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8 (2022)
75. H., Hmida, S.B. Hamida, A. Borgi , M. Rukoz, Sampling methods in genetic programming learners from large datasets: a comparative study. in *Advances in big data: proceedings of the 2nd INNS conference on big data, October 23–25, 2016, Thessaloniki, Greece 2*, (Springer, 2017), pp. 50–60
76. B.M. Lima, N.Sachetti, A. Berndt, C. Meinhardt, J.T. Carvalho, Adaptive batch size cgp: improving accuracy and runtime for cgp logic optimization flow. in *European conference on genetic programming (part of EvoStar)*, (Springer, 2023), pp. 149–164
77. M.S. Alsadi, R. Ghnemat, A. Awajan, Accelerating stochastic gradient descent using adaptive mini-batch size. in *2019 2nd international conference on new trends in computing sciences (ICTCS)*, pp. 1–7 (2019).
78. W. Khan, S. Ali, U.K. Muhammad, M. Jawad, M. Ali, R. Nawaz Adadiffgrad: an adaptive batch size implementation technique for diffgrad optimization method. in *2020 14th International conference on innovations in information technology (IIT)*, pp. 209–214 (2020)
79. Y. Gong, G. Liu, Y. Xue, R. Li, L. Meng, A survey on dataset quality in machine learning. Inf. Softw. Technol. **162**, 107268 (2023)
80. C. Northcutt, L. Jiang, I. Chuang, Confident learning: estimating uncertainty in dataset labels. J. Artif. Intell. Res. **70**, 1373–1411 (2021)
81. Northcutt, C.G., Athalye, A., Mueller, J.: Pervasive label errors in test sets destabilize machine learning benchmarks. arXiv preprint arXiv:2103.14749 (2021)
82. V. Gudivada, A. Apon, J. Ding, Data quality considerations for big data and machine learning: going beyond data cleaning and transformations. Int. J. Adv. Softw. **10**(1), 1–20 (2017)

83. S. Uddin, H. Lu, Dataset meta-level and statistical features affect machine learning performance. Sci. Rep. **14**(1), 1670 (2024)
84. A.C. Lorena, A.I. Maciel, P.B. Miranda, I.G. Costa, R.B. Prudêncio, Data complexity meta-features for regression problems. Mach. Learn. **107**, 209–246 (2018)
85. F. Thabtah, S. Hammoud, F. Kamalov, A. Gonsalves, Data imbalance in classification: experimental evaluation. Inf. Sci. **513**, 429–441 (2020)
86. J. Bacardit, D.E. Goldberg, M.V. Butz, X. Llora, J.M Garrell, Speeding-up pittsburgh learning classifier systems: modeling time and accuracy. in *Parallel problem solving from nature-PPSN VIII: 8th international conference, Birmingham, UK, September 18-22, 2004. Proceedings 8*, (Springer, 2004), pp. 1021–1031
87. L. Bottou, O. Bousquet, The tradeoffs of large scale learning. Adv. Neural Inf. Process. Syst. **20** (2007)
88. A. Vabalas, E. Gowen, E. Poliakoff, A.J. Casson, Machine learning algorithm validation with a limited sample size. PloS one **14**(11), 0224365 (2019)
89. A. Alwosheel, S. Cranenburgh, C.G. Chorus, Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis. J. Choice Modell. **28**, 167–182 (2018)
90. E. Debie, K. Shafi, Implications of the curse of dimensionality for supervised learning classifier systems: theoretical and empirical analyses. Pattern Anal. Appl. **22**, 519–536 (2019)
91. F. Ferrucci, P. Salza, F. Sarro, Using hadoop mapreduce for parallel genetic algorithms: a comparison of the global, grid and island models. Evolut. Comput. **26**(4), 535–567 (2018)
92. R. Rivers, A.R. Bertels, D.R. Tauritz, Asynchronous parallel evolutionary algorithms: leveraging heterogeneous fitness evaluation times for scalability and elitist parsimony pressure. in *Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation*, pp. 1429–1430 (2015)
93. D. Bartz, W. Straßer, Asynchronous parallel construction of recursive tree hierarchies. in *Parallel computation: 4th international ACPC Conference including special tracks on parallel numerics (parnum'99) and parallel computing in image processing, video processing, and multimedia Salzburg, Austria, February 16–18, 1999 Proceedings 4*, (Springer, 1999), pp. 427–436
94. A. Fonseca, B. Cabral, J. Rafael, I. Correia, Automatic parallelization: executing sequential programs on a task-based parallel runtime. Int. J. Parallel Program. **44**, 1337–1358 (2016)
95. G. Chennupati, R.M.A. Azad, C. Ryan, Automatic evolution of parallel recursive programs. in *Genetic programming: 18th european conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings 18*, (Springer, 2015), pp. 167–178.
96. Q.I. Mahmud, A. TehraniJamsaz, H.D. Phan, N.K. Ahmed, A. Jannesari, Autoparllm: Gnn-guided automatic code parallelization using large language models. arXiv preprint arXiv:2310.04047 (2023)
97. R. Zeng, Z. Huang, Y. Chen, J. Zhong, L. Feng, Comparison of different computing platforms for implementing parallel genetic programming. in *2020 IEEE congress on evolutionary computation* (CEC), pp. 1–8 (2020)
98. M.A. Franco, J. Bacardit, Large-scale experimental evaluation of GPU strategies for evolutionary machine learning. Inf. Sci. **330**, 385–402 (2016). https://doi.org/10.1016/j.ins.2015.10.025
99. K.K. Gupt, M., Kshirsagar, D.M. Dias, , J.P. Sullivan, C. Ryan, Adaptive case selection for symbolic regression in grammatical evolution. in *IJCCI*, pp. 195–205 (2023)
100. S.B. Hamida, H. Hmida, A. Borgi, M. Rukoz, Adaptive sampling for active learning with genetic programming. Cogn. Syst. Res. **65**, 23–39 (2021)
101. L. Guan, D.-S. Li, J.-Y. Liang, W.-J. Wang, K.-S. Ge, X.-C. Lu, Advances of pipeline model parallelism for deep learning training: an overview. J. Comput. Sci. Technol. **39**(3), 567–584 (2024)