EvoGrad: Accelerated Metaheuristics in a Differentiable Wonderland

Beatrice F.R. Citterio
Department of Computing Sciences
Bocconi University, Milan, Italy
beatrice.citterio2@studbocconi.it

Daniele M. Papetti
Department of Informatics, Systems and Communication
University of Milano-Bicocca, Milan, Italy
daniele.papetti@unimib.it

Giovanna Maria Dimitri
Department of Social and Political Sciences
University of Milan, Milan, Italy
giovanna.dimitri@unimi.it

Andrea Tangherloni
Department of Computing Sciences
Bocconi Institute for Data Science and Analytics
Bocconi University, Milan, Italy
andrea.tangherloni@unibocconi.it

***Abstract*—Differentiable programming has revolutionised real-valued optimisation by enabling efficient optimisation of highly complex, differentiable, parameterised functions and of complex models such as Deep Neural Networks. However, traditional Evolutionary Computation (EC) and Swarm Intelligence (SI) algorithms, widely successful in discrete search spaces or noisy and multimodal functions, typically do not leverage gradient information, limiting their efficiency and adaptability during the optimisation. To bridge these approaches, we introduce EvoGrad, a unified differentiable framework that integrates EC and SI with gradient-based optimisation. EvoGrad redefines all method components, ranging from candidate solution selection to evolutionary and swarm operators, in a differentiable manner, thereby facilitating their tuning and enabling small local searches. Extensive experiments on benchmark optimisation functions and on a real-world Parameter Estimation problem of a biochemical system reveal that our differentiable versions of EC and SI metaheuristics consistently outperform and compete with traditional, gradient-agnostic methods, setting a new standard for hybrid optimisation frameworks.**

***Index Terms*—Differentiable programming, gradient descent, metaheuristics, single-obj. optimisation, parameter estimation**

I. Introduction

Differentiable programming (DP) has emerged as a transformative computational paradigm [1], enabling end-to-end optimisation of computational models by leveraging Automatic Differentiation [2]. Neural networks (NNs) represent a specific yet highly successful instance of DP [3]; indeed, NNs are a composition of parameterised differentiable primitives that operate on D-dimensional tensors of, typically, real values. By combining such primitives in a modular fashion, NNs induce a differentiable computational graph (i.e., a graph that tracks the order of all executed operations) that can be used by DP to find the optimal parameter values. To do so, DP leverages backpropagation and gradient-based methods to formulate and solve an optimisation problem in the real-valued hyperspace induced by the computation graph, aiming to minimise a loss function. Thanks to DP's efficiency and the modularity of primitives, NNs have achieved remarkable performance and success across domains such as computer vision, natural language processing, and scientific modelling. Although DP excels in smooth, real-valued, differentiable domains, many real-world optimisation problems (e.g., symbolic regression, neural architecture search, combinatorial design, robotic morphology, and parameter estimation in biological networks) involve discrete, hybrid, or structurally complex search spaces that are difficult, or even impossible, to optimise with gradient-based methods alone. To tackle such non-differentiable or discontinuous problems, Evolutionary Computation (EC) [4], characterised by population-based global search strategies, and Swarm Intelligence (SI) [5], inspired by collective behaviours in nature, have been defined. EC methods, such as Genetic Programming (GP) [6], Genetic Algorithms (GA) [7], Evolutionary Strategies (ES) [4], and SI techniques, such as Particle Swarm Optimisation (PSO) [8], excel at handling discrete structures, nonlinear interactions, and combinatorial spaces, navigating black-box landscapes without relying on gradient information. Indeed, both EC and SI leverage different sets of rules for generating (or updating) candidate solutions that are, in principle, agnostic to any characteristic of the fitness landscape, thereby limiting the negative impact of nonlinearity and ruggedness of the landscape [9], [10]. However, classical EC and SI methods typically do not exploit the rich local gradient information present in differentiable regions of the search space, thereby limiting their efficiency and scalability. Recent work has begun to bridge this gap by introducing gradient-aware evolutionary algorithms that incorporate differentiable relaxations and hybrid gradient components. Early work in differentiable GP (e.g., Differentiable Cartesian Genetic Programming) [11] and exact symbolic derivatives for GP [12] paved the way for incorporating DP into symbolic regression. Authors introduced automatic differentiation within GP trees, enabling efficient tuning of numerical constants and opening new ways for symbolic regression and differential equation solving. Zeng *et al.* [13] and Anthes *et al.* [14] extended these ideas using continuous symbolic tree relaxations and transformer-based semantic operators, respectively, enabling gradient-informed evolution within high-dimensional symbolic domains. In neural architecture search (NAS), hybrid

approaches such as SpiderNet [15], CGP-NAS [16], and dCGPANN [17] combine discrete evolutionary exploration with differentiable surrogates or relaxed encodings, enabling rapid discovery of efficient neural models. Gradient-based methods have also been integrated into evolutionary strategies themselves: GENNES [18] uses generative neural networks to parameterise ES sampling distributions, while Evolution Transformer [19] meta-learns update rules via Transformer architectures for in-context generalisation across tasks, showing remarkable generalisation capabilities across diverse black-box tasks. Similarly, in PSO, hybrid algorithms that combine global particle exploration with local gradient descent (GD) methods have shown improved convergence in continuous settings; e.g., for photovoltaic estimation [20] and neural control applications [21]. These strategies use PSO for exploration and GD for fine-tuning, demonstrating the synergy between swarm and gradient-based optimisation. In evolutionary robotics and morphological optimisation, integrating differentiable physics simulation into EC strategies has yielded substantial improvements. Differentiable physics has also catalysed innovation in evolutionary robotics. Kurenkov and Maksudov reduced sample complexity in policy search via gradients from simulators [22], Horibe *et al.* evolved soft robots with gradient-based damage recovery [23], and Strgar *et al.* evolved entirely differentiable robotic morphologies for smoother training and fabrication [24].

More recently, the idea of making algorithm control parameters themselves differentiable has been explored in a few works. In [25], the crossover and bound-constraint operators of DE are replaced with smooth approximations, yielding a variant (smoDE) whose parameters can be tuned via gradient descent. An adaptive extension (ada-smoDE) embeds an NN that outputs parameters on a per-generation basis, achieving competitive performance on the CEC 2018 suite. However, smoDE targets a single algorithm and a single aspect (parameter tuning); the internal evolutionary operators (selection, survival) remain non-differentiable, and the approach does not generalise to other metaheuristics such as PSO, GA, or CMAES. In a different work, Bohdal *et al.* [26] proposed an identically named method—also called *EvoGrad*—for efficient gradient-based meta-learning and hyperparameter optimisation in deep learning. Their approach uses evolutionary-inspired perturbations to estimate hypergradients without computing second-order derivatives, thereby reducing the cost of meta-learning. Although the name coincidence, the scope and goals are fundamentally different. Their method addresses hyperparameter optimisation of NNs (e.g., data augmentation, learning rate schedules) and does not consider the differentiability of evolutionary operators or the design of a general-purpose differentiable metaheuristic framework. Despite these advancements, existing approaches either use EC- and SI-based processes as the primary components of the method, sidelining GD and gradient information in local searches (e.g., memetic approaches), or target only a single operator or algorithm, thereby falling short of a unified, fully differentiable EC and SI optimisation framework in which *all* components (e.g., representation, selection, combination) are integrated within a differentiable computational graph. Thus, in this paper, we present EvoGrad, a novel framework that fully integrates DP into SI and EC. Compared with prior art, the key contributions of EvoGrad are: (*i*) it provides the first *unified* differentiable reformulation of four distinct metaheuristics (PSO, GA, DE, CMAES), whereas existing methods address at most one algorithm; (*ii*) it makes *every* component differentiable—representation, operators, hyperparameters, and selection—rather than only algorithm-level parameters; and (*iii*) it enables end-to-end gradient-based adaptation within a single computational graph, facilitating future integration with meta-learning and differentiable simulation pipelines. EvoGrad is distributed as a GPU-powered Python package, fully implemented in PyTorch, with a unified API across methods. In addition to classical (i.e., non-differentiable) methods, EvoGrad offers their corresponding differentiable variants by reframing the entire optimisation process with differentiable components and operators: 1) **Differentiable Representation**: individuals (e.g., particles, chromosomes) are learnable tensors parametrised within a differentiable model, allowing for local GD-based refinements; 2) **Differentiable Operators**: mutation, crossover, and swarm operators implemented as differentiable layers, with learnable hyperprameters; 3) **Differentiable Setting of Hyperparameters**: the method hyperprameters and its operators (e.g., social, cognitive factors in PSO, mutation probs) are learnt during the optimisation loop by means of GD-based methods; 4) **Differentiable Selection and Fitness Assignment**: population selection is implemented via softmax and Gumbel–Softmax distributions, enabling backpropagation through parent selection and replacement. We tested EvoGrad on the CEC 2017 benchmark suite, which was completely rewritten in PyTorch, and on multi-basin benchmark functions, and compared it with their classical counterparts. In addition, we assessed its effectiveness on a real-world Parameter Estimation (PE) problem for a biochemical system.

To support reproducibility and its use, EvoGrad is publicly available on GitHub [1].

II. Methods

In this work, we revisit four canonical population-based metaheuristics, namely, PSO [8], real-coded GAs [7], Differential Evolution (DE) [27], and Covariance Matrix Adaptation Evolution Strategy (CMAES) [28], and introduce EvoGrad, a differentiable and modular reformulation of these methods. EvoGrad offers a promising, differentiable alternative to classical metaheuristics, providing a foundation for future advances in learning-based optimisation. Classical metaheuristics are typically treated as black-box search procedures, which prevents gradient-based adaptation of algorithmic hyperparameters and limits integration with differentiable pipelines (e.g., meta-learning). EvoGrad addresses this by rewriting stochastic operators as reparameterised (pathwise) transformations and by relaxing discrete decisions (masks, categorical

[1] https://github.com/andreatangherloni/EvoGrad

sampling, ranking) via continuous surrogates. This enables gradients to propagate through the optimiser itself, allowing end-to-end learning of both candidate solutions and hyperparameters (e.g., inertia, mutation strength, crossover rate, selection temperature). From an algorithmic standpoint, a key novelty of EVOGRAD is its modular design. More specifically, EVOGRAD adopts a modular architecture inspired by `pymoo` [29] and organised in the following structures: (*i*) a `Problem` abstraction encapsulates the objective, bounds, and optional constraints; (*ii*) `Algorithm` implements a shared *infill/advance* pattern, decoupling offspring generation from state updates; (*iii*) reusable `Operators` implement sampling, selection, crossover, mutation, survival, repair, and differentiable relaxations; (*iv*) `Termination` and `Callbacks` provide standard runtime controls (e.g., history, early stopping, and logging). This organisation makes algorithms interchangeable through a common interface and supports extensions via operator composition rather than fixed implementations. Importantly, both `operators` hyperparameters (e.g., mutation scales, crossover logits, selection temperatures, covariance factors) and the population itself are represented as differentiable quantities (i.e., PyTorch `nn.Parameters`). Consequently, operator behaviour and individual positions can be jointly adapted via standard backpropagation and a GD-based optimiser, thereby enabling adaptive operators and a learnable population within a single autodiff graph.

A. Problem setting

We consider D-dimensional bounded continuous optimisation problems where the global best is $\mathbf{x}^\star \in \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$, where $\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{l} \le \mathbf{x} \le \mathbf{u}\}$ with $\mathbf{l}, \mathbf{u} \in \mathbb{R}^D$ denoting the coordinate-wise lower and upper bounds. In this setting, population-based methods evolve a population of candidate solutions $\mathbf{X}^{(t)} = \{\mathbf{x}_i^{(t)}\}_{i=1}^N$ using stochastic variation and selection to balance exploration and exploitation. Note that we summarise the classical operators only as needed to introduce their differentiable counterparts.

B. Differentiable reformulation

Pathwise sampling: The central step is to express random draws as deterministic transforms of parameter-free noise. For a distribution $p_\theta(\mathbf{x})$, depending on parameters θ, we sample $\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})$ and set $\mathbf{x} = g_\theta(\boldsymbol{\varepsilon})$, yielding a differentiable Monte-Carlo objective $\tilde{L}(\theta) = f(g_\theta(\boldsymbol{\varepsilon}))$ with gradients obtained by automatic differentiation [30]. Gaussian noise is reparameterised by $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and affine maps; uniform noise uses $\mathbf{u} \sim \mathcal{U}(0,1)$.

Relaxing discrete decisions: Many metaheuristic operators rely on non-differentiable discrete variables, such as mutation masks, crossover decisions, parent indices, and ranking. We replace Bernoulli variables by the Gumbel–Sigmoid (Binary-Concrete) relaxation

$$\tilde{m} = \sigma\left(\frac{1}{\tau}\big(\log u - \log(1-u) + \alpha\big)\right), \quad u \sim \mathcal{U}(0,1),$$

with temperature $\tau > 0$ and learnable logit α [31]. To preserve the original discrete semantics at runtime, we apply a straight-through (ST) estimator: the forward pass uses the hard decision $m = \mathbb{I}[\tilde{m} > 0.5]$, while the backward pass propagates gradients through $\tilde{m}$. Categorical sampling (e.g., tournament/parent selection) is relaxed with Gumbel–Softmax [32]; in our experiments, we use the soft selection probabilities during training to stabilise gradients. Note that we compared Gumbel–Softmax, RELAX, and ARM in terms of gradient variance and runtime, measured on the Ackley function in 30 dimensions. In this setting, Gumbel–Softmax and RELAX exhibited nearly identical variance ($\sim 4.34e{-}7$), but Gumbel–Softmax was 22% faster and required only a single forward pass, while RELAX required two. By contrast, ARM exhibited variance that was more than five orders of magnitude higher (0.0933), confirming its unsuitability for high-dimensional continuous objectives.

C. Unified differentiable optimisation loop

EVOGRAD views each generation as a differentiable computation followed by a state commit. The population $\mathbf{X}^{(t)} \in \mathbb{R}^{N\times d}$ is treated as learnable parameters, together with algorithm hyperparameters (stored in unconstrained form, e.g., on log scales). Importantly, operator controls (e.g., mutation scales, crossover logits, selection temperatures, covariance parametrisations) are also exposed as differentiable parameters. Each generation performs the following three steps:
1) `Infill`: generate offspring and evaluate fitness using reparameterised operators. 2) `Gradient step`: backpropagate a scalar loss and update all learnable parameters (both the population and operator/hyperparameters) using a GD-based optimiser. 3) `Advance`: commit the evolutionary state update (survival/replacement, best tracking, bound repair).

By representing operators and individuals within the same autodiff graph, EVOGRAD supports adaptive operators and a learnable population that co-evolve under gradient feedback, blending global stochastic exploration with gradient-driven adaptation. Note that *Adaptive* variants adjust evolutionary hyperparameters based on gradients of the objective but rely on purely stochastic variation; *Differentiable* (Diff) variants keep fixed hyperparameters while using gradients of the objective to guide population refinements; *Full* variants combine both mechanisms, jointly adapting hyperparameters and exploiting gradient information to perform population modifications.

D. Differentiable instances

Differentiable PSO: In EVOGRAD, we retain the standard PSO update [8], but parametrise coefficients as learnable variables (optionally per-particle). For each particle i:

$$\mathbf{v}_i^{(t+1)} = \omega_i \mathbf{v}_i^{(t)} + c_{1,i}\mathbf{r}_1 \odot (\mathbf{p}_i - \mathbf{x}_i^{(t)}) + c_{2,i}\mathbf{r}_2 \odot (\mathbf{g} - \mathbf{x}_i^{(t)}),$$
$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)},$$

with $\mathbf{r}_1, \mathbf{r}_2 \sim \mathcal{U}(0,1)^d$ sampled pathwise. Note that bound handling is applied via a repair operator.

Differentiable GA: We implement real-coded GA operators [7] with differentiable variation. SBX [33] is reparametrised through uniform noise, and the crossover distribution index is learned via $\log \eta_c$. Mutation uses a Binary-Concrete mask per gene, a differentiable polynomial mutation with learnable $\log \eta_m$, and mutation-rate logits [34]. This

yields gradients with respect to both the population and GA hyperparameters (i.e., crossover/mutation controls).

Differentiable DE: We differentiate common DE mutation strategies [27], including *DE/rand/1* and *DE/current-to-best/1*, by learning the scale factor as $F = \exp(\phi)$ with ϕ optimised by gradients. Parent indices are obtained via a Gumbel–Softmax relaxation [32]. Binomial crossover uses a Binary-Concrete mask with a learnable logit for the crossover rate. Greedy replacement is retained; when a hard decision prevents backpropagation, an ST estimator is used to provide a learning signal to the meta-optimiser. To guarantee explicit elitism, the global best is injected after replacement by overwriting the worst individual.

Differentiable CMAES Following standard CMAES [28] (with full path updates [35]), we learn the mean $\boldsymbol{\mu}$, step-size $\log \sigma$, and Cholesky factor $\mathbf{L}$ (parameterising the covariance) and sample via the exact reparameterisation $\mathbf{x} = \boldsymbol{\mu} + \sigma \mathbf{L}\mathbf{z}$, $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. To avoid non-differentiable ranking, recombination weights are computed from a temperature-controlled softmax over fitness values, while discrete indicators in the evolution-path update are smoothed with logistic gates. A small amount of detached jitter is used to improve numerical stability in Cholesky operations.

Remark on algorithm-specific relaxations: The differentiable reformulations described above employ *different* relaxation primitives depending on each algorithm's structure. PSO relies solely on pathwise reparameterisation of its velocity coefficients; GA uses reparameterised SBX together with Binary-Concrete mutation masks; DE combines Gumbel–Softmax parent selection with Binary-Concrete crossover masks and an ST estimator for greedy replacement; CMAES uses Cholesky reparameterisation with softmax-weighted recombination and logistic-smoothed path updates. These differences are inherent to the algorithms themselves, not design inconsistencies: indeed, each relaxation is chosen as the minimal, tightest surrogate that preserves the original operator semantics while enabling gradient flow. The shared pathwise-gradient formalism and the common three-step loop (`Infill–Gradient step–Advance`) ensure that all variants remain comparable despite using different relaxation primitives.

III. Results

A. CEC 2017 benchmark suite

We tested EvoGrad differentiable methods against standard implementations of the considered methods on the CEC 2017 benchmarking suite in 30 and 100 dimensions [36][2]. In particular, we compared the Adaptive, Diff, and Full variants of DE, GA, PSO, and CMAES with a baseline implementation of these methods and, for a fair comparison, their versions implemented in `pymoo`. We selected configurations that closely match their implementations (e.g., by disabling restarts and hyperparameter tuning). To assess statistically significant differences, we performed 30 independent runs for each method, using the same random seeds to enable paired comparisons. It is worth noting that we assessed statistically significant differences among our variants. Results were compared using the one-sided Wilcoxon signed-rank test with $\alpha = 0.05$, testing the directional hypothesis H_1 that "the differentiable variants achieve lower (i.e., better) fitness than the classical baseline." For each algorithm family (i.e., DE, GA, PSO, and CMAES), we compared three variants (Adaptive, Diff, and Full) against the classical baseline across all 29 CEC 2017 functions, yielding 87 comparisons per algorithm. To control the false discovery rate under multiple testing, p-values were adjusted using the Benjamini-Hochberg procedure applied separately within each algorithm family. The choice to perform this comparison is motivated by our goal of providing an initial framework that integrates DP into canonical standard methods, paving the way for further improvements to the approach in both single- and multi-objective problems. For each benchmark function, the search space was set to $[-100, 100]^D$, as stated in the competition, and each method used 100 individuals/particles until a total of $10000 \cdot D$ fitness evaluations was performed. EvoGrad variants employed a dual-optimisation strategy, using stochastic GD (SGD) with momentum for population-level local search and the Adam optimiser for adaptive hyperparameter learning. Algorithm-specific default learning rates and gradient clipping thresholds were selected based on preliminary tuning. Gradient norms were clipped separately for population variables and hyperparameters to ensure stability. For adaptive components, a ReduceLROnPlateau scheduler (factor = 0.5) was applied, triggered after 100 generations/iterations without improvement. This choice of SGD for population updates and Adam for hyperparameters reflects their complementary roles. SGD with momentum yields stable, conservative steps for high-dimensional candidate solutions, whereas Adam's per-parameter adaptive rates are better suited to the lower-dimensional hyperparameter space. The learning-rate scheduler, gradient clipping, and the use of 30 independent runs with paired statistical tests (Wilcoxon signed-rank, Benjamini–Hochberg correction) jointly mitigate the risk of overfitting and ensure that reported improvements reflect genuine performance gains rather than favourable random seeds or unstable gradient dynamics.

All experiments were conducted on a workstation equipped with 2 Intel Xeon Gold 6152 CPUs and an NVIDIA RTX 4090 GPU. Notably, EvoGrad implementations executed approximately $3\times$ faster than the equivalent `pymoo` baselines running on CPU, despite incorporating gradient-based refinement. In what follows, we focus on a representative subset of the CEC 2017 benchmarking suite (due to space limitations). Function F3 is unimodal, F5 and F8 are simple multimodal functions, F10, F14 and F18 are hybrid functions, and F22, F25 and F29 are composition functions that combine different standard benchmark landscapes. In 30 dimensions, the results reveal some interesting patterns (data not shown). First, CMAES consistently achieves the best performance across most functions, particularly on hybrid (i.e., F14, F18) and

[2]We did not consider the function F2, as it was removed from the competition due to definition issues

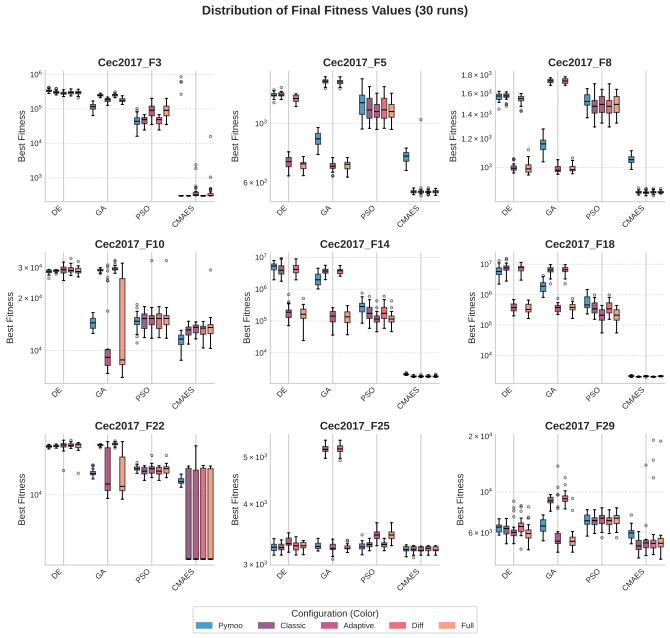

Fig. 1. Distribution of the best solutions found over 30 runs on a subset of the CEC 2017 benchmark suite in $100D$ by the `pymoo` implementation (blue), the Classic variant implemented in EvoGrad (purple), and the three differentiable variants: Adaptive (pink), Diff (red), and Full (orange).

composition functions (i.e., F22, F25), where it outperforms all other algorithms often by orders of magnitude. Second, DE shows strong performance, with EvoGrad variants matching or exceeding `pymoo` on several functions (i.e, F3, F14, F18). For GA, `pymoo` achieves better results on some functions (i.e., F5, F10, F22), though the EvoGrad differentiable variants often improve upon the Classic baseline within the same algorithm family. PSO variants perform consistently across all configurations, with minimal differences between `pymoo` and EvoGrad implementations. Comparing the differentiable variants (Adaptive, Diff, Full) against their Classic EvoGrad baselines, we observe that incorporating gradient information frequently improves performance, particularly for GA and DE. On F14, all GA differentiable variants outperform the Classic baseline, while on F18, DE differentiable variants achieve notably better results. The Full variant, which combines both adaptive hyperparameter learning and population refinement, shows the most consistent improvements across algorithms. Results on $100D$, reported in Figure 1, confirm the trends observed in the $30D$ setting. CMAES remains the strongest baseline on hybrid and composition functions, particularly on F14 and F18, where it outperforms all other algorithms by a wide margin. As dimensionality increases, the advantage of EvoGrad over `pymoo` becomes more pronounced: some DE variants consistently outperform `pymoo` on several functions, including F5, F8, F14, and F18, indicating favourable scaling behaviour. The differentiable variants exhibit trends consistent with the $30D$ experiments and generally match or improve upon their classical counterparts. In contrast to lower-dimensional settings, the Adaptive and Full configurations tend to provide the best trade-off between performance and stability in $100D$, frequently achieving lower median fitness and reduced variance compared to both classical and Diff variants, particularly for DE and PSO. These results suggest that combining gradient information with online hyperparameter adaptation becomes increasingly beneficial as problem dimensionality grows. The differentiable variants showed statistically significant improvements in 31% of comparisons overall, with substantial variation across algorithms. GA benefited most from gradient-based refinement, achieving significant improvements in 70.1% of comparisons, followed by DE at 46.0%. In contrast, PSO (6.9%) and CMAES (1.1%) showed minimal gains, likely because these algorithms already incorporate adaptive mechanisms that partially overlap with the gradient-based refinement. Among the differentiable variants, Full achieved the highest improvement rate (41.4%), followed by Adaptive (35.3%) and Diff (16.4%), suggesting that combining both hyperparameter learning and population refinement yields the greatest benefit. These results should be interpreted in context: the CEC 2017 benchmark suite was designed to challenge gradient-based methods through non-separable, multimodal, and discontinuous landscapes. Despite this, the differentiable variants never performed substantially worse than their classical counterparts while frequently outperforming them. This validates EvoGrad's design principle: gradient-based refinement can be safely integrated into EAs, enabling end-to-end differentiability for downstream applications such as hyperparameter meta-learning and integration with neural network pipelines, while simultaneously improving optimisation performance on a significant fraction of problems.

B. Multi-Basin Benchmark Functions

To further evaluate the effectiveness of differentiable EAs for problems that require both global exploration and local exploitation, we introduce a family of *multi-basin benchmark functions*. These functions create landscapes with multiple basins of attraction, each containing a single global optimum, and aggregate them using a smooth, differentiable aggregation operator. This design creates problems where (*i*) pure gradient methods converge to suboptimal basins based on initialisation, (*ii*) classical EAs can explore globally but struggle with local exploitation, while (*iii*) differentiable EAs leverage both capabilities synergistically. The foundation of our multi-basin construction is the *log-sum-exp* smooth minimum operator. Given K function values $f_1(x), \ldots, f_K(x)$, the smooth minimum is defined as:

$$f(\mathbf{x}) = -\tau_f \log \left(\sum_{k=1}^{K} \exp \left(-\frac{f_k(\mathbf{x})}{\tau_f} \right) \right),$$

where $\tau_f > 0$ is the *temperature* parameter controlling approximation sharpness. The log-sum-exp operator satisfies several important properties: 1) **Smooth approximation**: As $\tau_f \to 0^+$, the *log-sum-exp* smooth minimum operator converges pointwise to $\min_k f_k(\mathbf{x})$, while larger τ_f values produce smoother transitions between basins, 2) **Differentiability**: Unlike the hard minimum, $f(\mathbf{x})$ is differentiable everywhere, 3) **Bounds**: $\min_k f_k(\mathbf{x}) \leq f(\mathbf{x}) \leq \min_k f_k(\mathbf{x}) +$

TABLE I

RESULTS ON MULTI-BASIN RASTRIGIN ($D = 30$, BOUNDS $[-5, 5]^D$. BEST MEAN VALUE IN **BOLD**.

Configuration	Best	Mean	Std	Time (s)
CMAES Classical	0.00	2.22	3.04	25.66
CMAES Differentiable	0.00	1.49	2.16	9.77
CMAES Adaptive	0.00	**0.99**	**1.36**	45.24
CMAES Full	0.00	1.29	2.12	**7.94**
Adam (pop-based)	116.41	153.77	13.98	3.88

$\tau_f \log K$. Crucially, unlike the non-differentiable $\min$ operator, the *log-sum-exp* smooth minimum operator is differentiable everywhere, allowing gradient information to flow across basin boundaries. This enables gradient-based refinement to operate effectively once the population identifies promising regions.

Multi-Basin Rastrigin

The *Multi-Basin Rastrigin* function combines global funnel structure with local multimodality. Each basin contains a Rastrigin function, creating $\mathcal{O}(10^D)$ local minima per basin.

Basin Definition: Each basin $k \in \{1, \ldots, K\}$ is a Rastrigin function centred at $\mathbf{c}_k \in \mathbb{R}^D$ with offset $\delta_k \geq 0$:

$$f_k(\mathbf{x}) = A{\cdot}D + \sum_{i=1}^{D} \left[(x_i - c_{k,i})^2 - A\cos\left(2\pi(x_i - c_{k,i})\right)\right] + \delta_k$$

where $A = 10$ is the Rastrigin amplitude. The cosine term creates approximately 10^D local minima within each basin, with the basin's global minimum at $\mathbf{x}_k^* = \mathbf{c}_k$ achieving value $f_k(\mathbf{c}_k) = \delta_k$.

Multi-Basin Construction: In our tests, we used $K = 3$ basins, temperature $\tau_f = 5$, and search domain $[-5, 5]^D$, with the following configuration: 1) **Global basin** ($k = 1$): $\mathbf{c}_1 = \mathbf{0}$, $\delta_1 = 0$; 2) **Distractor basins** ($k = 2, 3$): $\mathbf{c}_2 = -2{\cdot}\mathbf{1}$, $\mathbf{c}_3 = 2{\cdot}\mathbf{1}$, with $\delta_k \sim \mathcal{U}[5, 15]$.

Optimisation Challenge: This benchmark presents a *two-scale* challenge: globally, the optimiser must identify the correct basin among K competing funnels; locally, it must navigate Rastrigin's multimodal landscape. Pure GD-based algorithms converge to the nearest local minimum within the nearest basin. Classical EAs can explore basins but struggle with local exploitation, while differentiable EAs combine population-based basin discovery with gradient-accelerated local refinement. We evaluated the Multi-Basin Rastrigin using EVOGRAD's CMAES variants, which achieved the best overall performance across the CEC 2017 benchmark suite (see Section III-A) and consistently outperformed DE, PSO, and GA on both unimodal and multimodal functions. Table I shows the results on the Multi-Basin Rastrigin benchmark in $D = 30$ dimensions, averaged over 30 independent runs with a budget of 150,000 fitness evaluations.

To ensure a fair comparison, we evaluated Adam on a population of $N = 100$ parallel solutions, matching the CMAES population size. Each solution is initialised uniformly in $[-5, 5]^D$ and updated independently via gradient descent on the mean population loss. This *multi-start* approach provides Adam with multiple opportunities to explore different basins. The Adam (pop-based) baseline achieves a mean fitness of 153.77, which is more than 2 orders of magnitude worse than EVOGRAD's CMAES variants. This confirms our hypothesis that Adam converges to local minima within distractor basins and cannot escape. The low standard deviation (13.98) indicates consistent failure across all 30 runs, showing that Adam reliably finds *some* local minima, but never the global optimum. All CMAES variants achieve a best fitness of 0.00 across 30 runs, demonstrating that population-based exploration successfully identifies the global basin in all configurations. The adaptive variant (learnable hyperparameters, no gradient flow through the population) achieves the lowest mean fitness (0.99) and standard deviation (1.36), indicating superior average performance and greater consistency. This suggests that learned adaptation of CMAES parameters (e.g., step size) is particularly beneficial for navigating Rastrigin's complex local structure. While all configurations reach the global basin, the differentiable and full variants converge significantly faster. The full configuration achieves the best trade-off: competitive mean fitness (1.29) with the fastest runtime, demonstrating that combining adaptive hyperparameters with gradient flow provides complementary benefits. Thus, EVOGRAD's population-based search successfully identifies the global basin, with differentiable variants offering substantial speedups.

C. Parameter Estimation of Biochemical Systems

To complement the synthetic multi-basin benchmarks with a real-world application, we evaluated EVOGRAD on the Parameter Estimation (PE) problem for biochemical systems [37], [38]. PE is a fundamental challenge in Systems Biology, where kinetic parameters governing reaction rates must be inferred from experimental time-series data. This problem is known to generate fitness landscapes that differ substantially from standard benchmark functions [37], making it an ideal test case for assessing the practical applicability of optimisation algorithms. We considered the Brusselator model, a theoretical model for autocatalytic chemical reactions that exhibits oscillatory behaviour [39]:

$$\frac{dX}{dt} = A - (B+1)X + X^2Y \qquad \frac{dY}{dt} = BX - X^2Y \tag{1}$$

where X and Y denote the concentrations of 2 intermediate species, and the kinetic parameters $\boldsymbol{\theta} = (A, B)^\top$ control the reaction rates. Given a discrete-time target series (DTTS) $\{(t_f, \mathbf{Y}(t_f))\}_{f=1}^{F}$ consisting of F observations at time points $t_1, \ldots, t_F$, the PE problem seeks parameters $\boldsymbol{\theta}$ minimising the discrepancy between observed and simulated dynamics. We used the *Mean Squared Error* (MSE) as the fitness function:

$$\mathcal{F}(\boldsymbol{\theta}) = \frac{1}{F \cdot N} \sum_{f=1}^{F} \sum_{q \in \{X,Y\}} \left(X_q^{\boldsymbol{\theta}}(t_f) - Y_q(t_f)\right)^2 \tag{2}$$

where $X_q^{\boldsymbol{\theta}}(t_f)$ denotes the simulated concentration of species q at time t_f using parameters $\boldsymbol{\theta}$, $Y_q(t_f)$ is the corresponding observed value, F is the number of time points, and N is the number of species.

Based on the comprehensive comparison made in [37], which demonstrated that PSO consistently outperforms other meta-heuristics on PE problems, we opted to test our EVO-

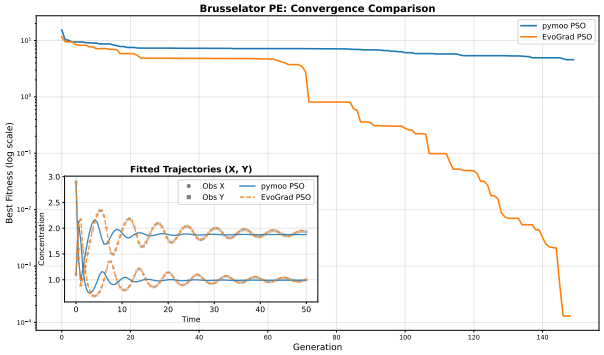

Fig. 2. PE on the Brusselator model comparing pymoo's classical PSO (blue) with EvoGrad's differentiable PSO (orange). **Main panel**: Convergence curves showing best fitness (MSE) over 150 generations (i.e., $4,500$ fitness evaluations). EvoGrad achieves a final fitness of $\sim 10^{-4}$, outperforming pymoo (~ 5) by over four orders of magnitude. **Inset**: Fitted trajectories for species X and Y. Grey markers denote observed data (circles: X, squares: Y); solid lines show pymoo's fit, dashed lines show EvoGrad's fit.

Grad's PSO against pymoo's PSO. For a fair comparison, all methods used identical population sizes (i.e., $N = 30$), initialised with a *uniform distribution*, and evaluation budgets (i.e., 4,500 fitness evaluations, corresponding to 150 generations). The 2 implementations differ in their ODE integration backends. The pymoo-based PSO employs SciPy's `solve_ivp` with the `DOP853` integrator (8th-order Dormand-Prince) [40]. In contrast, EvoGrad's PSO leverages `torchdiffeq` [41] with the `dopri8` integrator, enabling gradient flow through the ODE solver via automatic differentiation. Figure 2 presents the convergence dynamics and fitted trajectories. The results show a striking difference: while pymoo's classical PSO plateaus at a fitness of approximately 5, EvoGrad's PSO achieves a fitness below 10^{-4}—an improvement of over 4 orders of magnitude. The inset panel confirms that EvoGrad's estimated parameters accurately reconstruct the observed damped oscillatory dynamics of both species X and Y, whereas pymoo's solution exhibits visible deviations from the observed data.

IV. Conclusions

We introduced EvoGrad, a unified differentiable framework for classic population-based metaheuristics, including CMAES, PSO, GAs, and DE. EvoGrad re-parameterises each stochastic operator (e.g., mutation, crossover) and sampling strategy, enabling end-to-end gradient-based learning over all algorithmic components, from the population to the algorithm's hyperparameters. This differentiable reformulation transforms classical metaheuristics into fully differentiable modules, thereby enabling new opportunities for hybrid gradient-based and gradient-free optimisation. Through a consistent pathwise gradient formalism, we showed that (*i*) sampling from multivariate Gaussians (as in CMAES) can be expressed using reparameterised Cholesky transforms; (*ii*) crossover and mutation operators (e.g., SBX and polynomial mutation) can be relaxed with Binary-Concrete and Gumbel–Sigmoid variables, enabling differentiable masks; (*iii*) parent selection, typically implemented via discrete tournaments or stochastic rankings, can be approximated using the Gumbel–Softmax trick, allowing soft, differentiable mixtures of individuals; (*iv*) inertia and social-cognitive parameters in PSO, and crossover/mutation rates in GA and DE, can be made learnable and optimised via backpropagation. Our results show that EvoGrad differentiable methods consistently compete or outperform their classical counterparts. With EvoGrad, we reframed classical metaheuristics as differentiable modules, thereby enabling meta-learning, adaptive search, and hybrid optimisation pipelines. The main limitation of EvoGrad lies in the memory cost of unrolled optimisation, which is required to backpropagate through time across multiple generations. While our meta-loss formulation circumvents this in the single-step case, full credit assignment over long horizons (e.g., in curriculum learning or reinforcement learning) will benefit from truncated backpropagation or learned meta-optimisers. For instance, EvoGrad's PSO (Full version) has a peak memory footprint of 0.1 GB and approximately 30 MFLOPs per generation on a task with 5,000 dimensions, using 100 individuals. Another open question is how best to balance exploration and exploitation in this differentiable setting. While classic algorithms rely on fixed heuristics (e.g., inertia decay or covariance adaptation), our framework allows learning these schedules, which may require new regularisers or control mechanisms to ensure stability and generalisation. As future work, we plan to extend EvoGrad along several directions. First, we aim to incorporate more advanced variants of DE and CMAES, such as SHADE [42] and CatCMA [43], as well as other complex metaheuristics like the Evolutionary Algorithm for COmplex-process oPtimization (EACOP) [44]. We also plan to perform a comprehensive testing of all variants on a larger set of benchmarks from several benchmarking suites and in additional dimensions. Second, we intend to generalise EvoGrad to *multi-objective optimisation*, where differentiable Pareto dominance ranking and diversity-preserving mechanisms (e.g., crowding distance) could enable gradient-based learning of trade-off preferences and reference point adaptation in algorithms such as NSGA-II [45] and MOEA/D [46]. Third, we plan to address *discrete combinatorial problems* by extending our relaxation techniques to permutation-based representations (e.g., for travelling salesman or scheduling problems) using differentiable sorting networks [47] and Sinkhorn operators [48], as well as binary encodings via straight-through estimators. Finally, we plan to extensively test and apply EvoGrad to large-scale, high-dimensional problems (i.e., more than $1{,}000D$) for training neural networks, to improve convergence speed and identify better parameter configurations.

Acknowledgments

This work was partially supported by the MUR under the grant "Dipartimenti di Eccellenza 2023-2027" of the Department of Informatics, Systems and Communication of the University of Milano-Bicocca, Milan, Italy, and by the National Plan for NRRP Complementary Investments (established with the decree-law May 6, 2021, n. 59, converted by law n. 101

of 2021) in the call for the funding of research initiatives for technologies and innovative trajectories in the health care sectors (Directorial Decree n. 931 of 06-06-2022) – project n. PNC0000003 – AdvaNced Technologies for Human-centrEd Medicine (project acronym: ANTHEM). This work reflects only the authors' views and opinions, neither the Ministry for University and Research nor the European Commission can be considered responsible for them.

REFERENCES

[1] M. Blondel and V. Roulet, "The elements of differentiable programming," *arXiv preprint arXiv:2403.14606*, 2024.

[2] A. G. Baydin *et al.*, "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, vol. 18, no. 153, pp. 1–43, 2018.

[3] S. Scardapane, "Alice's adventures in a differentiable wonderland–volume i, a tour of the land," *arXiv preprint arXiv:2404.17625*, 2024.

[4] T. Bäck *et al.*, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997.

[5] R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm intelligence*. Elsevier, 2001.

[6] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, pp. 87–112, 1994.

[7] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, ieee, 1995.

[9] Z.-H. Zhan *et al.*, "A survey on evolutionary computation for complex continuous optimization," *Artificial Intelligence Review*, vol. 55, no. 1, pp. 59–110, 2022.

[10] A. P. Engelbrecht *et al.*, "The influence of fitness landscape characteristics on particle swarm optimisers: Ap engelbrecht et al.," *Natural Computing*, vol. 21, no. 2, pp. 335–345, 2022.

[11] D. Izzo *et al.*, "Differentiable genetic programming," in *European conf. on genetic programming*, pp. 35–51, Springer, 2017.

[12] P. Rockett *et al.*, "d (tree)-by-dx: Automatic and exact differentiation of genetic programming trees," in *Hybrid Artificial Intelligent Systems*, pp. 133–144, Springer, 2019.

[13] P. Zeng *et al.*, "Differentiable genetic programming for high-dimensional symbolic regression," *arXiv preprint arXiv:2304.08915*, 2023.

[14] P. Anthes *et al.*, "Transformer semantic genetic programming for symbolic regression," in *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 952–960, 2025.

[15] R. Geada and A. S. McGough, "Spidernet: Hybrid differentiable-evolutionary architecture search via train-free metrics," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1962–1970, 2022.

[16] C. Garcia-Garcia *et al.*, "Continuous cartesian genetic programming based representation for multi-objective neural architecture search," *Applied Soft Computing*, vol. 147, p. 110788, 2023.

[17] M. Märtens and D. Izzo, "Neural network architecture search with differentiable cartesian genetic programming for regression," in *Proc. of the Genetic and Evolutionary Computation Conference Companion*, pp. 181–182, 2019.

[18] L. Faury *et al.*, "Improving evolutionary strategies with generative neural networks," *arXiv preprint arXiv:1901.11271*, 2019.

[19] R. Lange *et al.*, "Evolution transformer: In-context evolutionary optimization," in *Proc. of the Genetic and Evolutionary Computation Conference Companion*, pp. 575–578, 2024.

[20] E. Zulu, R. Hara, and H. Kita, "An efficient hybrid particle swarm and gradient descent method for the estimation of the hosting capacity of photovoltaics by distribution networks," *Energies*, vol. 16, no. 13, p. 5207, 2023.

[21] D. A. Ejigu and X. Liu, "Gradient descent-particle swarm optimization based deep neural network predictive control of pressurized water reactor power," *Progress in Nuclear Energy*, vol. 145, p. 104108, 2022.

[22] V. Kurenkov and B. Maksudov, "Guiding evolutionary strategies by differentiable robot simulators," *arXiv preprint arXiv:2110.00438*, 2021.

[23] K. Horibe *et al.*, "Regenerating soft robots through neural cellular automata," in *European Conference on Genetic Programming (Part of EvoStar)*, pp. 36–50, Springer, 2021.

[24] L. Strgar *et al.*, "Evolution and learning in differentiable robots," *arXiv preprint arXiv:2405.14712*, 2024.

[25] H. Zhang *et al.*, "A gradient-based method for differential evolution parameter control by smoothing," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 423–426, 2024.

[26] O. Bohdal *et al.*, "EvoGrad: Efficient gradient-based meta-learning and hyperparameter optimization," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021.

[27] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, pp. 341–359, 1997.

[28] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[29] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020.

[30] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *Advances in neural information processing systems*, vol. 28, 2015.

[31] X. Geng *et al.*, "How does selective mechanism improve self-attention networks?," *arXiv preprint arXiv:2005.00979*, 2020.

[32] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[33] K. Deb *et al.*, "Simulated binary crossover for continuous search space," *Complex systems*, vol. 9, no. 2, pp. 115–148, 1995.

[34] K. Deb and D. Deb, "Analysing mutation schemes for real-parameter genetic algorithms," *International Journal of Artificial Intelligence and Soft Computing*, vol. 4, no. 1, pp. 1–28, 2014.

[35] A. Auger and N. Hansen, "Tutorial cma-es: evolution strategies and covariance matrix adaptation," in *Proc. of the Genetic and Evolutionary Computation Conference Companion*, pp. 827–848, 2012.

[36] N. Awad, , *et al.*, "Problem definitions and evaluation criteria for the cec 2017 special session and competition on single objective bound constrained real-parameter numerical optimization," in *Technical report*, pp. 1–34, Nanyang Technological University Singapore Singapore, 2016.

[37] A. Tangherloni *et al.*, "Biochemical parameter estimation vs. benchmark functions: A comparative study of optimization performance and representation design," *Applied Soft Computing*, vol. 81, p. 105494, 2019.

[38] C. G. Moles, P. Mendes, and J. R. Banga, "Parameter estimation in biochemical pathways: a comparison of global optimization methods," *Genome research*, vol. 13, no. 11, pp. 2467–2474, 2003.

[39] I. Prigogine and R. Lefever, "Symmetry breaking instabilities in dissipative systems. ii," *The Journal of Chemical Physics*, vol. 48, no. 4, pp. 1695–1700, 1968.

[40] E. Hairer, G. Wanner, and S. P. Nørsett, *Solving ordinary differential equations I: Nonstiff problems*. Springer, 1993.

[41] R. T. Chen *et al.*, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[42] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proc. of IEEE Congress on Evolutionary Computation*, pp. 71–78, IEEE, 2013.

[43] R. Hamano *et al.*, "Catcma with margin: Stochastic optimization for continuous, integer, and categorical variables," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 737–745, 2025.

[44] A. Tangherloni *et al.*, "A modified eacop implementation for real-parameter single objective optimization problems," in *Proc. of IEEE Congress on Evolutionary Computation*, pp. 01–08, IEEE, 2024.

[45] K. Deb *et al.*, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[46] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on evolutionary computation*, vol. 11, no. 6, pp. 712–731, 2007.

[47] A. Grover, , *et al.*, "Stochastic optimization of sorting networks via continuous relaxations," *arXiv preprint arXiv:1903.08850*, 2019.

[48] G. Mena *et al.*, "Learning latent permutations with gumbel-sinkhorn networks," *arXiv preprint arXiv:1802.08665*, 2018.