Reducing Energy of Approximate Feature Extraction in Heterogeneous Architectures for Sensor Inference via Energy-Aware Genetic Programming

Yinqi Tang, *Student Member, IEEE*, Hongyang Jia, *Student Member, IEEE*, and Naveen Verma, *Member, IEEE*

***Abstract*— Hardware acceleration substantially enhances both energy efficiency and performance, but raises major challenges for programmability. This is especially true in the domain of approximate computing, where energy-approximation tradeoffs at the hardware level are extremely difficult to encapsulate in interfaces to the software level. The programmability challenges have motivated co-design of accelerators with program-synthesis frameworks, where the structured computations resulting from synthesis are exploited towards hardware specialization. This paper proposes energy-aware code synthesis targeting heterogeneous architectures for approximate computing. A heterogeneous architecture for embedded sensor inference is employed, demonstrated in custom silicon, where programmable feature extraction is mapped to an accelerator via genetic programming. The high level of accelerator specialization and structured mapping of computations to the accelerator enable robust energy models, which are then employed in a genetic-programming algorithm to improve the energy-approximation Pareto frontier. The proposed algorithm is demonstrated in an electroencephalogram-based seizure-detection application and an electrocardiogram-based arrhythmia-detection application. At the same level of baseline inference performance, the energy consumption of genetic-programming models executed on the accelerator is 57.4% and 21.8% lower, respectively, with the proposed algorithm, compared to a conventional algorithm without incorporating energy models for execution on the accelerator.**

***Index Terms*— Approximate computing, energy efficiency, feature extraction, genetic programming, Pareto optimization.**

I. Introduction

THE explosion in embedded sensing applications, both in terms of the different kinds of signals of interest and the complexity of signal analysis required, has driven the need for increasingly specialized and sophisticated platforms. In particular, signal-analysis algorithms based on machine learning (ML) have become standard across applications [1], [2]. While deep-learning methods are playing a prominent role [3], [4], insights about mechanisms and embedded processes generating signals, and thus an understanding of how to model signals, have made methods based on hand-tuned features of high importance in many applications [5], [6].

However, the stringent energy constraints of sensor platforms and the increasing complexity of signal-analysis computations have posed a primary challenge for embedded systems. To address this, heterogeneous architectures based on accelerators are playing a critical role. Furthermore, the statistical nature of embedded signals and processes, and the resulting statistical nature of analysis, have opened the opportunity for energy reduction through approximate computing [7], where the approximation of final or intermediate results has shown to enable substantial gains in energy efficiency. While many ML systems exhibit some level of tolerance to compute errors [8], statistical optimization, intrinsic to ML algorithms, has further enabled approximation via optimization incorporating the statistics of the hardware itself [9], [10].

The primary challenge with hardware acceleration, further exacerbated by approximation, is the limitations imposed on programmability, especially across diverse applications. A key direction for addressing this has been to push specialization to the application-design level, but through constructs that reduce the programming burden on developers. Examples of this include domain-specific languages (e.g., Julia [11], Halide [12], Tensorflow [13]), program sketching [14], and program synthesis [15], where constructs provided to ease programming also enforce structure in the programs, which can be exploited for mapping to hardware. Program synthesis, in particular, can go so far as to reduce programming to providing input-output pairs for computation, from which a program can then be automatically developed.

Recent work has exploited the program-synthesis framework of genetic programming (GP) [16], [17], together with accelerator co-design, to achieve a programmable heterogeneous platform for approximate computing in sensor inference [18]. Structured models of computation, used to enable evolutionary algorithms for synthesis of application programs, were exploited towards a high level of hardware specialization in accelerators. In this paper, the hardware specialization and structured mapping of computations are exploited to form robust energy models for execution on the accelerator, which

Manuscript received April 2, 2019; revised September 26, 2019 and December 16, 2019; accepted December 19, 2019. Date of publication January 8, 2020; date of current version May 1, 2020. This article was recommended by Associate Editor F. J. Kurdahi. *(Corresponding author: Yinqi Tang.)*

The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: yinqit@princeton.edu; hjia@princeton.edu; nverma@princeton.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2019.2961643

are then incorporated in the program-synthesis algorithm to enhance the energy-approximation Pareto frontier. The specific contributions of this paper are as follows.

1) We develop an energy model for execution of structured computations on a programmable accelerator, using [18] for concrete demonstration of a complete heterogeneous microprocessor architecture implemented in a 130-nm CMOS process. The energy model is specifically developed in a form suitable for enabling energy-aware GP. This institutes a relationship between the computations in a programmable framework and the system resources (energy) required for their realization.
2) We develop an energy-aware GP algorithm. This incorporates the accelerator-execution energy model within the GP statistical-optimization objective function, in a manner that dynamically alternates between emphasizing model energy and model expressional complexity [19], in order to simultaneously improve the execution energy and maintain generalization ability of GP models. Doing so extends Pareto-frontier exploitation for code synthesis, explicitly exposing a designer trade space spanning statistical requirements at the application level and system-resource (energy) requirements from the underlying hardware level.
3) The proposed energy-aware GP algorithm is demonstrated on two practical medical-sensor applications: (1) electroencephalogram (EEG)-based seizure detection; and (2) electrocardiogram (ECG)-based arrhythmia detection. For seizure detection, at the same level of baseline inference performance (100% sensitivity, 3.81 seconds latency, and 0.05 false alarms per hour), the energy consumption of GP models executed on the accelerator is 57.4% lower than that derived from a conventional GP algorithm. For arrhythmia detection, at the same level of baseline inference performance (82.05% sensitivity, 88.12% specificity, and 87.92% accuracy), the energy consumption of GP models executed on the accelerator is 21.8% lower than that that derived from a conventional GP algorithm.

The remainder of this paper is organized as follows. Section II provides background on the GP algorithm, the heterogeneous microprocessor [18] used for this study, and approaches that have been taken to reduce the complexity of GP models. Section III describes the proposed energy-aware GP algorithm. Section IV presents application-demonstration results of both the conventional and proposed GP algorithms. Finally, Section V provides conclusions.

II. Background

A. Genetic-Programming (GP) Algorithm

GP is a biologically-inspired ML technique [16], [17], which creates a definite and structured predictive function (mathematical model) that approximates the relationship between given inputs and outputs. The main characteristic of GP is that both the model structure and the related parameters are automatically evolved. The model is generally represented as a tree structure consisting of functions and terminals. The functions are chosen from a pre-defined function set (e.g., addition, multiplication, exponential, logarithm, sine). The terminals, which contain the arguments for the functions, can be either input variables or constants. In this paper, we employ an efficient GP variant, namely multi-gene GP (MGGP) [20]. In MGGP, a tree structure is called a gene. Each GP model consists of several genes. The gene outputs are scaled and linearly combined to generate the output of the GP model, as in the following equation:

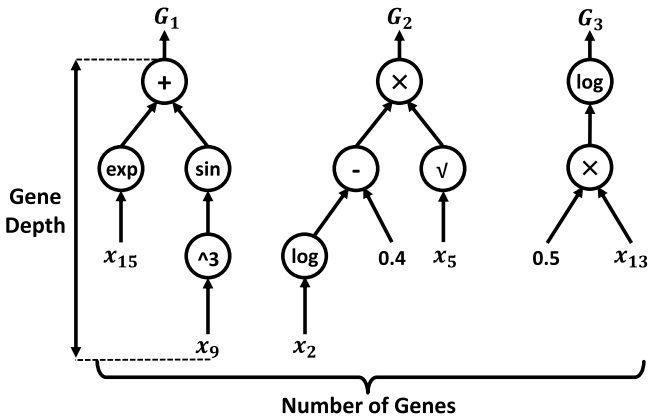


Fig. 1. An example tree-based GP model: $G_1 = e^{x_{15}} + sin(x_9^3)$, $G_2 = (log(x_2) - 0.4)\sqrt{x_5}$, and $G_3 = log(0.5x_{13})$. x_i denotes an input variable.

$$y_{GP} = \alpha_0 + \beta_1 G_1 + \beta_2 G_2 + \cdots + \beta_M G_M, \tag{1}$$

where α_0 denotes a bias term, β_n denotes the gene weight associated with the gene output G_n, M denotes the total number of genes comprising the GP model, and y_{GP} is the output of the GP model. The bias term and the gene weights are derived using the linear-least-squares fitting technique, similar to typical linear regression. In this way, MGGP integrates model-structure evolution of GP with parameter estimation of linear regression, to represent the nonlinear behaviors relating the given inputs and outputs. An example tree-based GP model is illustrated in Fig. 1.

The main procedure of the GP algorithm is performed in a loop comprising model generation, model evaluation, and model selection. Specifically, a random population of GP models is created initially to achieve high diversity. The population is then evolved based on the survival-of-the-fittest principle over several generations. For each generation, GP evaluates and selects models to create a new population, which potentially contains better models. Each new model can be generated by modifying a set of selected models in the current population, using one of three genetic operators: mutation, crossover, and reproduction. Mutation refers to randomly mutating part of the selected model. Crossover refers to swapping parts of two selected models at random, thus generating two new models. Reproduction refers to copying an unchanged model into the next generation. Conventionally, the model providing the highest fitness value defines the output of GP. In this paper, we use the common coefficient of

determination (R^2) for the fitness evaluation, as shown below:

$$R^2 = \left(1 - \frac{\sum_{i=1}^{N}(y_i - y_{i,GP})^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}\right) \times 100\%, \qquad (2)$$

where y_i is a given output value, $y_{i,GP}$ is the corresponding GP-predicted output value, N is the total number of outputs, and $\bar{y}$ is the mean of all N given outputs. R^2 denotes the proportion of the variation in the output values explained by the GP model. Higher R^2 indicates a better GP model.

B. Heterogeneous Microprocessor Exploiting GP

A typical inference system consists of two stages: (1) feature extraction; and (2) classification. Feature extraction transforms the original inputs into a new representation, to enhance the generalization for making specific desired decisions in the subsequent classification stage. For instance, this may correspond to amplifying and/or making more explicit variances in a signal that have strong correspondence with the decision of interest, while suppressing variances that have weak or no correspondence with the decision of interest. Classification involves the application of a model to make decisions about the inputs based on the extracted features.

In energy-constrained platforms, hardware acceleration is of interest for both feature extraction and classification. For instance, in the case of deep learning (neural networks), feature extraction can dominate computations, thus motivating considerable research on hardware acceleration [21]. However, while classification is typically performed through comparatively fixed-function kernels (support vector machines (SVMs), fully-connected neural networks, decision trees, etc.), feature extraction poses the challenge that it requires a high level of programmability, because its computations strongly depend on the application inputs and decisions of interest. With classification readily delegated to highly-specialized accelerators, feature extraction is observed to dominate energy even in the case of hand-tuned features [22]. Addressing this thus raises the difficult challenge of programmable acceleration.

In [18], this challenge is addressed for embedded sensing applications by exploiting GP. Many embedded sensing applications (e.g., medical, audio, industrial) rely on hand-tuned features. As described above, GP enables automatic program synthesis for feature extraction, based on specific tree-structured models of computation, which is exploited for accelerator specialization. Fig. 2 shows the architecture and chip die photo (implemented in 130-nm CMOS) of the resulting heterogeneous microprocessor.

Some important blocks include: (1) a simple CPU (MSP430 instruction set) for top-level control; (2) a direct-memory-access (DMA) module for automated data movement between peripherals and accelerators without CPU intervention; (3) a four-core programmable feature extraction accelerator (FEA) for executing GP models; and (4) a configurable SVM accelerator for classification, which can support various SVM kernels (linear, polynomial, and radial basis function (RBF)). The accelerators and other modules communicate with the CPU via a peripheral bus. Based on the MGGP formulation, the four gene-computation (GC) cores in the FEA

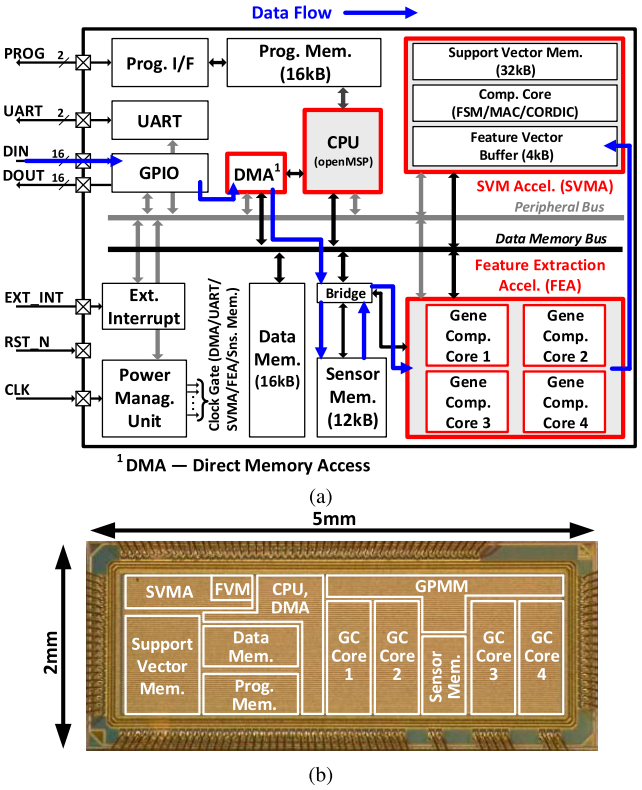


Fig. 2. (a) Architecture of the heterogeneous microprocessor for sensor inference [18]. (b) Chip die photo of the heterogeneous microprocessor [18].

compute gene trees for a GP model in parallel, and apply the corresponding scaling factors (gene weights) to their outputs. The outputs of all gene trees are then combined by a GP model manager (GPMM) within the FEA to derive the final output of the GP model. Detailed analysis of circuit design is provided in [18], while this work focuses more on a co-designed algorithm for GP-model generation. For two medical-sensor applications demonstrated, the FEA was shown to result in 325× and 156× energy reduction for programmable feature extraction, compared to implementation on the programmable CPU [18].

The statistical nature of GP makes it a form of approximate computing. Indeed, the inherent tolerance to approximation exhibited by many ML applications has enabled the use of GP in many such systems [23]–[26]. However, approaches have also emerged that explicitly exploit the attribute of statistical optimization intrinsic to ML algorithms, in order to adapt inference-model parameters not only to the statistics of application data, but also the statistics of approximate computing [9], [10]. For instance, in [9], computational errors in feature extraction are overcome by adapting the classification-model parameters to the feature-vector distributions that arise as a result of the computational errors, yielding what is referred to as an *error-aware model*. This substantially enhances the applicability of approximate computing in ML systems. In fact, [9] shows that error tolerance is achieved up to levels limited by the fundamental information retained in the resulting distributions for a classification task.

Such applicability can be exploited towards aggressive energy scalability. In particular, GP enables a tradeoff between approximation and computational complexity, by allowing constraints on the generated GP models (types of functions, number of genes, depth of trees). This was exploited in [18] in an indirect and limited way, by placing such constraints, presuming resulting energy savings, and then applying the conventional GP algorithm, to achieve a corresponding level of accuracy. In this paper, the dual objectives of low energy and model expression for high accuracy are integrated within the GP-model optimization objective function. This involves forming robust energy models for program execution on the accelerator and alternating emphasis on model energy and model expressional complexity [19], during the optimization process. The result is accelerator performance that achieves significantly enhanced energy-approximation Pareto frontier.

C. Reducing Complexity of GP Models

As shown in Fig. 1, the complexity of GP models is affected by choice of the pre-defined function set, the maximum number of genes allowed for each model (G_{max}), and the maximum tree depth allowed for each gene (D_{max}). G_{max} and D_{max} in particular directly influence the size of the search space (inherently large) and the number of possible solutions explored by GP. Increasing G_{max} or D_{max} thus increases the performance of generated GP models, but at the expense of significantly increased model-optimization time and model-execution complexity. A particularly important problem, prominent in the case of MGGP, is that the generated models may contain some overly-complex genes, which contribute little to fitness. This phenomenon is referred to as *bloat* in GP [27]–[29]. As complex GP models generally are also energy-intensive, there exists a need for regularization of model complexity. We are interested in the GP models that are energy-efficient and limited in complexity, sometimes in favor of models that may enable the highest fitness values.

There are many approaches proposed in the literature to reduce the complexity of GP models [28], [29]. One possible solution is to apply parsimony pressure on model complexity. Specifically, a term of model complexity is included in the fitness metric as a form of regularization. The problem with such an approach of one single composite metric is that the fitness-complexity tradeoff is often problem-dependent and hard to define (difficult to choose a proper scaling factor for two competing objectives) prior to the model discovery process. To overcome this, we employ the concept of Pareto optimization [19], [30], which simultaneously optimizes model fitness and model complexity by improving the Pareto frontier. The key aspects of Pareto optimization and how they are utilized will be explained in detail in Section III.

III. Energy-Aware GP Algorithm

In this section, we describe the proposed energy-aware GP algorithm, with emphasis on aspects different from the conventional GP algorithm. This includes developing an energy model for execution on the accelerator, employing Pareto optimization to improve the energy-approximation Pareto frontier, alternating model complexity objective during the optimization process, and post-processing on GP models.

Algorithm 1 Conventional GP Algorithm

```
Initialize a random population
for t in 1 to T_g do
    Evaluate fitness of all models
    Elitism selection in population for N_e models
    Initialize N ← 0
    while N < N_p − N_e do
        Randomly select a genetic operator
        Tournament selection in population
        Generate new model(s)
        Check constraints of G_max and D_max
        Update N ← N + 1 or N ← N + 2
    Combine all N_p models for a new population
```

A. Algorithm Overview

Algorithm 1 summarizes the main steps of the conventional GP algorithm, where T_g denotes the total number of generations to evolve models, N_p denotes the number of models in the population, and N_e denotes the number of elite models preserved for the next generation. Elitism selection at the beginning of each generation helps maintain a small fraction of models with the highest fitness values, because the improvement of models through mutation or crossover is not guaranteed. The remaining $(N_p - N_e)$ models are generated using genetic operators based on tournament selection in each generation. Specifically, to select one model for breeding, a group of N_t models (N_t is the tournament size and typically much smaller than N_p) is randomly selected from the current population, and the model with the highest fitness value is finally chosen. Therefore, tournament selection ensures that models with higher fitness values are more likely to participate in building models in the following generations. This, together with elitism selection, ensures the convergence to the optimal solution with enough generations. However, as is typical with genetic algorithms, the rate of convergence and the value of the optimal solution (and the gap between it and the current solution) are not known in each generation. For mutation and reproduction, one new model is generated each time. For crossover, two new models are generated each time. All new models need to be checked against model constraints of G_{max} and D_{max}.

Algorithm 2 summarizes the main steps of the proposed energy-aware GP algorithm, with the key differences from the conventional GP algorithm shown in bold. As seen, in addition to model fitness, model complexity is evaluated for each generated model, which dynamically alternates between model energy and model expressional complexity [19] for better generalization. This alternating-objective GP algorithm is an approach to multi-objective optimization, as described in [31]. The key extension of this work is incorporating a silicon-calibrated energy model in the algorithm for energy optimization. The energy model is developed based on the feature extraction accelerator from the custom heterogeneous

Algorithm 2 Energy-Aware GP Algorithm

```
Initialize a random population
for t in 1 to T_g do
    Evaluate fitness and complexity of all models
    Update Pareto-frontier archive
    Elitism selection in population for N_e models
    Initialize N ← 0
    while N < N_p − N_e do
        Randomly select a genetic operator
        Tournament selection in population and archive
        Generate new model(s)
        Check constraints of G_max and D_max
        Update N ← N + 1 or N ← N + 2
    Combine all N_p models for a new population
    Determine whether to switch complexity objective
Post-processing on models
```

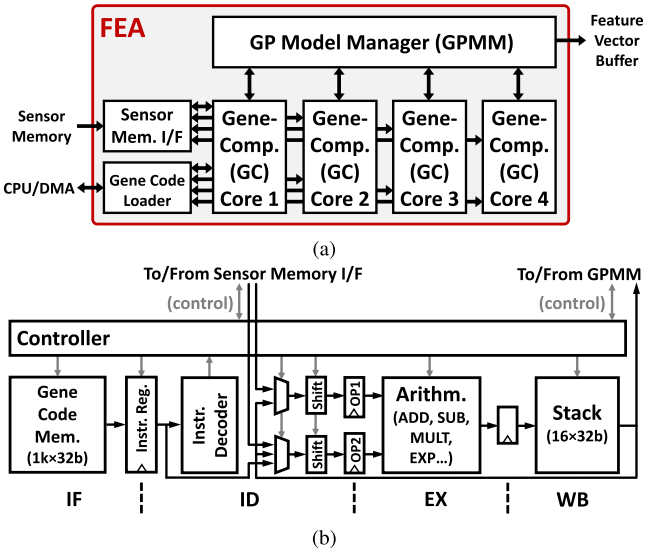


Fig. 3. (a) Details of the FEA in the heterogeneous microprocessor [18]. (b) Microarchitecture of a GC core within the FEA [18].

microprocessor [18]. Further, a Pareto-frontier archive of best-performing models is maintained and updated in each generation for Pareto optimization [19], [30]. This enables tournament selection to include the current population and the archive for breeding models. Similar to the conventional GP algorithm, the energy-aware GP algorithm also provides assurance of convergence to the optimal solution with enough generations, due to the use of elitism selection and tournament selection. Finally, a post-processing step is performed to generate more energy-efficient models. All new characteristics of the proposed GP algorithm will be covered in the following subsections.

B. Energy Model

As previously described, the heterogeneous microprocessor in [18] comprises the FEA with four GC cores, which perform GP-model computations. Each GC core is a stack-based machine, leveraging the data flow and precedence control implicit in reverse polish notation (RPN) [32], to optimize instruction decoding and data movement for gene-tree computations. Fig. 3 shows the details of the FEA and the microarchitecture of a GC core. The GPMM within the FEA performs the linear combination of gene-tree computations. The GC core consists of four stages: instruction fetch (IF), instruction decode and operand fetch (ID), execution (EX), and write-back (WB). Each instruction in the GC core corresponds to a gene-tree node. Energy consumption of a gene-tree node thus consists of three components: (1) instruction fetch; (2) memory accessing for fetching operands from input-data (sensor) memory (constants for function inputs are coded within the instructions); and (3) computation of the node function, including stack push and pop necessary for facilitating RPN data flow. Based on this, we develop the energy model, as shown below:

$$E_{total} = (N_f + M + 1)E_i + N_v E_d + (C_f + 3M + 3)E_c, \quad (3)$$

where E_i denotes the energy consumption of fetching one instruction from gene-code memory within the GC core, E_d denotes the energy consumption of input-data memory per access, E_c denotes the energy consumption of GC core per clock cycle for computing gene trees, N_f denotes the number of functions in the model (gene-tree nodes), N_v denotes the number of input variables in the model, C_f denotes the number of clock cycles required for N_f functions, and M denotes the total number of genes. The microprocessor in [18] supports eight different node functions, as listed in Table I. Linear functions for addition, subtraction, multiplication, and square (ADD/SUB/MULT/SQUARE) require ∼3 clock cycles. Nonlinear functions for exponential, natural logarithm, square root, and reciprocal (EXP/LN/SQRT/1/X) require ∼33 clock cycles (implemented by a CORDIC unit [18]). In addition to N_f functions in the model, there are M multiplications required for applying gene weights and one bias addition (integrated within one gene tree for computation), which in total consume the energy of $[(M + 1)E_i + (3M + 3)E_c]$. Note that the energy of $(M - 1)$ additions for the combination of gene-tree outputs is included in E_c. The high level of accelerator specialization and the structured execution that results enable robust specification of values for the energy-model parameters. Table II summarizes the energy-model parameters (E_i, E_d, E_c) based on the prototype measurement results in [18]. For validation, Fig. 4 shows the energy predicted by the model and

TABLE I
OPERANDS AND CLOCK CYCLES OF NODE FUNCTIONS FOR EXECUTION ON THE FEA

	Operands	FEA Cycles
ADD	2	3
SUB	2	3
MULT	2	3
SQUARE	1	3
EXP	1	33
LN	1	33
SQRT	1	33
1/X	1	33

TABLE II

ENERGY-MODEL PARAMETERS BASED ON THE PROTOTYPE MEASUREMENT RESULTS IN [18]

Technology	GF 130-nm CMOS
Supply Voltage	1.2 V
Instruction Fetch Energy (E_i)	35 pJ
Data-Memory Access Energy (E_d)	42 pJ
Gene-Tree Computation Energy (E_c)	57.6 pJ

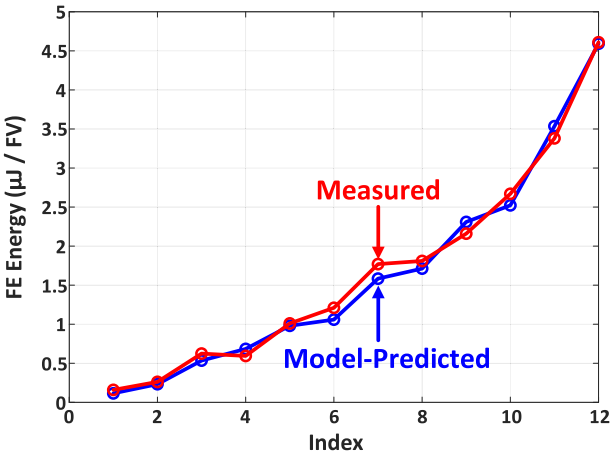


Fig. 4. GP-based feature-extraction energy per feature vector for EEG-based seizure detection, predicted using the energy model and measured from silicon, for different G_{max} and D_{max} parameter settings.

that measured from silicon for feature vectors derived using different G_{max} and D_{max} parameters (indexed from 1 to 12) in the EEG-based seizure-detection application. Good agreement is observed.

C. Pareto Optimization

Fig. 5 shows a scatter plot of an example population of 500 GP models, in terms of model fitness and model energy. Models on the Pareto frontier (marked in green) are the ones that represent optimal points in the tradeoff between two metrics. They thus represent promising models for further investigation. This provides a basis for the GP algorithm to focus on models on or near the Pareto frontier, so that the Pareto frontier can be improved and further pushed to the lower-left corner over the generations. In this way, both objectives of interest are considered in the optimization.

To achieve this, we employ the concept of Pareto optimization [19], [30], based on model archiving and multi-objective evaluation. Specifically, a Pareto-frontier archive (e.g., 5%-10% of the population size), which contains a fixed number of models on the Pareto frontier of two objectives, is maintained during the evolutionary process. In each generation, the archive is updated based on models in the current population and the previous archive. Different from the conventional GP, tournament selection is used among both the population and the archive for breeding new models. Mutation is only performed on models in the archive to enhance best-performing models found in previous generations. Crossover is performed between one model in the archive and one model

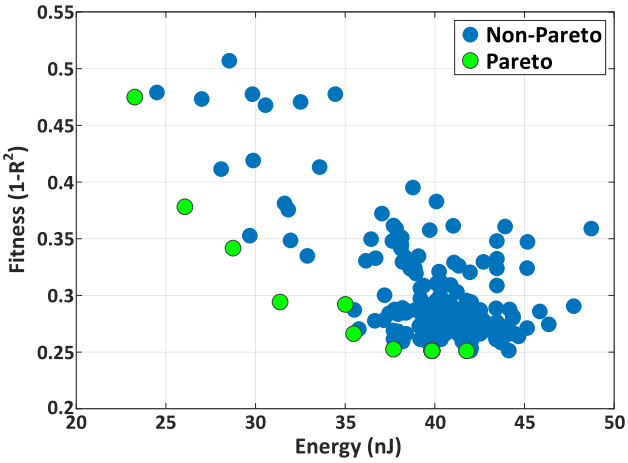


Fig. 5. An example population of 500 GP models, in terms of model fitness and model energy. Models on the Pareto frontier are marked in green.

in the population, to maintain the ability to explore new model structures.

A common problem for updating the Pareto-frontier archive is that the number of models on the Pareto frontier may be smaller than the pre-defined archive size. To overcome this, we use the fast non-dominated sorting algorithm [33], to extract models in the order of non-dominance. Specifically, the models on the Pareto frontier are first extracted, assigned a rank of 1, and removed from the model candidates. Then, the models on the Pareto frontier for the remaining candidates are extracted, assigned a rank of 2, and removed. This procedure is repeated until the desired archive size is reached. If the total number of extracted models exceeds the archive size, models with the lowest rank are randomly chosen for removal.

The main advantage of Pareto optimization is that it considers two competing objectives (e.g., model fitness and model complexity) to be equally important, which avoids the need to specify a parsimony pressure *a priori*. In contrast, the conventional GP algorithm only considers model fitness, and spends all computational efforts in improving the performance of models with high fitness as well as high complexity, even though the fitness improvement is often marginal compared to other, much-less complex, models in the population.

D. Alternating Model Complexity Objective

The energy model we develop can be viewed as one way to characterize the model complexity. However, the models generated by Pareto optimization on model fitness and model energy may consist of many simple and nested functions (e.g., addition, multiplication). The unbalanced tree structures often have the risk of overfitting on the data used for evolving models, resulting in decreased generalization ability. To mitigate this, we employ another complexity metric, named expressional complexity [19], to favor flatter and more balanced tree structures. The expressional complexity of a tree is computed by summing together the node count of the tree and all its possible full subtrees. For the MGGP variant we employ in

Algorithm 3 Alternating Model Complexity Objective

```
Randomly initialize a complexity objective
Initialize T_c ← 0
for t in 1 to T_g do
    Select N_b models with highest fitness in population
    Calculate average complexity C of N_b models
    if T_c > 0 then
        if |C − C_p| <= C_Δ then
            Update T_c ← T_c + 1
            if T_c >= T_cmax then
                Switch complexity objective
                Reset T_c ← 0
        else
            Reset T_c ← 1
    else
        Set T_c ← 1
    Update C_p ← C
```

this paper, the expressional complexity of a GP model is the sum of the expressional complexity of all its genes.

The problem with adding more objectives (e.g., model fitness, model energy, model expressional complexity) for optimization is that the model discovery efficiency will be degraded, due to the difficulty in covering the Pareto-frontier hypersurface for the GP algorithm. Multi-objective approaches often suffer from such "curse of dimensionality" [34], and scale badly with an increased number of objectives.

To overcome the scaling problem, we develop a heuristic approach for two-dimensional optimization of model fitness and model complexity, where the complexity objective is alternated between model energy and model expressional complexity during the evolutionary process. This approach is inspired by the work in [31], which aims to generate both accurate and generalized models. The main difference is that [31] focuses on the order of nonlinearity for a smoother response surface, but we instead focus on model energy for a better connection with hardware mapping and execution. Additionally, rather than simply alternating the objectives in each generation (as in [31]), we alternate the objectives dynamically based on the evolutionary history, as summarized in Algorithm 3. T_g denotes the total number of generations to evolve models. T_c denotes the number of successive generations in which the change of complexity is under a threshold C_Δ. Note that when $T_c = 0$, the current generation is the first generation after complexity objective initialization or switching. T_{cmax} is the maximum value we can tolerate on T_c to switch complexity objective. The complexity comparison is based on the average complexity of several models with the highest fitness values in the population (C_p denotes the average complexity of selected models in the previous generation). The complexity threshold C_Δ is set as below:

$$C_\Delta = sN_{max} = s\left(2^{D_{max}} - 1\right) G_{max}, \tag{4}$$

where s denotes a scaling factor (for either model energy or model expressional complexity), and N_{max} denotes the maximum number of gene-tree nodes allowed for each model

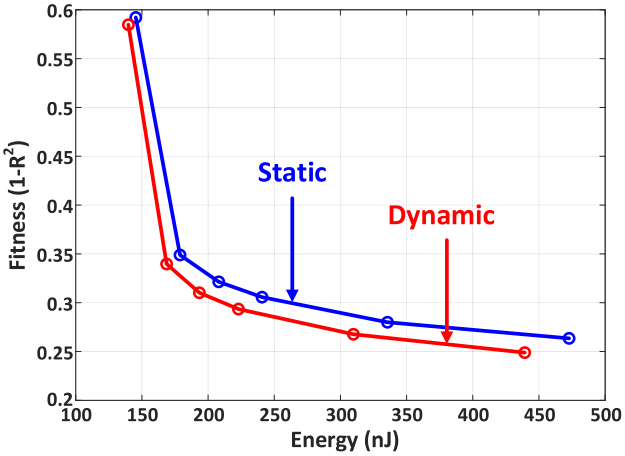


Fig. 6. Performance evaluation of the dynamic approach achieved by GP models, in terms of model fitness and model energy, averaged from five independent GP runs. The GP models are generated for EEG-based seizure-detection application, with G_{max} set to 10 and D_{max} ranging from 2 to 7. The model fitness is the average fitness among all eight GP models, and the model energy is the total energy of all eight GP models.

under the constraints of G_{max} and D_{max}. The scaling factor s needs to be set properly for a given application. If s is too small, the complexity objective will never be switched. If s is too large, the complexity objective will be alternated every T_{cmax} generations. This dynamic approach helps alleviate the problem of getting stuck at local optima and the possible loss of model diversity. Our experiments show better performance of the generated models than that observed for models generated by the static approach [31]. Fig. 6 shows the model fitness and model energy of the two approaches for the same parameter settings.

E. Post-Processing on GP Models

As mentioned in Section II, the generated GP models may contain some overly-complex genes, which offer little fitness improvement. One simple way to identify these genes is by checking the correlation of different gene pairs in the model. If two genes are highly correlated with each other, the more complex (energy-intensive) gene can be removed, at little fitness cost. After removing all overly-complex genes in the model, the gene weights and the bias term need to be recomputed based on the remaining genes. In this paper, we use the common Pearson correlation coefficient for correlation evaluation. Two genes are claimed to be highly-correlated if the absolute value of the correlation coefficient of their outputs (evaluated on the given inputs) is larger than a pre-defined threshold. Note that the removal of highly-correlated genes is only performed on generated GP models in the last generation instead of every generation. This is done to maintain model diversity based on GP-model structures over the optimization generations, rather than otherwise suppressing such diversity based on the data used for fitting.

IV. Application Demonstration

A. Algorithm Parameter Settings

To compare the performance with the conventional GP algorithm, we implement the energy-aware GP algorithm

TABLE III

COMMON PARAMETER SETTINGS FOR THE CONVENTIONAL GP ALGORITHM AND THE ENERGY-AWARE GP ALGORITHM

	Parameter	Setting
Model Generation	Number of Generations (T_g)	1000
	Population Size (N_P)	500
	Gene-Number Constraint (G_{max})	{1, 5, 10, 15}
	Gene-Depth Constraint (D_{max})	{2, 3, ⋯, 7}
	Mutation Rate	0.10
	Crossover Rate	0.85
	Reproduction Rate	0.05
Model Selection	Elitism Size (N_e)	25
	Population Tournament Size	20

TABLE IV

PARAMETER SETTINGS FOR THE ENERGY-AWARE GP ALGORITHM

	Parameter	Setting
Pareto Optimization	Archive Size	50
	Archive Tournament Size	5
Alternating Model Complexity Objective	Number of Selected Models (N_b)	15
	Tolerant Generations (T_{cmax})	3
	Threshold Scaling Factors (EEG)	0.10, 20 pJ
	Threshold Scaling Factors (ECG)	0.02, 2 pJ
Post-Processing	Correlation Threshold	0.95

employing the GPTIPS toolbox [35]. Two medical-sensor applications, namely EEG-based seizure detection and ECG-based arrhythmia detection, are developed for demonstration. These two applications have served as common benchmarks in the area of medical sensors, both because of their relevance to low-power embedded systems and because of the public availability of anonymized datasets. To account for the statistical nature of GP, we perform five independent runs for each experiment and present the average performance results. Some common parameter settings for both two algorithms are summarized in Table III. As seen, various combinations of the gene-number constraint (G_{max}) and the gene-depth constraint (D_{max}) are used, to evaluate performance with different maximum complexity control on models. 1000 generations and 500 population size are found to be adequate in our experiments. Table IV lists some parameter settings specific to the energy-aware GP algorithm. For alternating the complexity objective in the two applications, we use different scaling factors for setting complexity thresholds. The parameters listed in Table III and Table IV are common in GP-related algorithms [20], [26], [31]. In practice, if the number of generations and population size are adequate, the results exhibit little sensitivity to other parameters (thus parameter settings can be transferred across applications).

B. EEG-Based Seizure Detection

For the EEG-based seizure-detection application, we employ the CHB-MIT database [36], which contains EEG recordings with seizure and non-seizure segments. The recordings are sampled at 256 Hz and measured using multiple channels. Fig. 7 shows the detection system based on the algorithm in [37]. The baseline feature extraction consists of three steps: (1) decimation, to reduce the sampling rate of EEG recordings to 64 Hz; (2) linear filtering, to process the down-sampled EEG recordings through eight band-pass

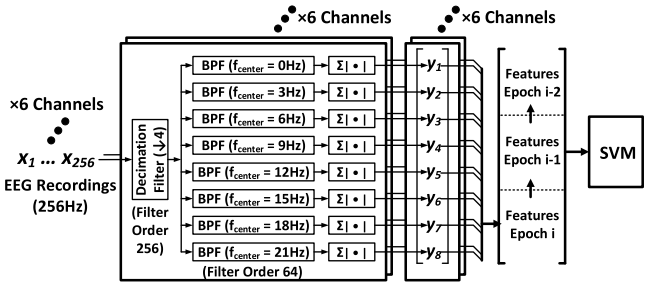


Fig. 7. Baseline EEG-based seizure-detection system.

filters (BPFs), each with different center frequencies; and (3) energy accumulation, to extract the baseline features corresponding to the spectral energy distribution of each channel. This results in eight features per epoch for each channel. Then, the features from three successive epochs are concatenated, giving a final feature vector with the dimensionality of 144 ($8 \times 3 \times 6$, for in total six channels used) for SVM classification based on RBF kernel.

All three steps of the baseline feature extraction are modeled using GP. Therefore, we generate eight GP models for the eight features (extracted per epoch per channel), using a function set consisting of subtraction and square root (suggested by [26]). Three metrics defined in [37] are employed to evaluate the inference performance: (1) sensitivity, to measure the fraction of correctly-detected seizures over all seizures in the database; (2) latency, to measure the delay in seizure detection compared to the ground-truth seizure onset; and (3) false alarms per hour, to measure the average number of falsely-detected seizures per hour. The baseline inference performance (on the subset of the 8th case) is 100% sensitivity, 3.81 seconds latency, and 0.05 false alarms per hour. For both the conventional and proposed GP algorithms, the SVM classifiers are retrained based on the corresponding GP-approximated features.

Table V presents the performance results of EEG-based seizure detection for the conventional GP algorithm and the energy-aware GP algorithm. The feature-extraction energy reported (in the first column of both two algorithms) is for each feature vector, and the fitness reported (in the second column of both two algorithms) is the average R^2 among all eight GP models. As seen, a wide range in performance is achieved with different settings of G_{max} and D_{max}. The model fitness increases when G_{max} or D_{max} increases, but at the cost of increased model energy. The fitness-energy Pareto frontier is improved for the energy-aware GP algorithm, achieving a better tradeoff between model fitness and model energy. Fig. 8 illustrates this using an example of generated populations corresponding to one of the eight GP models for the two algorithms. Note that for the conventional GP, the Pareto frontier is only identified for post-run analysis, and it does not affect the evolutionary process for optimizing model fitness. In contrast, the energy-aware GP actively updates and optimizes the Pareto frontier during the GP run.

As the energy-aware GP optimizes both model fitness and model energy, the fitness of GP models is often slightly lower than that from the conventional GP in the same setting of G_{max} and D_{max}. However, despite the fitness reduction,

TABLE V

PERFORMANCE RESULTS OF EEG-BASED SEIZURE DETECTION FOR THE CONVENTIONAL GP ALGORITHM AND THE ENERGY-AWARE GP ALGORITHM

		Conventional GP Algorithm						Energy-Aware GP Algorithm					
G_{max}	D_{max}	FE Energy (μJ/FV)	Fitness R^2 (%)	Sensitivity (%)	Latency (second)	False Alarms Per Hour	Class. Energy (μJ/FV)	FE Energy (μJ/FV)	Fitness R^2 (%)	Sensitivity (%)	Latency (second)	False Alarms Per Hour	Class. Energy (μJ/FV)
1	2	0.11	16.58	100.00	13.01	0.35	17.33	0.11	16.44	100.00	13.01	0.34	17.24
	3	0.13	30.63	100.00	7.05	0.65	17.85	0.13	30.18	100.00	8.13	0.74	18.15
	4	0.19	36.08	100.00	7.73	0.39	19.73	0.20	37.41	100.00	7.61	0.30	20.12
	5	0.30	44.36	100.00	6.61	0.43	19.26	0.32	46.06	100.00	6.85	0.31	19.95
	6	0.55	55.66	100.00	6.01	0.28	18.39	0.51	54.78	100.00	5.93	0.22	18.31
	7	0.82	57.61	100.00	5.85	0.20	18.59	0.74	58.88	100.00	5.77	0.28	18.36
5	2	0.53	37.13	100.00	11.97	0.67	19.62	0.50	36.04	100.00	12.05	0.72	19.67
	3	0.59	59.62	100.00	5.29	0.38	16.32	0.58	58.16	100.00	5.61	0.43	16.80
	4	0.80	62.55	100.00	5.09	0.27	16.83	0.66	61.40	100.00	5.09	0.31	16.74
	5	1.36	66.92	100.00	5.25	0.16	17.50	1.04	66.05	100.00	5.21	0.23	17.39
	6	2.13	70.17	100.00	4.45	0.13	16.48	1.46	70.00	100.00	5.01	0.10	16.57
	7	3.28	74.12	100.00	4.41	0.16	16.42	1.84	72.36	100.00	4.37	0.16	16.22
10	2	1.06	43.31	100.00	12.09	0.44	17.02	0.84	41.52	100.00	11.85	0.48	17.74
	3	1.18	67.79	100.00	4.57	0.31	15.51	1.01	66.03	100.00	4.77	0.27	15.58
	4	1.69	70.74	100.00	4.37	0.20	16.07	1.16	68.98	100.00	4.37	0.22	16.11
	5	2.30	72.95	100.00	4.61	0.06	16.32	1.34	70.65	100.00	4.65	0.08	16.76
	6	3.35	75.31	100.00	4.69	0.09	16.24	1.86	73.23	100.00	4.49	0.08	16.45
	7	4.92	77.26	100.00	4.29	0.07	15.41	2.64	75.11	100.00	4.53	0.03	15.72
15	2	1.58	45.55	100.00	11.65	0.38	16.25	1.13	43.72	100.00	11.85	0.39	16.95
	3	1.77	71.26	100.00	4.41	0.21	14.63	1.43	69.59	100.00	4.57	0.16	15.16
	4	2.67	74.36	100.00	4.09	0.17	14.87	1.78	72.84	100.00	4.37	0.12	15.50
	5	3.35	76.53	100.00	4.41	0.06	15.50	**2.03**	**74.87**	**100.00**	**4.13**	**0.06**	**16.03**
	6	**4.76**	**77.91**	**100.00**	**4.09**	**0.09**	**15.15**	2.62	75.96	100.00	4.21	0.11	15.66
	7	6.34	79.40	100.00	4.13	0.09	15.36	3.44	77.11	100.00	4.17	0.11	15.60

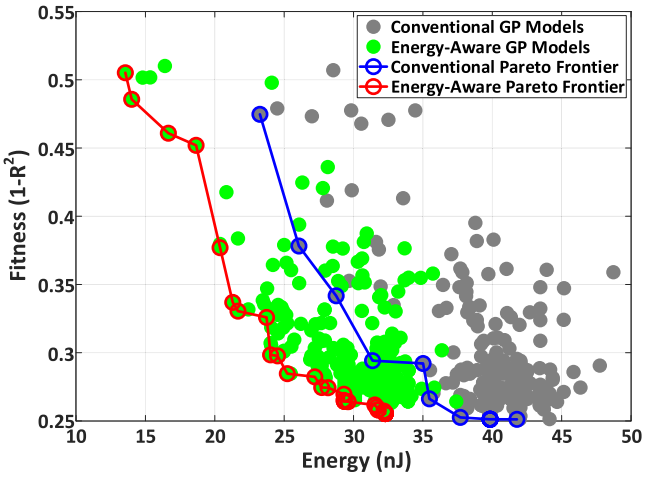


Fig. 8. Populations of GP models from one GP run, in terms of model fitness and model energy. All models correspond to one of the eight GP models for EEG-based seizure detection, with G_{max} set to 10 and D_{max} set to 5. The models generated by the conventional GP algorithm are plotted as gray circles, with the corresponding Pareto-frontier models highlighted in blue. The models generated by the energy-aware GP algorithm are plotted as green circles, with the corresponding Pareto-frontier models highlighted in red.

the final inference performance is restored through explicit classifier retraining to account for approximation statistics, as shown in Table V, yet with lower energy consumption of GP models. When G_{max} or D_{max} increases, the energy savings for models also increase, due to the larger optimization space for GP. As denoted in bold in Table V, at the same level of baseline inference performance, the feature-extraction energy per feature vector is 2.03 μJ for the energy-aware GP, 57.4% lower compared to 4.76 μJ for the conventional GP.

The GP models are generated for feature extraction. However, for the total energy consumption of the detection system, we also need to consider the classification energy. As the RBF kernel is employed for SVM classification in EEG-based seizure detection, the classification energy is dependent on the number of support vectors used. As shown in Table V (in the last column of both two algorithms), the classification energy dominates the system energy when G_{max} and D_{max} are small. However, at the same level of baseline inference performance, the classification energy and feature-extraction energy are comparable. Indeed, with the increase of G_{max} and D_{max}, model fitness also increases, resulting in improved feature quality. This could ease the classifier, for instance requiring fewer support vectors in the SVM used, thus reducing the classification energy. The classification energy is roughly the same for the conventional and energy-aware GP algorithms in different settings of G_{max} and D_{max}.

C. ECG-Based Arrhythmia Detection

For the ECG-based arrhythmia-detection application, we employ the MIT-BIH database [36], [38], which contains ECG recordings with segments of arrhythmia and normal heartbeat. Fig. 9 shows the detection system based on the algorithm in [39]. The baseline feature extraction is performed using a four-stage wavelet filter bank (cascaded high-pass filters (HPFs) and low-pass filters (LPFs)), followed by principal component analysis (PCA) to reduce the feature dimensionality. This gives a final feature vector with the dimensionality of 20 for SVM classification based on the polynomial kernel.

TABLE VI

PERFORMANCE RESULTS OF ECG-BASED ARRHYTHMIA DETECTION FOR THE CONVENTIONAL GP ALGORITHM AND THE ENERGY-AWARE GP ALGORITHM

		Conventional GP Algorithm					Energy-Aware GP Algorithm				
G_{max}	D_{max}	FE Energy (nJ/FV)	Fitness R^2 (%)	Sensitivity (%)	Specificity (%)	Accuracy (%)	FE Energy (nJ/FV)	Fitness R^2 (%)	Sensitivity (%)	Specificity (%)	Accuracy (%)
1	2	11.40	31.70	69.34	85.22	84.55	11.52	31.66	70.87	86.13	85.35
	3	11.84	31.72	69.32	85.35	84.67	10.18	31.73	69.28	85.20	84.51
	4	12.56	31.77	69.33	85.24	84.55	10.03	31.80	69.21	85.20	84.49
	5	25.23	32.69	70.05	85.31	84.61	23.96	32.86	70.45	85.09	84.44
	6	34.40	33.48	70.28	84.86	84.22	32.29	33.31	70.43	84.09	84.04
	7	52.63	33.75	70.67	85.09	84.42	48.82	33.58	70.36	85.17	84.50
5	2	30.10	75.49	78.49	84.28	84.24	28.80	75.16	78.27	84.06	83.92
	3	29.82	75.45	78.73	82.88	82.82	28.49	75.11	77.70	83.06	82.97
	4	29.56	75.20	78.18	83.18	83.23	28.60	74.96	78.35	83.03	82.90
	5	29.79	74.85	78.45	83.66	83.64	28.72	75.09	77.70	84.29	84.11
	6	42.33	75.25	78.22	84.07	83.73	34.53	75.18	77.97	83.43	83.43
	7	40.76	74.98	77.95	83.04	82.83	35.53	75.04	77.35	83.87	83.64
10	2	55.01	90.66	79.39	87.46	87.10	47.56	89.97	79.68	86.31	86.29
	3	54.43	90.56	79.57	86.25	86.03	48.00	89.98	79.63	86.17	86.03
	4	60.17	90.68	79.97	86.95	86.71	46.73	89.75	79.95	86.28	86.28
	5	66.93	90.72	79.23	87.67	87.30	46.95	89.84	79.75	86.77	86.41
	6	65.81	90.23	79.49	87.09	86.74	47.30	89.65	79.82	86.73	86.67
	7	59.87	90.50	79.90	86.52	86.43	46.80	89.52	79.42	86.30	86.15
15	2	82.01	97.46	81.32	87.71	87.41	63.36	96.41	80.84	87.43	87.21
	3	**79.97**	**97.34**	**81.25**	**87.48**	**87.18**	62.93	96.29	80.70	87.06	86.72
	4	82.96	97.35	80.86	87.49	87.24	**62.52**	**95.91**	**81.02**	**87.59**	**87.27**
	5	90.46	97.38	81.33	87.35	87.08	62.84	96.11	80.91	87.53	87.44
	6	79.98	97.34	81.43	87.21	86.93	61.71	95.82	80.48	87.08	86.82
	7	104.97	97.23	80.78	87.36	87.04	61.57	96.00	80.52	86.84	86.58

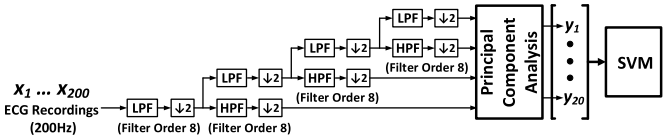


Fig. 9. Baseline ECG-based arrhythmia-detection system.

Both two steps of wavelet-based filtering and PCA for the baseline feature extraction are modeled using GP. Therefore, we generate 20 GP models for the 20 features, using a function set consisting of multiplication and exponential. Three metrics are employed to evaluate the inference performance: (1) sensitivity, to measure the fraction of correctly-detected arrhythmias over all arrhythmias in the database; (2) specificity, to measure the fraction of correctly-detected normal heartbeats over all normal heartbeats in the database; and (3) accuracy, to measure the fraction of correctly-identified data segments. The baseline inference performance is 82.05% sensitivity, 88.12% specificity, and 87.92% accuracy. For both the conventional and proposed GP algorithms, the SVM classifiers are retrained based on the corresponding GP-approximated features.

Table VI presents the performance results of ECG-based arrhythmia detection for the conventional GP algorithm and the energy-aware GP algorithm. As seen, the model fitness and the corresponding inference performance increase when G_{max} increases. However, different from the case of EEG-based seizure detection, increasing D_{max} in the same setting of G_{max} does not enhance the performance too much. This indicates that the generated GP models prefer shallow gene trees. Deep gene trees may even have the risk of overfitting on the data used for evolving models.

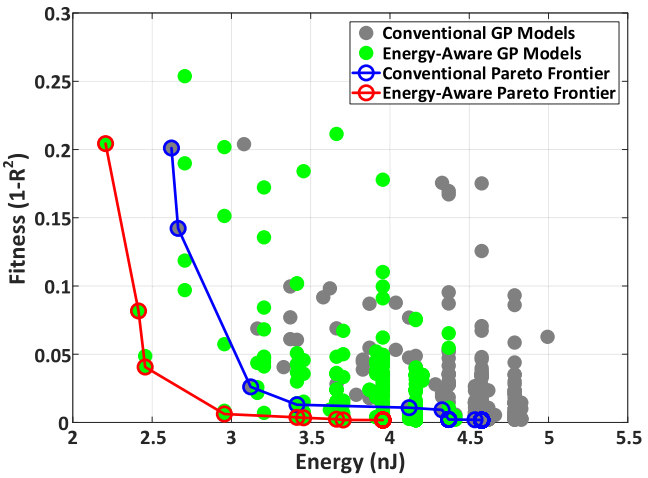


Fig. 10. Populations of GP models from one GP run, in terms of model fitness and model energy. All models correspond to one of the 20 GP models for ECG-based arrhythmia detection, with G_{max} set to 15 and D_{max} set to 2. The models generated by the conventional GP algorithm are plotted as gray circles, with the corresponding Pareto-frontier models highlighted in blue. The models generated by the energy-aware GP algorithm are plotted as green circles, with the corresponding Pareto-frontier models highlighted in red.

Additionally, the fitness-energy Pareto frontier is improved for the energy-aware GP algorithm, as illustrated in Fig. 10.

Similar to the case of EEG-based seizure detection, despite the slight fitness reduction for the energy-aware GP algorithm, the final inference performance is restored thanks to explicit classifier retraining, yet with lower energy consumption of GP models. As denoted in bold in Table VI, at the same level of baseline inference performance, the feature-extraction energy per feature vector is 62.52 nJ for the energy-aware GP, 21.8% lower compared to 79.97 nJ for the conventional GP.

TABLE VII
COMPARISON TABLE WITH STATE-OF-THE-ART MEDICAL-SENSOR SYSTEMS

	JSSC'13 [22]	JSSC'13 [40]	JSSC'14 [41]	TBioCAS'16 [42]	JSSC'17 [43]	JSSC'18 [44]	This Work
Technology	130 nm	180 nm	180 nm	180 nm	180 nm	130 nm	130 nm
Application	General	Seizure	Seizure	Seizure	Sleep	General	General
Classification Accelerator	SVM	Linear SVM	LLS	Non-Linear SVM	Decision Trees	EDM-SVM	SVM
Feature-Extraction Accelerator	No	BPF	FFT, ApEn	BPF	FFT, SEF	SE, PLV, CFC	GP Models
Energy Efficiency	273 μJ/class. (Seizure)	2.03 μJ/class. (Eye Blink)	77.91 μJ/class. (Seizure)	1.83 μJ/class. (Eye Blink)	0.7 μJ/class. (Sleep)	168.6 μJ/class. (Seizure)	18.06 μJ/class. (Seizure)
Feature-Extraction Programmability	Yes	No	No	No	No	Yes	Yes

For the SVM classification based on the polynomial kernel in ECG-based arrhythmia detection, we take advantage of a technique to transform the support vectors into a matrix form [22]. This allows the classification energy to be independent of the number of support vectors used. According to the prototype measurement results in [18], the classification energy per feature vector is 180 nJ, in all settings of G_{max} and D_{max} for the conventional and energy-aware GP algorithms. Similar to the case of EEG-based seizure detection, the classification energy dominates the system energy when G_{max} and D_{max} are small, yet comparable to the feature-extraction energy at the same level of baseline inference performance.

D. Comparison With State of the Art

Table VII lists the comparison with some state-of-the-art medical-sensor systems. We note that due to the different applications and thus different feature-extraction and classification computations involved, a direct comparison of energy efficiency is difficult. However, we do see that for the design goal of programmable acceleration, the proposed approach leads to high energy efficiency, by exploiting structured GP computations and approximate computing.

V. CONCLUSION

The primary challenge with hardware acceleration in embedded sensing systems, especially in the domain of approximate computing, is the limitations imposed on programmability. This has motivated co-design of accelerators with program-synthesis frameworks, where the structured computations resulting from synthesis are exploited towards a high level of accelerator specialization. This paper proposes energy-efficient code synthesis targeting heterogeneous architectures for approximate computing, leveraging the accelerator specialization and structured mapping of computations, in order to develop robust energy models for execution on a programmable feature extraction accelerator. The energy models are then employed in an energy-aware GP algorithm to enhance the energy-approximation Pareto frontier. Compared to the conventional GP algorithm, the energy consumption of GP models executed on the accelerator is 57.4% and 21.8% lower, respectively, for equal accuracy in an EEG-based seizure-detection application and an ECG-based arrhythmia-detection application.

REFERENCES

[1] M. El Ayadi, M. S. Kamel, and F. Karray, "Survey on speech emotion recognition: Features, classification schemes, and databases," *Pattern Recognit.*, vol. 44, no. 3, pp. 572–587, 2011.

[2] I. Yoo *et al.*, "Data mining in healthcare and biomedicine: A survey of the literature," *J. Med. Syst.*, vol. 36, no. 4, pp. 2431–2448, Aug. 2012.

[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[4] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: Review, opportunities and challenges," *Briefings Bioinf.*, vol. 19, no. 6, pp. 1236–1246, 2017.

[5] E. B. Mazomenos *et al.*, "A low-complexity ECG feature extraction algorithm for mobile healthcare applications," *IEEE J. Biomed. Health Inform.*, vol. 17, no. 2, pp. 459–469, Mar. 2013.

[6] M. Price, J. Glass, and A. P. Chandrakasan, "A 6 mW, 5,000-word real-time speech recognizer using WFST models," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 102–112, Jan. 2015.

[7] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62-1–62-33, Mar. 2016.

[8] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *Proc. Asia South Pacific Des. Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 201–206.

[9] Z. Wang, K. H. Lee, and N. Verma, "Overcoming computational errors in sensing platforms through embedded machine-learning kernels," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 8, pp. 1459–1470, Aug. 2015.

[10] Z. Wang, R. E. Schapire, and N. Verma, "Error adaptive classifier boosting (EACB): Leveraging data-driven training towards hardware resilience for signal inference," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 4, pp. 1136–1145, Apr. 2015.

[11] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, Feb. 2017.

[12] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 519–530, Jun. 2013.

[13] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, Nov. 2016, pp. 265–283.

[14] A. Solar-Lezama, "Program sketching," *Int. J. Softw. Tools Technol. Trans.*, vol. 15, no. 5, pp. 475–495, Oct. 2013.

[15] S. Gulwani, O. Polozov, and R. Singh, "Program synthesis," *Found. Trends Program. Lang.*, vol. 4, nos. 1–2, pp. 1–119, Jul. 2017.

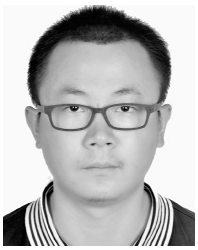
[16] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[17] W. B. Langdon and R. Poli, *Foundations of Genetic Programming*. Berlin, Germany: Springer, 2013.

[18] H. Jia and N. Verma, "Exploiting approximate feature extraction via genetic programming for hardware acceleration in a heterogeneous microprocessor," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 1016–1027, Apr. 2018.

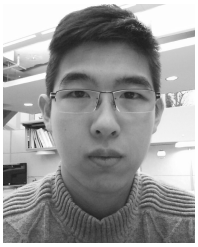
[19] G. F. Smits and M. Kotanchek, "Pareto-front exploitation in symbolic regression," in *Genetic Programming Theory and Practice II*. Boston, MA, USA: Springer, 2005, pp. 283–299.

[20] A. H. Gandomi and A. H. Alavi, "A new multi-gene genetic programming approach to nonlinear system modeling. Part I: Materials and structural engineering problems," *Neural Comput. Appl.*, vol. 21, no. 1, pp. 171–187, Feb. 2012.
[21] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
[22] K. H. Lee and N. Verma, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," *IEEE J. Solid-State Circuits*, vol. 48, no. 7, pp. 1625–1637, Jul. 2013.
[23] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," *IEEE Trans. Evol. Comput.*, vol. 17, no. 3, pp. 368–386, Jun. 2013.
[24] D. Y. Harvey and M. D. Todd, "Automated feature design for numeric sequence classification by genetic programming," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 474–489, Aug. 2015.
[25] K. Nag and N. R. Pal, "A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification," *IEEE Trans. Cybern.*, vol. 46, no. 2, pp. 499–510, Feb. 2016.
[26] J. Lu, H. Jia, N. Verma, and N. K. Jha, "Genetic programming for energy-efficient and energy-scalable approximate feature computation in embedded inference systems," *IEEE Trans. Comput.*, vol. 67, no. 2, pp. 222–236, Feb. 2018.
[27] M. J. Streeter, "The root causes of code growth in genetic programming," in *Genetic Programming*. Berlin, Germany: Springer, 2003, pp. 443–454.
[28] S. Luke and L. Panait, "A comparison of bloat control methods for genetic programming," *Evol. Comput.*, vol. 14, no. 3, pp. 309–344, Sep. 2006.
[29] P. A. Whigham and G. Dick, "Implicitly controlling bloat in genetic programming," *IEEE Trans. Evol. Comput.*, vol. 14, no. 2, pp. 173–190, Apr. 2010.
[30] M. Kotanchek, G. Smits, and E. Vladislavleva, "Pursuing the Pareto paradigm: Tournaments, algorithm variations and ordinal optimization," in *Genetic Programming Theory and Practice IV*. Boston, MA, USA: Springer, 2007, pp. 167–185.
[31] E. J. Vladislavleva, G. F. Smits, and D. D. Hertog, "Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 333–349, Apr. 2009.
[32] W. B. Langdon and W. Banzhaf, "A SIMD interpreter for genetic programming on GPU graphics cards," in *Genetic Programming*. Berlin, Germany: Springer, 2008, pp. 73–85.
[33] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[34] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *Computational Intelligence and Bioinspired Systems*. Berlin, Germany: Springer, 2005, pp. 758–770.
[35] D. P. Searson, "GPTIPS 2: An open-source software platform for symbolic data mining," in *Handbook of Genetic Programming Applications*. Cham, Switzerland: Springer, 2015, pp. 551–573.
[36] A. L. Goldberger *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, Jun. 2000.
[37] A. Shoeb and J. Guttag, "Application of machine learning to epileptic seizure detection," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jun. 2010, pp. 975–982.
[38] G. B. Moody and R. G. Mark, "The impact of the MIT-BIH arrhythmia database," *IEEE Eng. Med. Biol. Mag.*, vol. 20, no. 3, pp. 45–50, May 2001.
[39] E. D. Übeyli, "ECG beats classification using multiclass support vector machines with error correcting output codes," *Digit. Signal Process.*, vol. 17, no. 3, pp. 675–684, May 2007.
[40] J. Yoo, L. Yan, D. El-Damak, M. A. B. Altaf, A. H. Shoeb, and A. P. Chandrakasan, "An 8-channel scalable EEG acquisition SoC with patient-specific seizure classification and recording processor," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 214–228, Jan. 2013.
[41] W.-M. Chen *et al.*, "A fully integrated 8-channel closed-loop neural-prosthetic CMOS SoC for real-time epileptic seizure control," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 232–247, Jan. 2014.
[42] M. A. B. Altaf and J. Yoo, "A 1.83 μJ/classification, 8-channel, patient-specific epileptic seizure classification SoC using a non-linear support vector machine," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 1, pp. 49–60, Feb. 2016.
[43] S. A. Imtiaz, Z. Jiang, and E. Rodriguez-Villegas, "An ultralow power system on chip for automatic sleep staging," *IEEE J. Solid-State Circuits*, vol. 52, no. 3, pp. 822–833, Mar. 2017.
[44] G. O'Leary, D. M. Groppe, T. A. Valiante, N. Verma, and R. Genov, "NURIP: Neural interface processor for brain-state classification and programmable-waveform neurostimulation," *IEEE J. Solid-State Circuits*, vol. 53, no. 11, pp. 3150–3162, Nov. 2018.



Yinqi Tang (Student Member, IEEE) received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2014, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2016, where he is currently pursuing the Ph.D. degree.

His current research interests include energy-efficient hardware systems for machine learning and deep learning applications, in both algorithm and hardware design aspects.



Hongyang Jia (Student Member, IEEE) received the B.Eng. degree in microelectronics from Tsinghua University, Beijing, China, in 2014, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2016, where he is currently pursuing the Ph.D. degree.

His research focuses on ultra-low energy system design for inference applications. His primary research interests include programmable in-memory computing platforms, CMOS IC design leveraging approximate computing technique for model complexity reduction, automatic program synthesis, and mapping for feature extraction on various sensing platforms.

Mr. Jia received Analog Devices Outstanding Student Designer Award in 2017.

Naveen Verma (Member, IEEE) received the B.A.Sc. degree in electrical and computer engineering from The University of British Columbia, Vancouver, BC, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2005 and 2009, respectively.

Since 2009, he has been with the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, where he is currently a Professor. His research focuses on advanced sensing systems, including low-voltage digital logic and SRAMs, low-noise analog instrumentation and data-conversion, large-area sensing systems based on flexible electronics, and low-energy algorithms for embedded inference, especially for medical applications.

Dr. Verma was a recipient or co-recipient of the 2006 DAC/ISSCC Student Design Contest Award, the 2008 ISSCC Jack Kilby Paper Award, the 2012 Alfred Rheinstein Junior Faculty Award, the 2013 NSF CAREER Award, the 2013 Intel Early Career Award, the 2013 Walter C. Johnson Prize for Teaching Excellence, the 2013 VLSI Symposium Best Student Paper Award, the 2014 AFOSR Young Investigator Award, the 2015 Princeton Engineering Council Excellence in Teaching Award, and the 2015 IEEE TRANSACTIONS ON CPMT Best Paper Award. He is a Distinguished Lecturer of the IEEE Solid-State Circuits Society, and serves on the technical program committees for the International Solid-State Circuits Conference (ISSCC), the VLSI Symposium, DATE, and the IEEE Signal-Processing Society (DISPS).