The following publication X. Wu, S. -H. Wu, J. Wu, L. Feng and K. C. Tan, "Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap," in IEEE Transactions on Evolutionary Computation, vol. 29, no. 2, pp. 534-554, April 2025 is available at https://doi.org/10.1109/TEVC.2024.3506731.

Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap

Xingyu Wu, Sheng-Hao Wu, *Member, IEEE*, Jibin Wu, *Member, IEEE*, Liang Feng, *Senior Member, IEEE*, Kay Chen Tan, *Fellow, IEEE*

***Abstract*—Large language models (LLMs) have not only revolutionized natural language processing but also extended their prowess to various domains, marking a significant stride towards artificial general intelligence. The interplay between LLMs and evolutionary algorithms (EAs), despite differing in objectives and methodologies, share a common pursuit of applicability in complex problems. Meanwhile, EA can provide an optimization framework for LLM's further enhancement under black-box settings, empowering LLM with flexible global search capacities. On the other hand, the abundant domain knowledge inherent in LLMs could enable EA to conduct more intelligent searches. Furthermore, the text processing and generative capabilities of LLMs would aid in deploying EAs across a wide range of tasks. Based on these complementary advantages, this paper provides a thorough review and a forward-looking roadmap, categorizing the reciprocal inspiration into two main avenues: LLM-enhanced EA and EA-enhanced LLM. Some integrated synergy methods are further introduced to exemplify the complementarity between LLMs and EAs in diverse scenarios, including code generation, software engineering, neural architecture search, and various generation tasks. As the first comprehensive review focused on the EA research in the era of LLMs, this paper provides a foundational stepping stone for understanding the collaborative potential of LLMs and EAs. The identified challenges and future directions offer guidance for researchers and practitioners to unlock the full potential of this innovative collaboration in propelling advancements in optimization and artificial intelligence. We have created a GitHub repository to index the relevant papers: https://github.com/wuxingyu-ai/LLM4EC.**

***Index Terms*—evolutionary algorithm (EA), large language model (LLM), optimization problem, prompt engineering, algorithm generation, neural architecture search**

I. Introduction

In recent years, large language models (LLMs[1]) [1]–[3] have achieved notable research breakthroughs, showcasing remarkable performance in the field of natural language processing [4]. As the scale of these models expands, LLMs showcase not only excellence in language-related tasks but also reveal expansive potential applications in diverse domains [5]. This includes a spectrum of optimization and generation tasks, representing a pivotal milestone in the evolution of artificial general intelligence. The advancement of LLMs has also catalyzed progress in technologies and methodologies across various research field [6]–[10]. Notably, this impact extends to evolutionary computation, offering both new opportunities and challenges. The primary goal of this review is to explore the dynamic interplay and synergies between LLMs and evolutionary algorithms (EAs), with the intention of establishing a complementary relationship between the two within the contemporary era of LLMs.

The LLM and EA, despite substantial disparities in objectives and principles, share a common pursuit of applicability in various scenarios, which are different from most models that aimed for high performance in specific domain problems. LLM achieves a unified approach across diverse tasks by learning from extensive data [11], while EA, as a general-purpose solver, has lower reliance on problem characteristics and information compared to traditional mathematical optimization methods [12], enabling it to solve a wider range of problems with different characteristics. Therefore, in terms of application scenarios, both EAs and LLMs demonstrate unique advantages in addressing complex problems with vast search spaces and uncertain environments [5], [13]. This similarity suggests potential complementarity and mutual inspiration between LLM and EA when dealing with large-scale and complex problems.

Although LLM has achieved success in various applications, it has still faced criticism attributable to its black-box nature and inflexible searching. Due to the intricate LLM architecture, the specific details of the internal decision-making, reasoning, and generation processes are either uninterpretable or invisible for most users [14], especially in the case where commercially viable LLMs (such as GPT-4 [15]) typically keep their model parameters private. Exactly, as a classic black-box optimization technique [16], EAs hold potential for further enhancement within the black-box framework of LLM, such as prompt optimization [17] or neural architecture search (NAS) [18]. Another limitation of LLM is its finite search capability, as the search process is typically conducted in a one-shot manner without iterative progressive optimization. Moreover, the search capability of LLMs is constrained by prompts and training data, leading to a tendency to generate content that aligns with learned patterns and prompt information [19], thereby limiting global exploration of the

This work was supported by the National Key R&D Program of China (2022YFC3801700), the Research Grants Council of the Hong Kong SAR (Grant No. PolyU25216423, PolyU11211521, PolyU15218622, PolyU15215623, and C5052-23G), The Hong Kong Polytechnic University (Project IDs: P0043563, P0046094), and the National Natural Science Foundation of China (Grant No. 62306259 and U21A20512). (Corresponding author: Jibin Wu and Liang Feng.)

Xingyu Wu, Sheng-Hao Wu, Jibin Wu, and Kay Chen Tan are with the Department of Data Science and Artificial Intelligence, The Hong Kong Polytechnic University. Jibin Wu is also affiliated with the Department of Computing, The Hong Kong Polytechnic University. (email: xingy.wu@polyu.edu.hk, shenghao.wu@polyu.edu.hk, jibin.wu@polyu.edu.hk, kctan@polyu.edu.hk).

Liang Feng is with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: liangf@cqu.edu.cn).

[1]This paper views the LLM as the Transformer-based language model with a large number of parameters, pretrained on massive datasets using self/semi-supervised learning techniques.

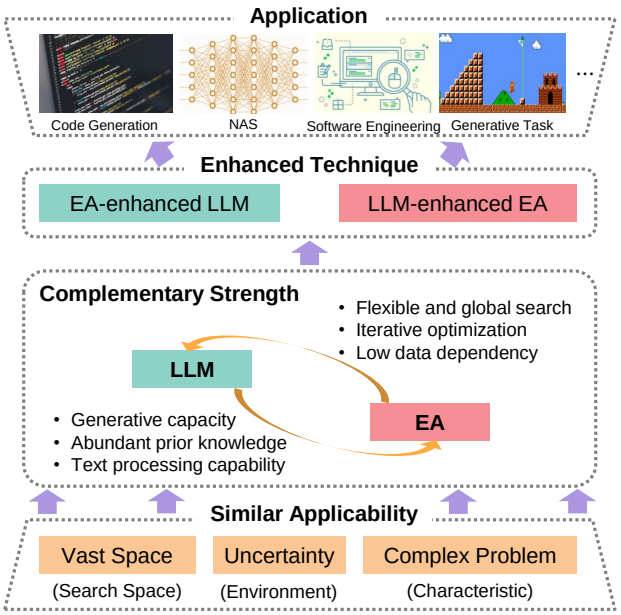


Fig. 1. A general framework for integrated research of LLMs and EAs.

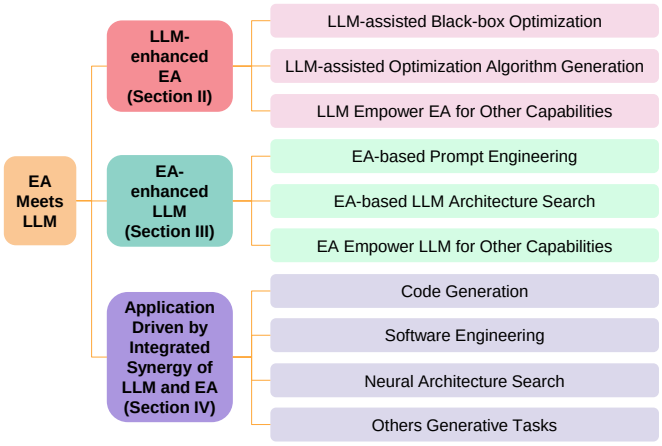


Fig. 2. Categorization of research works surveyed in this paper.

entire search space. EA's search superiority can mitigate this limitation in LLM. Firstly, EA is an iterative optimization method that can continuously evolve and improve the solutions generated by LLM, thus enhancing result quality. Additionally, EA can achieve more flexible search through well-designed searching space and evolutionary operator [20], [21]. The search capacity of EA proves particularly advantageous for complex tasks that require adequate optimization. This is highly relevant for the case of LLMs, which often require extensive tuning of hyperparameters and prompts to achieve peak performance [22], [23].

On the other hand, LLMs demonstrate superiority in general knowledge, text understanding, and generative capacity. These strengths can compensate for certain limitations encountered when applying EA independently. Firstly, taking benefit from pretraining on extensive text datasets, LLMs contain a wealth of knowledge across various domains, which can provide effective guidance during the EA search process. Although EA has powerful global search capabilities, it often requires more search steps to achieve desirable results due to the lack of task-related knowledge [24], particularly in situations with a challenging search space or encompass a diversified population [25]. Research has shown that LLM can offer valuable information at the early search stage, aiding EA in faster converging to promising solutions [24]. Additionally, another inherent strength of LLMs lies in their remarkable text understanding and generation capability. Since EA is typically designed for numerical problems and require additional encoding or preprocessing steps to adapt to text-related tasks [26], integrating LLM with EA obviously facilitates more convenient utilization of EA's algorithmic principles in tasks involving text generation and processing [23], [27]. Moreover, under the evolutionary framework, LLMs hold significant promise for a wide range of tasks that involve content generation, like prompt generation [17] and algorithm generation [28].

Given the above complementary advantages, the potential collaboration between EA and LLM has gained increasing attention from researchers and practitioners recently. Current research primarily focuses on three aspects to enhance mutual performance and jointly drive application developments, as shown in Fig. 2:

1) *LLM-enhanced Evolutionary Optimization:* LLMs can serve not only as evolution operators, leveraging rich domain knowledge to accelerate the search process, but can also utilize their code generation capabilities to enhance EAs at the algorithmic level.
2) *EA-enhanced LLM:* The black-box optimization advantage of EAs can aid LLMs in prompt engineering. This, in turn, enhances the outputs of LLMs with improved prompts. Additionally, the search capability of EAs can optimize the neural architecture of LLMs, resulting in versatile and lightweight LLMs.
3) *Applications Driven by Integrated Synergy of LLM and EA:* The collaborative synergy between LLM and EA has revolutionized numerous conventional application scenarios, including NAS, code generation, software engineering, and text generation.

In this paper, we present a comprehensive review and a forward-looking roadmap for the cross-disciplinary study of LLMs and EAs. We provide a detailed categorization of rapidly evolving areas, conduct a thorough review, and identify emerging directions. Our primary contributions are summarized as follows:

1) We present a systematic survey of the current state of cross-disciplinary research between LLMs and EAs. Through the analysis of existing methods, we comprehensively review related research progress and applications. To the best of our knowledge, this paper is the first comprehensive and up-to-date review specifically focused on the EA research in the era of LLMs.
2) We propose a comprehensive taxonomy that classifies the current cross-disciplinary research on LLMs and EAs into three distinct categories. This systematic categorization allows for a clear and organized overview

of this emerging field, enabling existing methods to be appropriately placed within their respective categories.
3) Through critical analysis on the strengths and weaknesses of the existing methods, we identify several key challenges in the cross-disciplinary research of LLMs and EAs. These findings serve as a source of inspiration for future investigations in this promising area.

The remainder of this paper is organized as follows. Sections II and III delve into the synergistic research on the complementary strengths of LLM and EA, specifically examining how LLM enhances EA and how EA enhances LLM. In Section IV, a comprehensive review is provided on the application domains that have been significantly influenced by the combined advancements of LLM and EA. Based on the extensive survey of existing research, Section V highlights several key areas that warrant future investigation and provides a roadmap for further exploration. Finally, Section VI concludes this paper.

II. LLM-enhanced EA

LLMs hold immense promise for solving optimization problems. With their robust natural language understanding and generation capabilities, LLMs can effectively handle complex problem descriptions and constraints. Currently, there are two primary approaches for leveraging LLMs to assist in solving optimization problems: (1) The first approach uses LLMs as black-box search operators for optimization problems. This approach leverages the LLM's generation ability to create novel solutions. A summary of this approach can be found in Table I. (2) The second approach leverages the representation power and generation abilities of LLMs to generate novel optimization algorithms for solving specific problems, as shown in Table II.

A. LLM-assisted Black-box Optimization: As Search Operator

The potential of LLMs in solving optimization problems has been extensively demonstrated by numerous studies, including two empirical study for comprehensive evaluations [29], [30]. In particular, LLMs as search operators in black-box optimization have emerged as a prominent research area. Several studies have validated its strong optimization abilities in solving small-sized problems. Specifically, these studies utilize LLMs to generate the next generation of solutions for both single-objective and multi-objective optimizations.

1) Single-objective optimization: Yang *et al.* first discovered that LLMs have the ability to progressively improve solutions in optimization tasks when provided with the problem and past trajectory in natural language [31]. They proposed Optimization by PROmpting (OPRO) to leverage LLMs as optimizers. OPRO is primarily applicable in the absence of gradients, where optimization problems are described in natural language and serve as meta-prompts. In each iteration, LLMs use the previously generated solutions and their values as prompts to generate new solutions, which are then evaluated and added to the prompt for the next iteration. While this study does not explicitly define the 'crossover' and 'mutation' operators in evolutionary optimization, the role of LLMs in black-box optimization, particularly OPRO's emphasis on utilizing optimization trajectories to help LLMs identify patterns of high-quality solutions, can be applied in evolutionary optimization. For instance, a similar study in evolutionary optimization is Language Model Crossover (LMX) [32], which employs LLMs to generate new offspring solutions from parent solutions represented as text. LMX acts as an evolutionary variation operator by prompting the LLM with concatenated parents and interpreting its output as offspring. LMX exhibits properties such as heritability of traits from parents to offspring and universality in representing any genetic operator. The authors demonstrate the performance of LMX in various scenarios.

TABLE I
SUMMARY OF LLM-ASSISTED BLACK-BOX OPTIMIZATION

Type	Method	Ref.
Evaluation	N/A	[29]
	N/A	[30]
Sing-objective	OPRO	[31]
	LMX	[32]
	LMEA	[24]
	EvoLLM	[33]
	Huang *et al.*	[34]
	LEO	[35]
	Application for soft robots	[36]
	Application for visual representation	[37]
Multi-objective	Decomposition-based MOEA	[38]
	QDAIF	[39]
	In-context QD	[40]
	CMOEA-LLM	[41]

Similar to the aforementioned research, some researchers have made improvements to the evolutionary optimization process based on LLM from different aspects. Liu *et al.* propose the LLM-driven EA (LMEA) [24], which not only uses LLM to perform crossover and mutation operations, but also constructs a prompt in each generation to guide the LLM in selecting parent solutions from the current population. Lange *et al.* propose EvoLLM and focus on prompt design to transform LLMs into evolution strategies for black-box optimization [33]. Huang *et al.* further leveraged GPT-4's multimodal capabilities [34]. In addition to textual prompts, they additionally provide the LLM with visual prompts representing the layout of the nodes in capacitated vehicle routing problem, to further boost performance. Brahmachary *et al.* propose the Language-model-based Evolutionary Optimizer (LEO) for population-based EAs, which focuses on the balance between exploration and exploitation [35]. LEO divides the solution set into exploration and exploitation pools. It utilizes LLMs to generate new solutions for the two pools separately based on different prompts. Then, LEO uses an elitism selection strategy to guide the evolutionary direction, which imports the solutions with the minimum objective function values from the exploration pool into the exploitation pool, while removing solutions with the maximum objective function values from the exploitation pool. In addition, LLM-guided EAs have demonstrated practicality in various real-world applications, such as the co-design of soft robot's morphology and control [36], as well as the interpretable and discriminative attribute representation for visual classification [37].

2) Multi-objective optimization: In the field of multiobjective EAs (MOEAs), Liu *et al.* utilize a decomposition-based MOEA framework and employ LLMs as black-box search operators to generate new offspring individuals for each subproblem [38], through prompt engineering and in-context learning. Moreover, they further design an explicit white-box linear operator based on interpreting the behavior of the LLM to approximate its results, and validate its generalization in experiments. Another example of applying LLMs to multi-objective evolutionary optimization is Quality-Diversity (QD) search, namely QD through AI Feedback (QDAIF) algorithm [39]. It uses LLMs to evaluate the quality and diversity of generated solutions, rather than relying on manually designed metrics, allowing it to be applied to more complex qualitative domains. The EA is responsible for maintaining the solution library, and replacing newly generated higher quality and more diverse solutions into the relevant positions in the library based on the evaluation of the LLM, realizing an iterative optimization search process. Another similar application of LLMs in QD problems is In-context QD [40].

Unlike previous methods, Wang *et al.* utilized the fine-tuned LLM as an evolutionary operator to generate 10% of offspring and accelerate the convergence rate of the population [41]. Specifically, the training samples provided to the LLM contained feasible and infeasible solutions with different qualities, along with the decision variable values, objective function values, and constraint violation degrees of each solution. Additionally, a specialized prompt language was designed to clarify the LLM's task of producing new solutions. The prompt language emphasized that new solutions should simultaneously consider objective function optimization and constraint satisfaction. According to the prompt language, the LLM learned how to select two solutions from the input samples and generate completely new offspring based on them through recombination or other operations. Through iterative training, the LLM continuously optimized its ability to produce high-quality solutions and more efficiently solve constrained multi-objective optimization problems.

Existing research has shown that LLMs have potential for small-scale optimization problem. Compared with traditional EAs, LLM can understand the optimization problems and the expected properties of solutions using natural language, which is more direct and simple than formally defining problems and implementing operators through programming. It also avoids the need for additional training and can generalize to different problems. Moreover, the rich prior knowledge of LLM can realize operators that are difficult to design manually in algorithm design, providing stronger exploration ability for the algorithm, which may even surpass artificially designed operators in this respect. In addition to optimizing results, the utilization of evolution operators based on LLMs has shown significant benefits in terms of efficiency. Several studies [24], [32], [33], [35] have demonstrated that LLM-guided evolution outperforms random optimization in terms of efficiency. Notably, research conducted by [32], [35] reveals that LLM-guided evolution operators achieve search efficiency comparable to manually designed genetic operators. Furthermore, as the scale of the LLM increases, the performance of LLM-guided evolution operators improves, and their evolutionary efficiency advances alongside the progress of the LLM itself. Additionally, [24] highlights that controlling the temperature of the LLM to balance exploration and exploitation can further enhance the search efficiency of LLM-guided evolution. Moreover, [33] showcases the potential advantages of LLM-based evolution operators in terms of initial convergence speed, and efficiency can be further enhanced by incorporating optimization hints. LLM also brings EA advantages such as flexible input and output scales, good memory of the optimization process, and better zero-shot learning effects. However, applying LLMs to practical complex optimization problems poses major challenges [30]. Problems with high dimensions, constraints, and precision require interactions that exceed LLMs' context abilities. Additionally, current evaluations focus narrowly and consider limited factors, insufficient to demonstrate LLMs' full optimization capabilities. Overall, while initial studies are promising, significant barriers remain for applying LLMs to real-world complex optimization problems.

TABLE II
SUMMARY OF LLM-ASSISTED OPTIMIZATION ALGORITHM GENERATION

Method	Generated Algorithm or Target Problem	Ref.
Pluhacek *et al.*	Hybrid swarm intelligence optimization algorithm	[42]
OptiMUS	Mixed-integer linear programming problem	[43]
ZSO	Zoological search optimization algorithm	[44]
AEL	Heuristic algorithm	[45]
ReEvo	Heuristic algorithm	[46]
LLM_GP	Genetic Programming	[47]
SR-EAD	Evolutionary strategy or evolution transformer	[48]
EvolCAF	Cost-aware Bayesian optimization	[49]
Kramer	Evolution Strategies	[50]
LLMOPT	Multi-objective Optimization	[51]

B. LLM-assisted Optimization Algorithm Generation

In addition to using LLMs as evolutionary operators as mentioned earlier, some studies have leveraged LLMs for algorithmic optimization. By automatically generating optimization algorithms, LLM provides powerful support for solving optimization problems. At the beginning, the code generation methods were primarily *single-round*, relying on additional debug steps to optimize the code. After the emergence of [28], researchers started to use EA to achieve *iterative generation*.

1) Single-round Generation: Pluhacek *et al.* used an LLM to generate hybrid swarm intelligence optimization algorithms [42]. Specifically, the LLM first selects suitable candidate algorithms for the given problem and analyzes the components of each algorithm, then automatically designs the hybridization approach while providing reasoning and code implementation. Later, OptiMUS further develops the role of LLMs specifically for mixed-integer linear programming (MILP) problems [43], enabling automated optimization problem modeling and solving. It leverages LLMs to achieve automation throughout the problem-modeling and solving phases, including mathematical modeling, algorithm selection, code writing, debugging, and validity checking. Another recent study focused on the animal-inspired metaheuristic domain [44].

2) Iterative Generaiton: In the nascent stages of LLM development, Lehman *et al.* pioneered the integration of EA

into the field of algorithm generation [28][2]. They leveraged the iterative optimization framework of EA to systematically refine algorithms, employing LLMs as the evolutionary operators that directly manipulate code text. This innovative approach paved the way for a surge of subsequent studies [21], [52]–[54], which have since expanded the application of this framework to generate algorithms and code across a diverse range of domains, including the optimization domain. Following this framework, Liu *et al.* proposed Algorithm Evolution using LLM (AEL) [45] for automated heuristic algorithm design, which treats the heuristic idea/thought as an independent evolutionary object, and optimizes it collaboratively with the code. Reflective Evolution (ReEvo) presented by Ye *et al.* improves this framework by introducing short-term and long-term reflection mechanisms to better use previously generated algorithms [46]. Other methods following this iterative manner have also emerged in different scenarios, e.g., the genetic programming (GP) [47], acquisition functions in cost-aware Bayesian optimization [49], evolution strategies [50], and multi-objective optimization [51].

3) EA Representation Learning-based Generation: Beside the aforementioned methods, Lange *et al.* propose a EA generation method based on the algorithm representation, called Self-Referential EA Distillation (SR-EAD) [48], to implement EA generation based on their own trained Evolution Transformer, a model characterizing various families of evolutionary strategies. Specifically, the model processes the population members, fitnesses, and search distribution statistics of each generation to output the update of the search distribution for the next generation. This model is trained with EA Distillation to clone existing black-box optimization algorithms, such as Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [55]. After training, the model can perform in-context evolutionary optimization on new tasks, demonstrating that it has learned general optimization principles. The parameters of the well-trained Evolution Transformer are randomly perturbed to generate multiple mutated model instances. These mutated model instances each run on a set of optimization tasks to generate multiple trajectories of self-optimization. The trajectories are filtered according to their optimization performance, and the better performing trajectory is selected. This optimization trajectory is used as the supervision signal to train the original Evolution Transformer model. After several iterations, it is hoped to find optimization strategies that are better than the original model.

These investigations primarily leverage the algorithm comprehension, representation, and generation capabilities of LLMs to enhance the performance of EAs. Rather than directly serving as search operators in the optimization process, LLMs are employed to refine EAs at the algorithmic level. This avenue holds significant promise, and in Section IV-A, we will provide a comprehensive overview of code generation techniques based on LLMs and EAs, as well as their challenges, including handling intricate problem descriptions, processing large volume of numerical inputs, and addressing efficiency concerns stemming from time-intensive interactions.

[2]The method presented in [28] does not concentrate exclusively on the automated generation of optimization algorithms, but applicable across a wide range of domains. Given that this subsection is specifically focused on the optimization algorithm generation, the discussion of other more general methods for algorithm/code generation, including the techniques in [28], will be detailed in Section IV-A.

C. LLM Empowering EA with Other Capabilities

In addition to using LLM as an evolutionary operator or directly generating optimization algorithms, LLM can also provide assistance for EAs from other aspects. Chen *et al.* proposed OptiChat [56] to diagnose the infeasible model, identifying the main constraints that cause the infeasibility. Specifically, the infeasible model means that the constraints of an optimization algorithm cannot be satisfied simultaneously [57], which are very common in practical applications, mainly because the model parameters are set incorrectly, or some conflicting new constraints are added. OptiChat uses LLMs to provide a natural language interface for non-expert users, making it more convenient to understand and repair infeasible optimization model problems. The user provides the definition of the optimization model, including decision variables, parameters, constraints, and objective function, by using the Pyomo algebraic modeling language [58]. OptiChat can provide explanations of the optimization model, identify potential sources of infeasibility, and offer suggestions to make the model feasible through interactive conversations.

Besides, Maddigan *et al.* leveraged LLMs to provide explainability for results of EA, improving understanding for non-expert users [59]. They focused on using GP for nonlinear dimensionality reduction, where GP individuals are tree expressions that directly map each new low-dimensional feature to combinations of high-dimensional features. Specifically, a system is constructed for running GP-based nonlinear dimensionality reduction processes and visualizing results. LLMs are integrated into the system to develop a conversational interface to provide explanations, using prompts containing dataset and tree expression information to provide contextual knowledge. They also built a vector repository utilizing vector similarity search to provide related literature knowledge to the model and address knowledge gaps. Another recent study uses LLM to help decision makers understand and interpret the optimal solution set in multi-objective evolutionary optimization [60].

III. EA-enhanced LLM

This section delves into the emerging research of enhancing LLMs through the application of EA. The primary focus of this exploration is twofold: EA-based prompt engineering and EA-based LLM architecture search, as summarized in Tables III and IV.

A. EA-based Prompt Engineering

Black-box prompt engineering [61] enables adjusting prompts without requiring access to the underlying model's parameters and gradients, making it particularly valuable for closed-source LLMs [62]. This advancement allows for effective optimization of closed-source models, overcoming previous limitations imposed by the unavailability of model

TABLE III
SUMMARY OF EA-BASED PROMPT ENGINEERING.

Type	Method	Used EA	Ref.
Discrete Prompt Optimization	GPS	GA	[63]
	GrIPS	Greedy search algorithm	[64]
	EvoPrompt	GA, Differential EA	[17]
	Plum	Metaheuristic algorithm	[65]
	PromptBreeder	GA	[66]
	SPELL	Any EA	[67]
	EoT prompting	Any EA	[68]
	iPrompt	Iteration similar to EA	[69]
	PhaseEvo	Designed framework	[70]
	InstOptima	NSGA-II	[71]
	EMO-Prompts	NSGA-II, SMS-EMOA	[72]
Gradient-Free Soft Prompt Optimization	BBT	CMA-ES	[73]
	BBTv2	CMA-ES	[74]
	Clip-Tuning	CMA-ES	[75]
	Shen *et al.*	CMA-ES	[76]
	BPT-VLM	CMA-ES	[77]
	Fei *et al.*	CMA-ES	[78]
Prompt Generation for Data Augmentation	Evol-Instruct	Mutation and selection	[79]
	Sun *et al.*	Any EA	[80]
Prompt Generation for Security	AutoDAN	GA	[81]
	Jailbreak Attacks	GA	[82]
	SMEA	Any EA	[83]
	Shi *et al.*	Any EA	[84]

parameters. Currently, evolutionary computation plays a role in two types of prompt engineering: discrete prompt optimization and continuous prompt optimization. The former considers meaningful textual instructions as prompts, as shown in Section III-A1, while the latter uses vectors composed of continuous numerical values as prompts, as shown in Section III-A2.

1) Textual Prompt Optimization: Before the emergence of ChatGPT, several research studies had already explored the use of metaheuristic algorithms, such as EAs, for prompt engineering in pretrained language models. Notable examples include Gradient-free Instructional Prompt Search (GrIPS) [64], which employed a greedy search strategy, and Genetic Prompt Search (GPS) [63], which utilized a Genetic Algorithm (GA) [85]. These studies typically employed EAs as the underlying search framework, with the LLM being responsible for generating and evaluating prompts. However, it is important to note that these investigations were limited in their scope and primarily focused on specific prompt engineering scenarios.

In order to fully unlock the potential of discrete optimization in the realm of black-box prompt optimization, Guo *et al.* proposed an EA-based discrete Prompt tuning framework (EvoPrompt) [17]. In EvoPrompt, the LLM emulates evolutionary operators using a fixed prompt and generates new candidate prompts through crossover and mutation based on the differences between parent prompts. The EA is used to guide the optimization process and retain the best prompts. EvoPrompt has been validated using two evolutionary methods: GA and Differential EA [86]. Pan *et al.* proposed a similar prompt optimization paradigm called Prompt Learning Using Metaheuristic (Plum) [65], which formulates prompt learning as a black-box non-convex discrete optimization problem, allowing various metaheuristic algorithms to be applied to help discover effective prompts. Beside the GA [85], this work implemented another five algorithms, including Hill Climbing, Simulated Annealing [87], Tabu Search [88], and Harmony Search [89]. Their experiments showed that Harmony Search was more efficient than GA, achieving better performance with fewer API calls.

In contrast to EvoPrompt's initialization approach (hand-designed task-specific prompts), Fernando *et al.* introduced PromptBreeder, an automated approach that utilizes LLM to optimize prompts based on problem descriptions [66]. PromptBreeder employs an EA framework to automatically explore prompts in the problem domain and simultaneously evolves task prompts and mutation prompts, rather than using fixed prompts in EvoPrompt. Additionally, PromptBreeder adopts diverse mutation operators, including zeroth-order, first-order, and higher-order mutations, to more thoroughly explore the space of prompt strategies. Cui *et al.* proposed a prompt optimization framework called PhaseEvo that enables joint optimization of prompt instructions and examples [70]. The framework adopts a four-phase optimization strategy and designs five different evolutionary operators to achieve different goals in each phase, including global initialization phase (using Lamarckian operator or semantic mutation operator to generate initial population), local feedback mutation phase (using feedback mutation operator to accelerate convergence), global evolution mutation phase (using EDA operator and crossover operator to escape local optima), and local semantic mutation phase (using semantic mutation operator to find the global optimum). Additionally, PhaseEvo uses performance-based vectors and Hamming distance to evaluate candidate similarity instead of commonly used semantic similarity. Other similar approaches include Semantic Prompt Evolution based on a LLM (SPELL) [67], zero-shot EoT prompting [68], and interpretable autoprompting (iPrompt) [69].

Moreover, Yang *et al.* argued that the quality of instructions should not be measured solely from a performance perspective; other objectives such as instruction length and perplexity can be considered as well [71]. Similar to the aforementioned studies, Yang *et al.* proposed InstOptima, which utilizes the ChatGPT to simulate instruction mutation and crossover operations. Under multi-objective optimization, InstOptima employs the NSGA-II algorithm [90] for non-dominated sorting and obtains a set of excellent instructions (the Pareto front) in terms of multiple objectives. EMO-Prompts proposed by Baumann *et al.* also optimizes prompts from an evolutionary multi-objective optimization perspective [72]. It showcases an interesting scenario of finding prompts that cause the model to generate text containing two emotions.

2) Gradient-Free Soft Prompt Optimization: In some closed-source LLMs, users can only access the model through the inference API and do not have access to the model parameters and gradient information. To optimize continuous prompts using limited samples in this scenario without relying on gradient-based optimization of model performance, Sun *et al.* proposed Black-Box Tuning (BBT) [73]. BBT represents the continuous prompts to be optimized as a vector with fixed initialization. It projects the prompt vector into a low-dimensional subspace using a random projection matrix to reduce the optimization difficulty. The CMA-ES is used to sample new values for this vector in the subspace, and the loss value of the vector is queried from the LLM to modify the distribution for the next iteration of CMA-ES. This iterative

process continues until the stop condition is met, where the obtained subspace vector is then added to the fixed initialization vector and projected to obtain the final prompt vector.

Several studies aimed to improving the BBT. Sun *et al.* proposed BBTv2 [74]. BBTv2 adds continuous prompt vectors not only at the input layer but also at each hidden layer in LLM, forming a deep prompt representation. This significantly increases the number of tunable parameters, which is advantageous for handling more challenging tasks. To optimize higher-dimensional deep prompts, BBTv2 employs a divide-and-conquer algorithm to alternately optimize prompt vectors at different levels, decomposing the original problem into multiple lower-dimensional subproblems. Additionally, the random projection matrix is no longer generated using a uniform distribution but is adapted to different models based on the statistical distribution of model word vectors or hidden states, better accommodating different models. To improve search efficiency and provide more fine-grained and multi-perspective evaluation feedback, Chai *et al.* replaced the method of reducing the space dimensionality by using a single random projection matrix in BBT with the Clip-Tuning approach [75]. Clip-Tuning applies Dropout sampling to the pre-trained model during inference, generating multiple sub-networks that can be viewed as projections of predictions for the samples in the original high-dimensional space. By blending rewards from multiple sub-network predictions, the search algorithm converges faster to the optimal solution. Shen et al. learned the complete distribution over soft prompts, rather than just considering a point estimate like BBT [76], in order to achieve uncertainty quantification of the predictive results. They used variational inference and ensemble learning to learn and approximate the posterior distribution over soft prompts. Similar to BBT, the EA in these studies is the CMA-ES. Some other studies focused on the vision-language models [77], [78], which are not detailed here.

In addition to prompt engineering aimed at optimizing prompts, the prompt generation can have broader applications, such as data augmentation for training LLM or conducting jailbreak attacks on LLM to test security. Sections III-A3 and III-A4 will introduce how EA plays a role in these two tasks.

3) Prompt Generation for Data Augmentation: In contrast to the aforementioned approaches that purely focus on prompt optimization, Xu *et al.* employed prompt data augmentation using LLM and EA, referred to as Evol-Instruct, to enhance instruction data [79]. The framework uses a small number of manually written seed instructions as the initial dataset, treats LLM as an instruction evolver, iteratively evolves instructions based on different evolutionary prompts (e.g., adding constraints, deepening), and uses LLM to generate instruction responses and filter out ineffective instructions. The evolved instructions are then added to the training data, and the process is repeated iteratively to generate diverse and high-quality instruction sets. Experimental results demonstrate that Evol-Instruct improves the performance of instruction-based models on various downstream tasks, such as language modeling and text classification. Sun *et al.* adopted a similar approach to build high-quality domain-specific instruction data, and designed a multidimensional quality evaluation method to assess the generated data [80].

4) Prompt Generation for LLM Security: Another improvement of LLM focuses on model security. Specifically, alignment techniques ensure the safety and interpretability of model outputs by collecting human-annotated data and reinforcement learning methods [91], making the models generate responses that are more in line with human values and expectations. However, with properly designed problem prompts, known as "jailbreak attacks", aligned LLMs can still generate inappropriate responses. Currently, handcrafted prompts for jailbreak attacks are difficult to automate and scale, while automatically generated prompts are often semantically incoherent and difficult to defend against semantics-based defenses [92]. Therefore, some studies use EA-based prompt engineering for automatic jailbreak attacks. Liu *et al.* model the jailbreak attack problem as an optimization problem and use EAs to automatically optimize prompts [81], namely Automatically generating DAN-series-like jailbreak prompts (AutoDAN). It utilizes existing handcrafted jailbreak prompts to initialize the population and designs an adaptive function suitable for structured discrete data like text. In addition, AutoDAN adopts a hierarchical GA to consider the optimization of prompts at the sentence and vocabulary levels, and designs a momentum word scoring mechanism to balance search ability and semantic coherence. The research by Lapid *et al.* achieves a similar goal to AutoDAN [82]. Zou *et al.* noted the importance of system messages in LLMs jailbreaking, as well as the transferability of jailbreaking prompts under different system messages. They proposed the System Message Evolutionary Algorithm (SMEA) [83] to search for optimized system messages with stronger resistance against jailbreaking attacks. Another study focused on testing the robustness of reliability detection systems for LLM-generated texts [84]. They utilized the protected auxiliary model ChatGPT to generate word substitution candidates, and performed query-based word substitution attacks based on EA. The search involved instructional prompts to change the generation style and make the detection system difficult to detect, which is essentially a prompt optimization process for adversarial purposes.

Based on the discussions above, EA-based prompt engineering has played a significant role in LLM. Researchers have used EAs as a search framework, combined with prompt generation and evaluation in both discrete and continuous prompt optimization. Additionally, EAs have also been applied to data augmentation and LLM security, further expanding the scope of prompt generation applications. However, current methods still face challenges such as the selection of initial populations, expanding search spaces, and method stability. Addressing these challenges will require better strategies for selecting initial populations, efficient search algorithms, and a deeper understanding of LLMs. Therefore, EA-based prompt optimization provides powerful tools and methods for improving LLM performance and expanding its applications. With further research and advancements, we can expect breakthroughs and innovations that will contribute to the progress of prompt optimization in LLM.

TABLE IV
SUMMARY OF EA-BASED LLM ARCHITECTURE SEARCH METHODS

Method	Main Task	Used EA	Evaluated LLM	Ref.
AutoBERT-Zero	Discover new universal LLM backbone from scratch	Any EA	BERT	[18]
SuperShaper	Search hidden unit dimensions in each Transformer layer	Any EA	BERT	[93]
AutoTinyBERT	Automatically optimize LLM architecture hyperparameters	Any EA	BERT	[94]
LiteTransformerSearch	Independently vary hyperparameters of each decoder layer	Any EA	GPT-2	[95]
Klein *et al.*	Various LLM architecture optimization methods	Multi-objective EA	BERT	[96]
Choong *et al.*	Obtain specialized models optimized for specific tasks	MO-MFEA	M2M100-418M, ResNet-18	[97]

B. EA-based LLM Architecture Search

Prompt engineering goes beyond the realm of LLM models by optimizing the input format to enhance the quality of model outputs. Differing from prompt engineering, another approach known as LLM architecture search focuses on directly optimizing the architecture of LLM models to achieve superior performance and lighter LLM models. With the increasing complexity and size of neural networks, manual design and optimization of architectures have become laborious and time-consuming tasks. EAs offer an effective approach to automate the search process and discover promising architectures [98], [99]. By employing evolutionary operators such as mutation, crossover, and selection, these algorithms can generate a diverse set of candidate architectures. This exploration-exploitation trade-off allows for a comprehensive exploration of the architecture space while gradually converging towards promising solutions. Previously, EA-based NAS methods were primarily applied to small-scale models and achieved promising results. Since the study by So *et al.* focusing on NAS for Transformers [100], EA-based NAS has been utilized on diverse large-scale models[3] as shown in Table IV.

The first work, using an NAS algorithm based on evolutionary search is AutoBERT-Zero [18], to discover a new universal LLM backbone from scratch. In AutoBERT-Zero, a well-designed search space is introduced, containing primitive math operations to explore novel attention structures in the intra-layer level, and leveraging convolution blocks as supplementary to attention in the inter-layer level. Additionally, this work proposed the Operation-Priority evolution strategy which utilizes prior information of operations to flexibly balance exploration and exploitation during search. Furthermore, a training strategy called Bi-branch Weight-Sharing is designed to speed up model evaluation by initializing candidates with weights extracted from a super-net. Different from the more complex search space in AutoBERT-Zero [18], SuperShaper proposed by Ganesan *et al.* [93] focuses on searching the hidden unit dimensions of each Transformer layer. This is achieved by adding bottleneck matrices between each Transformer layer, thus enabling the variability of hidden dimensions across layers. By slicing the bottleneck matrices, different structured sub-networks are sampled from the super network, and the hidden dimensions between layers are determined by the slicing. The sub-network search process considers optimizing two objectives simultaneously, including perplexity and latency, and trained predictors are used to approximate them. Finally, an EA is used to search for the optimal hidden unit shape distribution across layers that simultaneously meets the requirements of accuracy and latency.

Some methods try to achieve NAS on more complex search space, where key hyperparameters of each layer of Transformer are considered, such as hidden state dimension, number of attention heads, and feedforward network dimension. Yin *et al.* applied EA-based NAS methods for the first time to automatically optimize the architecture hyperparameters of LLMs [94]. The proposed AutoTinyBERT searches within the space of structures with identical layer depths and dimensions, thereby simplifying the search space from exponential to linear scale and greatly reducing the search complexity. In addition, this work trained a big model SuperPLM with one-shot learning that contains all potential sub-structures. Therefore, when evaluating a specific structure, they do not need to train it from scratch, but can directly extract the corresponding sub-model from SuperPLM to serve as high-quality initializations for various latent structures. Yin *et al.* used EAs to search, and designed a sub-matrix extraction method to quickly extract different structural sub-models from SuperPLM for evaluating structure performance, selecting elite structures and mutating to generate new generations, repeating this process to search for the optimal structure. LiteTransformerSearch proposed by Javaheripi *et al.* [95] also allows the hyperparameters of each decoder layer, such as hidden size, number of heads and feedforward dimensions, to vary independently, creating a heterogeneous search space. The search space also includes other hyperparameters like number of layers and embedding dimensions. LiteTransformerSearch leverages the empirical observation that decoder parameter count has a high correlation with validation perplexity, establishing the first training-free low-cost proxy for Transformer architecture search - decoder parameter count. It uses EAs to sample candidate architectures from the search space based on this proxy, while also measuring hardware metrics directly on the target device. This enables LiteTransformerSearch to perform multi-objective NAS to obtain a Pareto frontier estimation that optimizes perplexity, latency and memory. And its effectiveness has been evaluated on two popular autoregressive Transformer backbones GPT-2 and Transformer-XL. Similarly, Klein *et al.* also conduct NAS in multi-objective manner [96], which discover multiple subnetworks of LLMs that balance model performance and size.

Different from the above methods, Choong *et al.* did not directly study the NAS task, but proposed the concept of "Set of Sets" [97], which refers to a set of models that can simultaneously meet multiple task settings and resource

[3]The papers reviewed in this subsection mainly concentrate on designing NAS methods for pretrained language models (PLMs) without explicitly distinguishing between the concepts of LLMs and PLMs.

constraints. They studied how to obtain a set of smaller-scale models optimized for specific tasks from LLMs through multi-objective multi-task EAs (such as the MO-MFEA algorithm [101], [102]). Among them, the LLM plays the role of a general-purpose basic model. Experimental results show that the specialized models obtained in this way can achieve better performance or greater compression rates than the original large model in various application fields and neural network architectures.

In conclusion, EAs have shown great potential in assisting LLM architecture search. By leveraging EAs, researchers can automate the process of optimizing LLM architectures, which is otherwise laborious and time-consuming. EAs enable the generation of diverse candidate architectures through evolutionary operators such as mutation, crossover, and selection, striking a balance between exploration and exploitation. Several studies have successfully applied EA-based NAS methods to different aspects of LLM architecture, including discovering new universal LLM backbones, optimizing hidden unit dimensions, and tuning hyperparameters. However, there are still some challenges remaining, such as the high time consumption and the limited generalization ability. Future research should focus on addressing these challenges and exploring more effective LLM architecture search approaches.

C. EA Empowering LLM for Other Enhanced Capabilities

In addition to enhancing the performance of LLM through NAS, some researchers have utilized the search capability of EA to assist in improving other aspects of LLM.

Before the emergence of LLM, Kim *et al.* have studied the Length-Adaptive Transformer model [103], which can automatically adjust the sequence length according to different computational resource constraints during inference. They proposed LengthDrop technique, which allows the model to randomly reduce the sequence length of each layer during training, making the trained model more robust to changes in sequence length during inference. On the trained model, an EA is used to continuously optimize the population of length configurations to maximize accuracy under various computational budgets. During inference, the corresponding optimal length configuration is directly used without retraining the model to achieve efficient inference. Jiang *et al.* studied how to deploy generative inference services for large foundation models in a heterogeneous distributed environment to reduce the costs of centralized data centers [104]. Their research allows each pipeline parallel stage to be assigned with different numbers of Transformer layers, while also allowing each stage to set different degrees of tensor model parallelism. They used a GA with operations of merging, splitting, and swapping different pipeline groups. By combining with a dynamic programming algorithm to evaluate the costs of each scheme, they searched for the optimal solutions of pipeline group allocation, GPU device allocation within each pipeline group, and layer allocation for each pipeline stage. Ding *et al.* studied how to extend the context window of LLMs to 2048k tokens [105]. They discovered two forms of non-uniformities in the positional encoding - across different dimensions and token positions. Through an EA, they searched for the optimal rescaling factors for each dimension and initial positions. This allowed different dimensions and positions to be rescaled with different degrees of interpolation or extrapolation. It achieved better preservation of the original positional information compared to existing methods that uniformly handled all dimensions. Akiba *et al.* proposed a model merging approach that utilizes CMA-ES algorithm to optimize merged performance in both parameter space and data flow space [106], which can automatically discover effective combinations of diverse models from different domains. Li *et al.* propose to leverage a small model to store domain knowledge in order to assist LLMs in tackling the lack of domain-specific knowledge [107]. To learn and apply the knowledge, the small model undergoes domain-specific pretraining, knowledge instruction tuning, and Bayesian optimization, where CMA-ES is used to find soft prompts that optimizes the consistency between the outputs of the two models. This allows the small model to better satisfy the needs of the large model in downstream tasks. Moreover, the studies of self-evolution in LLMs have also widely adopted the ideas of EAs, which has been discussed in [108].

IV. Applications Driven by Integrated Synergy of LLM and EA

In recent years, the synergy between LLM and EA has attracted increasing attention. Researchers combine the strengths of LLM and EA to enhance performance in various downstream applications, as shown in Table V. This section provides a review from a problem-based perspective and discuss the collaborative effects of LLM and EA in several popular downstream applications.

A. Code Generation

LLMs and EAs have both shown promise in automating code generation. LLMs can be trained on vast amounts of publicly available source code to gain a broad understanding of programming concepts and patterns [109], [110]. However, their generation abilities are limited by their training data distribution. EAs, on the other hand, are capable of open-ended search through program spaces. But traditional mutation operators used in GP struggle to propose high-quality changes in a way that mimics how human programmers intentionally modify code. The joining of EAs and LLMs has opened up more opportunities for code generation.

1) Universal Code Generation: Lehman *et al.* applied LLMs into GP, called evolution through large models (ELM) [28]. ELM utilizes a small number of initially hand-written programs as seeds, combining the MAP-Elites search algorithm [134] with an LLM-based intelligent mutation operation to generate a large amount of functionally rich programs as design examples. This method improves the search efficiency of GP algorithms since the LLM trained on code promote a more intelligent and effective mutation operator and reducing the probability of random mutations generating meaningless code. Considering the LLM may achieve different performance improvements in different domains, these programs are used as a training set to train LLMs in order to guide

TABLE V
SUMMARY OF APPLICATIONS DRIVEN BY INTEGRATED SYNERGY OF LLM AND EA.

Category	Subcategory	Method	Description	Ref.
Code Generation	Universal Code Generation	ELM	Universal method for code generation	[28], [111]
		WizardCoder	Use Evol-Instruct to enhance the code generation	[112]
		Pinna *et al.*	Improve LLM-generated code by Grammatical Evolution	[113]
	Domain-specific Code Generation	SEED	Data cleaning tasks	[53]
		EUREKA	Design reward in reinforcement learning	[52]
		EROM	Design reward in reinforcement learning	[114]
		Diffusion-ES	Design reward in reinforcement learning	[115]
		GPT4AIGChip	Design AI accelerator	[54]
		FunSearch	For mathematical and algorithmic discovery	[21]
		L-AutoDA	For decision-based adversarial attacks	[116]
		LLM-SR	For scientific equation discovery from data	[117]
		Shojaee *et al.*	Node importance scoring functions in complex networks	[118]
	Security in Code Generation	DeceptPrompt	Generates code containing specified vulnerabilities	[119]
		G3P with LLM	Enhance code security	[120]
Software Engineering	Software Optimization	Kang *et al.*	Improves traditional genetic improvement	[121]
		Brownlee *et al.*	Improves traditional genetic improvement	[122]
	Software Testing	TitanFuzz	Test case generation	[20]
		CodaMOSA	Test case generation	[123]
	Software Project Planning	SBSE	Optimize selection of example sets	[124]
Neural Architecture Search	Representation Capability of LLM	GPT-NAS	Fine-tune GPT model to guide the NAS.	[125]
	Generation Capability of LLM	LLMatic	Generate code of architecture.	[126]
		Evoprompting	Architecture generation through soft prompt tuning.	[127]
		Guided Evolution	Use LLM to mutate and crossover the architecture code	[128]
		Others	Use LLM as an operator to generate new architectures	[129], [130]
		GPTN-SS	Employ GPT-4 to generate code of NAS algorithm	[131]
	Reasoning Capability of LLM	Jawahar *et al.*	Performance predictors of architecture.	[132]
		ReStruct	Use LLM as predictor and selector of structure	[133]

the LLM to output better code. Furthermore, by splicing new conditional inputs onto the LLM, it can teach the LLM to generate conditionally. This model will be fine-tuned through reinforcement learning to make the model able to generate appropriate outputs based on observed conditions. Bradley *et al.* proposed an open-source Python library based on ELM [111]. Luo *et al.* applied the Evol-Instruct introduced in Section III-A3 to generate a more complex and diverse training dataset [112] from the existing Code Alpaca instructions. They then fine-tuned the open-source code LLM StarCoder using this dataset, resulting in "WizardCoder" with the state-of-the-art performance on code generation. Different from the above studies, Pinna *et al.* did not directly apply evolutionary operators to code [113]. The code generated by LLMs are used as the initial individuals. And representing these individuals as abstract syntax trees, Grammatical Evolution algorithm continuously improve the code individuals and ultimately obtains the optimal code.

2) Domain-specific Code Generation: The combined approach of using LLM and EA has demonstrated practicality in various domains. These methods essentially leverage the iterative search framework of EA and the code generation and text understanding capabilities of LLM, with LLM acting as an evolutionary operator in the code evolution process. By iteratively improving the code, these methods enable the generation of algorithmic code by utilizing prompts that typically include contextual information such as environment source code, task descriptions, or example code. This approach has been applied to generate code for a wide range of tasks, including customized data cleaning methods [53], reward function design in reinforcement learning [52], [114], [115], automated AI accelerator design [54], mathematical and scientific discoveries [21], [117], decision-based adversarial attacks [116], and criticality evaluation of nodes in complex networks [118].

In these research studies, additional innovations have further enhanced the collaborative effect of LLM and EA. For example, as introduced by Ma et al., the Evolution-driven Universal REward Kit for Agent (EUREKA) tracks the changing values of each component in the reward function throughout the reinforcement learning training process [52]. It monitors metrics like the value curve of an individual penalty term over time. EUREKA generates a list of these tracked changes in a reward feedback text provided to the language model for reference. This allows the method to guide the language model towards targeted improvements to the reward function design. Additionally, EUREKA also permits humans to provide purely textual descriptions of the strengths and weaknesses of the current reward function's behavior as well as directions for desired behavioral changes. It takes these suggestions as context for the next round of search, generating reward functions that are better aligned with human preferences [52]. In Narin's work, they additionally utilize the capabilities of a multimodal LLM, GPT-4V, to comprehend the visual information in the environment and assist in the generation process [114]. The FunSearch method designed by Romera *et al.* uses an island-based EA strategy to split the program database into multiple subpopulations that evolve in parallel [21]. This can effectively avoid local optima and maintain exploration diversity. Moreover, clustering programs based on their signatures can effectively identify programs with similar functionality but different code implementations, thus preserving more diverse programs. This balancing of exploitation and exploration enhances the algorithm's ability to more comprehensively search the solution space and discover globally optimal solutions. The GPT for AI Generated Chip (GPT4AIGChip) proposed by Fu *et al.* introduces a demo-augmented prompt generator [54]. This generator can auto-

matically select the two most relevant demonstrations from the demonstration library that correspond to the input design instruction. It augments the selected demonstrations into the prompt as contextual information to effectively guide the LLM towards generating more accurate code. The method proposed by Mao *et al.* optimized various details including population design, management, evaluation, and experimental design [118].

3) Security in Code Generation: In addition to directly generating code, some studies have focused on the security of generated code. Wu *et al.* concerned about how to evaluate whether the generated code is vulnerable to attacks [119]. They proposed DeceptPrompt, which can generate adversarial natural language prefixes/suffixes to drive code generation models to produce functionally correct code containing specified vulnerabilities. DeceptPrompt continuously optimizes the prefixes and suffixes using GA, with LLM serving as the mutation operator. The fitness function is separately designed for the functionally correct part and the vulnerable part to guide the generated code to both retain functionality and contain the specified vulnerabilities. Empirical studies have shown that DeceptPrompt can successfully attack various popular code generation models. Tao *et al.* further considered methods to enhance the security of generated code [120]. They proposed a simple idea of combining LLM with Grammar-Guided GP (G3P) system [135], which enforces that any synthesized program adheres to Backus-Naur form (BNF) syntax, thus promoting the development/repair of incorrect programs and reducing the chances of security threats. Specifically, LLM is primarily employed to generate the initial code population in G3P, where the initial population is mapped to a predefined BNF syntax program before evolutionary search, allowing for improvements of these programs using program synthesis benchmarks. Afterwards, these programs will be handed over to G3P and evolved into better code.

B. Software Engineering

Due to the promising performance of LLM and EA in code generation tasks, some studies have further applied them to practical problems in software engineering [136], [137], including subtakss software optimization, software testing, and software project planning.

1) Software Optimization: Genetic Improvement (GI) [22] is a technique for automatically software optimization based on the ideas of evolutionary computation. Kang *et al.* leverage the advantages of LLM in code understanding and generation to improve the shortcomings of blind mutation in traditional GI [121]. The usage is similar to the code generation steps in Section IV-A, firstly letting the LLM improve the code for specific optimization objectives (such as time efficiency and memory consumption), then injecting the mutations generated by LLM into the candidate pool of GI, and continuously performing evolutionary operations until termination to obtain optimized code. Similarly, another study also enhanced the mutation operators of GI by leveraging LLMs to improve non-functional properties or fix functional bugs of software [122]. Brownlee *et al.* believed that the search space of GI is limited by the mutation operators it uses. By treating LLMs' suggestions as additional mutation operators, they aimed to enrich the search space and thus obtain more successful mutation results (generated code).

2) Software Testing: EAs have extensive applications in software testing techniques, where they are employed to search for effective test cases within the testing space to uncover errors and defects in software systems [138], [139]. Recently, some studies combine the code generation capability of LLMs and search capacity of EAs, and apply them to software testing. Deng *et al.* proposed TitanFuzz [20] to test for bugs in deep learning libraries. The core idea of TitanFuzz is to directly leverage LLMs that have been pretrained on tens of billions of code snippets, which implicitly learned the syntax and API constraints. It utilizes the generative ability of Codex [109] for seed generation and the infilling ability of InCoder [140] for mutation generation, to automatically produce a large number of input programs that meet the requirements for testing. The fitness function considers both the depth of the dataflow graph obtained from static analysis and the number of unique APIs called in the program, as well as the number of repeated API calls (as a penalty). This encourages the generation of programs with more complex API usage and richer API interactions. Lemieux *et al.* took a more direct approach [123], namely CodaMOSA. They first used conventional evolutionary search Many-Objective Sorting Algorithm (MOSA) [141] and monitored its coverage progress. When the algorithm reached a coverage plateau (state where further mutation of test cases finds it difficult to improve coverage [142], [143]), it would identify functions with low coverage in the module, and use these low-coverage functions as hints to request the LLM (e.g., CodeX [109]) to generate test cases. Afterwards, on the basis of the test cases generated by the LLM, the EA continued exploration. This method can significantly improve the coverage of test samples.

3) Software Project Planning: Story points are a commonly used method in software project planning and cost estimation to gauge the amount of work required to complete a user story [144]. One way to leverage LLMs for this task is through few-shot learning, by providing some labeled examples of past user stories and their estimated points to improve the model's ability to estimate new stories. However, the choice and number of examples can impact the estimation effectiveness. Tawosi *et al.* propose Search-Based Optimisation for Story Point Estimation (SBSE) to optimize the example selection through a multi-objective approach [124]. They use NSGA-II to simultaneously optimize three objective functions - the sum of absolute errors, the confidence interval of the error distribution, and the number of examples. This searches for the Pareto front of user story example sets. The GA explores the trade-off between estimation accuracy and complexity, as measured by the objective functions. The optimized Pareto front provides decision makers with different accuracy-complexity choice options when selecting an example set for few-shot learning with LLMs to estimate story points.

C. Neural Architecture Search (NAS)

LLMs and EAs also contribute to another field known as NAS[4]. Over the years, EAs have found widespread application in NAS [98], [99], revolutionizing the way we design and optimize neural networks. Their ability to simulate natural selection and iteratively improve architectures has proven invaluable in the quest for efficient and high-performing models. However, as we delve into the realm of cutting-edge advancements, a new player has emerged onto the NAS scene - LLMs. With their immense computational capacity and deep understanding of language and context, LLMs bring a fresh perspective to NAS. They possess the potential to offer novel insights and innovative solutions, empowering researchers and engineers to explore uncharted territories of network design. In the NAS applications based on LLM and EA, EAs are commonly employed to establish effective search frameworks, while LLMs leverage their unique abilities to contribute to NAS from diverse angles, including their representation capability, generation capability, and abundant prior knowledge.

1) With Representation Capability of Fine-tuned LLM: Yu *et al.* proposed a NAS method called GPT-NAS [125] that utilizes a GPT model to guide the search of neural architectures. In GPT-NAS, neural architectures are encoded as inputs for the GPT model using a defined encoding strategy. Then the GPT model is pre-trained and fine-tuned on neural architecture datasets to introduce prior knowledge into the search process. In the architecture search, GPT-NAS uses an EA with operations like crossover and mutation to optimize individuals and obtain offspring with better performance. The fine-tuned GPT model is then used to effectively predict excellent new architectures based on previous structural information, thereby reconstructing the sampled architectures to guide and optimize the entire search process. In this method, the EA and GPT achieve complementary advantages. The EA excels in search and optimization, yet its direct application may face limitations due to constraints in variation capability and the vastness of the search space. This can pose challenges in effectively uncovering truly exceptional architectures. By reorganizing the structures on the basis of individuals evolved by GPT, richer prior knowledge can be introduced to optimize the network architectures, effectively reducing the actual search space and guiding the search towards high-performance areas.

2) With Code Generation Capability of General LLM: Nasir *et al.* leveraged the code generation ability of large models to help NAS [126]. The proposed LLMatic does not directly search in the structure representation space, but uses the code of neural network structures as search points. Specifically, LLMatic establishes two archives: a network archive that stores trained network models, and a prompt archive that stores prompts and temperature parameters for generating networks. LLMatic employs QD algorithms [145] to perform the search, utilizing the repeated process of generation, training and selection to continuously optimize the quality and diversity of networks. The large model CodeGen [146] is used to complete crossover and mutation in the evolutionary process: in the mutation operation, CodeGen makes subtle modifications to the selected network based on the mutation prompt. In the crossover operation, CodeGen partially matches the selected multiple networks based on the crossover prompt. Chen *et al.* took a similar approach by using the PALM [4] as the crossover and mutation operators for the NAS task in the evolutionary search process. Moreover, they further improved the LLM's ability of generating candidate architectures through soft prompt tuning during NAS [127]. The Guided Evolution framework proposed by Morris *et al.* also uses LLM to mutate and crossover the code [128]. The main EAs adopted are SPEA-2 [147] and NSGA-II. SPEA-2 is used to retain elite individuals that performed well in the previous generation. NSGA-II is used to select individuals for crossover and mutation. At the same time, they proposed "Evolution of Thought" to continuously optimize the recommendations of LLM, which allows the LLM to propose improved modifications for new individuals based on the performance of past mutations, making the evolutionary process more data-driven and efficient. In addition to the above study, some papers [129], [130] do not explicitly mention using an EA framework, but employ LLM as an operator to generate new architectures. In addition to using LLM for architecture code generation, GPTN-SS employed GPT-4 to assist in generating code for a tensor network structure search algorithm [131]. Similar to other code generation methods discussed in Section IV-A, EA provides an iterative optimization framework, while LLM is utilized for evolutionary operations on the code.

3) With Reasoning Capability of LLM Based on Its Abundant Prior Knowledge: Unlike previous works that used LLM in the NAS search process, Jawahar *et al.* utilized LLM to build performance predictors [132]. Specifically, they designed predictor prompts (including task description, architecture parameter definitions, examples, etc.) to define the prediction task. The prompts and test architectures would be input into the LLM (such as GPT-4), and the LLM could make predictions based on the architecture knowledge it learned during pre-training by understanding the complex relationships between architectures and performances. Since LLMs learned architecture-related knowledge from a large amount of literature, and this type of predictor is designed simply without needing to train an over-parameterized network, the deployment cost is low. Chen *et al.* designed a novel reasoning meta-structure search framework called ReStruct to automatically discover meta-structures in heterogeneous information networks [133]. They developed a grammar translator that encodes meta-structures into natural language sentences, which facilitates LLMs' understanding of structural patterns. Based on this, they designed a predictor and selector based on LLMs. The predictor adopts the idea of few-shot learning - it randomly samples some instances from the historical structure-performance records stored in previous searches, and uses these instances to prompt the LLM to predict the performance of new generated candidate structures. The selector will receive the prediction results of various candidate structures

[4]Methods reviewed in this section differ from those presented in Section III-B. Approaches discussed in Section III-B primarily focus on LLM architecture search, and their techniques are based on EAs, whereas methods reviewed in this section leverage the synergistic combination of EAs and LLMs. Moreover, these NAS methods are more versatile and not limited to LLM architecture search alone, applicable to a broader range of NAS tasks.

given by the predictor, and interact with the LLM to obtain selection suggestions. These modules are used in the selection process of GA, and combined with the designed insertion, grafting, and deletion operations to continuously optimize the meta-structure population and drive the search process. ReStruct also includes an explanation module based on chain-of-thought prompting, which can ultimately obtain semantically explainable and high-performance meta-structures.

D. Other Generative Tasks

In addition to the three widely applied scenarios mentioned above, the collaboration between LLM and EA has also driven the performance improvement of more generative tasks. This subsection provides a brief introduction to these studies. Most of these studies combined the generative ability of LLM and the search ability of EA, generating better results based on rich prior knowledge and text understanding.

1) Text Generation: Text generation is one of the most direct applications of LLMs. Xiao *et al.* used the generation ability of LLM and the search ability of EA for news summary generation [27], where event patterns structure is employed to represent and describe the key information, relationships and characteristics of events. They used LLM to extract event patterns from text, and then used GA to evolve the event pattern pool according to the importance and information quantity of argument roles, selecting the event pattern with the highest fitness to input into LLM for news summary generation. Some studies draw inspiration from text generation to optimize non-textual domains. Lim *et al.* developed the SCAPE system to explore conceptual architecture design by combining EA and LLM [148]. SCAPE represents design schemes or concepts using text, which allows EA to endow LLM with stronger innovative capabilities. The system enables architects to participate in parent selection, significantly improving the quality and exploration efficiency compared to vanilla generative AI tools. Another interesting application is to generate Super Mario game levels using LLMs [23], [149]. Sudhakaran *et al.* used a string representation method similar to the Video Game Level Corpus dataset [150] to encode levels, and then trained a LLM namely MarioGPT based on GPT-2 that can generate Mario levels according to natural language descriptions. By combining MarioGPT with the Novelty Search algorithm [151], an open-ended level generation process was formed. Novelty search can continually discover levels with diverse structures and gameplay styles.

2) Text-to-image Generation: In this type of task, when using LLMs like Stable Diffusion [152] to generate photo-realistic images, it is necessary to optimize the input text prompts and model parameters in order to achieve better performance. Berger *et al.* tried to solve this problem and proposed the StableYolo method [153]. Its main innovation lies in combining EAs with computer vision for the first time, simultaneously optimizing language prompts and model hyperparameters to improve image generation quality. Among them, the LLM Stable Diffusion plays the main role of image generation, generating a series of images based on the input text prompts. While the GA plays the role of searching and optimizing. It uses the recognition confidence given by the YOLO object detection model as the fitness function to perform multi-objective optimization. The optimization objects include keywords in text prompts and various hyperparameters of the Stable Diffusion model. Through repeated iteration, the best language prompts and model settings that maximize image quality are found.

In addition to text-related tasks, some applications in natural sciences and social sciences are provided as follows to demonstrate the combined advantages of LLMs and EAs.

3) Natural Science: The researchers from McGill University conducted experiments using GPT-3.5 to fragment and recombine molecules [154] represented as a simplified molecular-input line-entry system (SMILES) string [155]. The LLM was able to successfully fragment molecules at rotatable bonds with a 70% success rate. When recombining molecules, the LLM generated new combined molecules with fragments from each parent molecule that were chemically reasonable more often than a random recombination operation. The researchers also explored using the LLM to optimize molecular properties based on a similarity score to vitamin C. By providing the LLM with a set of molecules and their performance scores, the LLM generated new molecules that it believed would improve the score. The generated modifications were found to be more chemically sound compared to traditional GAs. Teukam *et al.* utilized LLM and GA to optimize enzyme sequences, thereby improving their catalytic activity and stability [156]. Specifically, a protein language model based on evolutionary scale modeling (ESM-2) was employed to generate mutations and predict their functions. GAs then optimized the sequences through iterative selection and crossover operations, enhancing the predicted functionality. Ultimately, this approach yielded enzymes with optimized sequences that demonstrated enhanced catalytic performance compared to their wild-type counterparts.

4) Social Science: Suzuki *et al.* applied LLMs and EAs to study the evolutionary dynamics of social populations under conflicts of interest and behaviors between individuals [157], i.e. the propagation and changes of different personality traits and behavioral strategies. They used LLMs to make trait expressions more complex and higher-order, able to directly describe personality and psychological attributes that are difficult to model directly, and map them to behaviors. They then constructed an agent-based model where each agent uses a short natural language description to represent a personality trait related to cooperation as its gene. The LLM was used to derive a behavioral strategy for each agent based on the personality trait description and game history. The population evolved according to a GA, with offspring inheriting mutated versions of their parents' trait genes, thus simulating the evolutionary dynamics of population changes over generations. This study provided a novel approach for researching the evolution of complex social populations.

In summary, the collaboration between LLMs and EAs has driven advancements in distinct generative task domains. By exploiting their strengths, researchers have made significant progress in these domains, paving the way for further advancements in generative modeling and related fields.

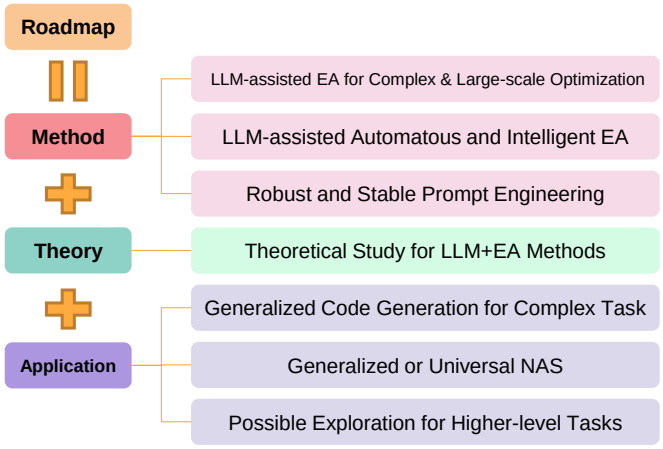


Fig. 3. Roadmap and Future Directions.

V. Roadmap and Future Directions

In the previous sections, we have reviewed the recent advances in unifying LLMs and EAs, but there are still many challenges and open problems that need to be addressed. In this section, we discuss the future directions of this research area. As depicted in Fig. 3, our perspective for future research includes investigations into methods, theory, and applications. Some of these studies aim to build upon the strengths and weaknesses highlighted in the reviewed research to propose more advanced approaches. Meanwhile, other research endeavors seek to explore new frontiers that go beyond the current body of knowledge, aiming to forge new research questions and infuse the field with renewed energy. This section will provide a separate introduction to these future research directions.

A. LLM-assisted EA for Complex & Large-scale Optimization

Existing research has partially validated the ability of LLMs to address small-scale numerical optimization problems [24], [31], [32], but practicality still poses challenges. Firstly, for complex optimization problems with high-dimensional search spaces, numerous constraints, and high-precision numerical optimization, the limited context understanding and prompt length restrictions of large models may increase the difficulty of interaction with LLMs. Secondly, as black-box optimizers, the decision-making process of LLMs is difficult to interpret, and it remains unknown whether their optimization ability stems from reasoning or randomness. Moreover, the scope of the evaluated problems in existing research is relatively narrow [24], [31], [32], often limited to specific optimization problems, and the assessments only consider limited influencing factors, which are insufficient to fully demonstrate the optimization capabilities of LLMs. Another issue is that empirical studies suggest that LLMs may struggle to handle constrained problems effectively, making it crucial to incorporate constraint conditions into LLM optimization frameworks [30].

Therefore, to further explore and utilize the capabilities of LLMs in solving optimization problems, future work can focus on addressing the aforementioned challenges and improving various aspects such as LLM behavior interpretation and evaluation, rational utilization of optimization information, and solving complex problems. Currently, efforts can be made to interpret and understand the behavior of LLMs, such as analyzing their internal attention mechanisms [158] or investigating the relationship between LLM temperature settings and exploration of optimal solutions, in order to gain a more objective understanding of LLM optimization abilities. Based on these foundations, improvement strategies or new pre-training models can be proposed to solve optimization problems with higher dimensions and complex constraint conditions. Furthermore, when dealing with domain-specific optimization problems, the corresponding domain knowledge graphs can be embedded into the reasoning of LLMs to further unleash the advantages of LLMs in text understanding and reasoning [159]. Meanwhile, by implementing complex evolutionary strategies through mixture-of-expert models [160], the collaboration between EAs and LLMs can be further enhanced, where different submodels focus on different optimization operations and choice of the operators can be determined based on the current evolutionary state. Beyond these research directions, the contextual understanding and processing abilities of LLMs can be further exploited, for example, by analyzing historical information of the optimization process and considering more advanced prompt methods. Conducting validation on large-scale, real-world problems going forward would also provide meaningful value.

B. LLM-assisted Automatous and Intelligent EA

LLM holds significant promise for broader application scenarios within evolutionary optimization. In the following, we introduce several prospective areas, which, despite being relatively underexplored at present, are expected to drive more automatous and intelligent EAs in black-box optimization across diverse domains.

First, multi-modal LLMs provide opportunities for cross-domain EAs. Some LLMs have learned the relationships between different modalities during pre-training, allowing them to be used for downstream tasks involving multiple modalities, such as image captioning and text-to-image generation. The ability of LLMs to understand correspondences across modalities can help EAs achieve cross-modal crossover and mutation by providing cross-modal prompts or examples. Additionally, population construction and evaluation can be improved across modalities. Furthermore, as LLMs continue to progress, evolutionary operators based on LLMs will also benefit and improve the performance of EAs. With increasing model scales and stronger generalization capabilities, evolutionary operators based on LLMs will be better able to simulate more complex evolutionary mechanisms precisely. This provides potential for tackling problems in complex search spaces. The emerging and rapidly advancing field of LLMs will also provide more choices for EAs. LLMs with different pre-training characteristics will continue to enrich and enhance the capabilities of evolutionary operators based on LLMs.

In addition, LLM is trained on a large corpus of textual data and encompasses knowledge from various domains. It can

serve as a knowledge base to assist evolutionary computation in better integrating domain knowledge, thereby improving optimization efficiency or optimality. For example, LLM's domain knowledge can be utilized to provide good initial solutions or improve problem formulation, including solution encoding, definition of solution space, and more. LLM can also provide valuable design principles for algorithm design based on its knowledge, enabling EC to tackle complex problems such as multi-objective, discrete, and dynamic problems through knowledge transfer.

C. Robust and Stable Prompt Engineering

One of the primary approaches to improving LLM through EAs is prompt engineering. Currently, a common method involves using LLM as an evolutionary operator to continuously generate new prompts within an evolutionary framework [17], [66], [67]. This approach has been proven effective and superior in numerous research studies. However, several challenges still remain. Firstly, the initial population of the evolution process significantly influences the results. This is because a general and adaptable prompt template plays a crucial role in prompt generation and efficacy [127]. Random initialization may struggle to leverage prior knowledge, while manual initialization can introduce bias. Additionally, for problems with abundant prior information, the prompt search space can be extensive. As the prompt length and vocabulary space increase, the search space exponentially grows, which can lead to overfitting or getting stuck in local optima. Lastly, these methods lack stability and heavily depend on the capabilities of LLM [67], making them susceptible to randomness. If LLM fails to comprehend and effectively utilize prompts, the effectiveness of the methods may be limited.

Regarding the issue of initialization, future research might consider multi-source initialization, simultaneously utilizing LLM to automatically expand the size and quality of the initial population. For problems with a high search space complexity, it is necessary to design more efficient evolutionary strategies, such as introducing richer evolutionary operators, leveraging the strengths of different EAs, and employing adaptive evolution. Another promising direction worth exploring is the utilization of human feedback to expedite and optimize the prompt discovery process, which has been overlooked in current research. In addressing the constraint of LLM on prompt performance, apart from considering the use of more powerful LLM models or designing dedicated fine-tuning methods, an alternative research direction at a lower level is to extract internal representations of LLM and study their mechanisms for understanding prompts, thereby guiding the design and representation of prompt strategies more effectively. Stable prompt optimization is also a promising direction to mitigate performance fluctuations caused by randomness. Bidirectional training is an optional strategy [161], where the forward model generates outputs (such as text) from the prompts, while the inverse model generates prompts from the outputs, thereby achieving stable prompting.

D. Theoretical Study for Specific LLM+EA Methods

Although empirical studies have validated the effectiveness of combining LLM and EA on small-scale problems, the incentives of their interaction are not yet clear. This reminds us to explore the sources of mutual promotion between LLM and EA in theoretical research, and to analyze in detail their complementary advantages and existing issues in large-scale empirical studies, thereby further promoting further improvement. This research can delve into two aspects:

1) Algorithm Analysis: Researchers can analyze specific algorithms that combine LLM and EA and explore their convergence, complexity, and other properties. For convergence analysis, researchers can verify whether the algorithm can converge to the optimal solution or local optimum during the iterative process. This may involve proving the optimization properties of the objective function or fitness function during the algorithm's iterative process and analyzing the impact of algorithm parameter settings on convergence. Regarding complexity analysis, future work can analyze the time and space complexity of algorithms that combine LLM and EA. This includes evaluating the complexity of key steps in the algorithm, such as generating new individuals, selection, crossover, and mutation operations. The applicability of methods that combine LLM and EA on different types of problems is another perspective could be studied, which may involve analyzing the relationship between problem attributes, constraints, problem size, and algorithm performance, as well as studying the performance guarantees or theoretical limits of algorithms on specific problem types.

2) Optimization Theory: In problem modeling and analysis, specific problems that combine LLM and EA can be modeled and analyzed from the perspective of optimization theory. This may involve defining and characterizing the objective function, constraints, and exploring the feasible solution space of the problem. The key point of research lies in the search strategy, where researchers can study how to select and design appropriate search strategies to improve the performance of LLM and EA combination. This may include comparing and analyzing the effects of different search strategies (such as selection, crossover, and mutation operations) and sensitivity analysis of search strategy parameters. Moreover, future work can analyze the complexity of these methods on specific problems, including analyzing the computational complexity category of problems (such as P, NP, NP-hard) and analyzing the approximate performance of algorithms to determine the theoretical limits of algorithms in solving specific problems.

E. Generalized Code Generation for Complex Task

LLM and EA jointly contribute to another field of advancement, i.e., code generation. Currently, a plethora of research has emerged in this domain, which has further spurred the development of various downstream tasks, such as software engineering and EA design.

In code generation methods, one approach involves utilizing LLM to generate a substantial amount of training data and subsequently employing reinforcement learning to fine-tune LLM [28]. However, a significant challenge with such methods

lies in the diversity and scale of the training data. LLM's training process heavily relies on extensive data support, and the LLM+EA generation approach may not cover all possible use cases. Moreover, the code generated by LLM might converge to a single solution, thereby limiting the model's capacity for generalization [28]. To address these limitations and generate more diverse training data that can adapt to a broader range of domains, one potential improvement is to incorporate continuous learning mutation operators. These operators can track changes in the problem space and introduce multiple initial solutions, facilitating multiple restarts of the search process. Additionally, by recording the mutations produced by each mutation operator and providing rewards or penalties based on the performance of the mutated offspring, the operators can also learn higher-quality mutation patterns.

Another approach type focuses on leveraging LLM's code generation capabilities and EA's search framework to continuously enhance code generation [21], [52]–[54]. However, these methods encounter difficulties when dealing with complex algorithmic logic [53]. For tasks that involve intricate logic, a single code snippet may prove inadequate, necessitating the collaboration of multiple code snippets. Unfortunately, LLM's performance in automatically generating such code collections is suboptimal, and the existing limitations on LLM's input-output length make it challenging to handle large-scale and complex logic code [53]. To mitigate these challenges, one possible approach is to modularize complex algorithmic logic by designing a universal modular design and generation method that can decompose complex tasks. Alternatively, a more user-friendly and straightforward approach could involve developing an interactive interface that allows users to determine how tasks should be decomposed. Subsequently, LLM and EA can generate code for each subtask accordingly.

F. Generalized or Universal NAS

In the realm propelled by the collaboration of LLM and EA, NAS stands out as a vital application scenario. The prior knowledge about network architecture and code generation capability of LLM can greatly assist EA in efficiently discovering optimal architectures. However, existing work still faces various challenges. Some of these challenges are common among NAS methods based on EAs, such as high time consumption. Additionally, the integration of LLM introduces new issues. Firstly, it is important to note that the current state-of-the-art LLM models, while excelling in various tasks, are not specifically tailored for NAS. Their ability to tackle NAS tasks stems from the presence of network architecture-related information in the training data [125]. Consequently, different LLM models exhibit significant variations in performance in NAS tasks. Furthermore, when compared to mainstream NAS methods, LLM-based approaches still exhibit some gaps in terms of their application scope and generalization ability. Additionally, certain ablation experiments reveal that the direct utilization of LLM prompts yields suboptimal learning results [127], underscoring the limitations of the current capabilities of LLM. Exploring fine-tuning methods that combine LLM with EAs has proven necessary.

Addressing these aforementioned issues would enhance the joint performance of LLM and EA in NAS tasks, unlocking the untapped potential of LLM in NAS and accelerating EA's search speed. Firstly, it is crucial to evaluate the performance of different LLM models in NAS tasks. Conducting fair evaluation experiments would validate the application scope and generalization ability of these LLM models across diverse network architectures. Leveraging additional training data to augment LLM's NAS capabilities and mitigating the impact of LLM pretraining quality on search results would improve robustness and stability. Moreover, optimizing the deeper structure of LLM during the fine-tuning process holds promise in generating superior solutions. To address efficiency concerns, subsequent research should explore leveraging historical search knowledge to expedite future searches and provide well-defined search spaces for LLM.

G. Applications and Innovations in Higher-level Tasks

In the discussed techniques within this paper, we observe their role as submodules in higher-level tasks. For instance, prompt optimization plays a role in LLM-based tasks, while code generation serves a purpose in NAS or software engineering tasks. In fact, these techniques can be utility in a broader range of higher-level tasks. Taking software engineering as an example, in higher-level tasks (such as repository-level software engineering), LLM and EA can fulfill more applicable demands. LLM can be used to implement cross-project collaboration. LLM not only can analyze the source codes and histories of multiple related projects to provide a large-scale search space for EA, but also its inherent prior knowledge can be used to identify the correspondence between knowledge of different projects, where EA can then utilize this relationship to perform knowledge transfer. In addition, EA can automatically generate version control strategies for different development stages, such as how to merge branches and when to release versions, balancing development efficiency and stability. Or EA can optimize multi-objective quality prediction models for different stages of the software lifecycle. Another example unmentioned before is the intelligent agent [162], [163], which can learn and evolve its behavioral strategies through interaction with the environment. In [164], the inference and reflection capabilities of the intelligent agent are realized through interaction with LLM. Among them, policy optimization is achieved through prompt optimization, following a simplified evolutionary process similar to "natural selection". Future research can explore more complex evolutionary mechanisms to aid in the self-evolution of intelligent agents. Moreover, by exploring the co-evolution of multiple agents, this approach is promising to be deployed in more complex and large-scale tasks [162].

VI. Conclusion

In this paper, we have undertaken a comprehensive exploration of the intersection between Evolutionary Algorithms (EAs) and Large Language Models (LLMs) in the transformative era of AI. We introduced three research paradigms: LLM-enhanced EA, EA-enhanced LLM, and applications driven by

the integrated synergy of LLM and EA. These paradigms exemplify the amalgamation of LLMs and EAs in various application scenarios, showcasing their collaborative strengths in tasks such as Neural Architecture Search (NAS), code generation, software engineering, and text generation. As we look forward, the collaboration between EA and LLM has garnered increasing attention, with research focusing on enhancing mutual performance and driving task-specific improvements. Currently, existing LLMs lack the direct capability to handle complex and large-scale optimization problems, but they have shown promise in generating optimization algorithms and evaluating optimization results. Future work should focus on leveraging LLMs to solve more complex optimization problems by exploring approaches such as interpreting LLM behavior and comprehensively evaluating performance on diverse issues. The exploration of enhancing LLMs with EAs is still in its early and fragmented stages. Areas beyond prompt engineering and NAS deserve further in-depth and systematic investigation. These endeavors are crucial in advancing Evolutionary Computing to effectively address the forefront challenges in the AI community. Theoretical analysis of algorithms combining LLMs and EAs from an optimization theory perspective is essential to understand properties like convergence and complexity. In practical applications, mature collaborative paradigms have emerged where LLMs and EAs work together. These paradigms leverage the iterative search framework provided by EAs and the intelligent evolutionary operators offered by LLMs to tackle various text-based optimization problems, such as code generation and prompt engineering. Generalized code generation for intricate tasks may involve modular and interactive methods. Broadening the application scope and generalization ability of LLMs in NAS through techniques such as pretraining specialized models also merits investigation. While improvements within this established paradigm or extensions to new application domains are valuable, it is of paramount importance to transcend this paradigm and explore novel collaborative mechanisms. Overall, the collaboration between EA and LLM holds significant potential for future advancements. By further refining the interaction between these fields, we can pave the way for more efficient and effective optimization strategies and drive innovation in the broader field of artificial intelligence.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[2] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[3] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[5] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *Transactions on Machine Learning Research*, 2022.

[6] L. Wu, Z. Zheng, Z. Qiu, H. Wang, H. Gu, T. Shen, C. Qin, C. Zhu, H. Zhu, Q. Liu *et al.*, "A survey on large language models for recommendation," *arXiv preprint arXiv:2305.19860*, 2023.

[7] J. Chen, Z. Liu, X. Huang, C. Wu, Q. Liu, G. Jiang, Y. Pu, Y. Lei, X. Chen, X. Wang *et al.*, "When large language models meet personalization: Perspectives of challenges and opportunities," *arXiv preprint arXiv:2307.16376*, 2023.

[8] Y. Zhou, X. Wu, B. Huang, J. Wu, L. Feng, and K. C. Tan, "Causalbench: A comprehensive benchmark for causal learning capability of large language models," *arXiv preprint arXiv:2404.06349*, 2024.

[9] R. K. Luu and M. J. Buehler, "Bioinspiredllm: Conversational large language model for the mechanics of biological and bio-inspired materials," *Advanced Science*, vol. 11, no. 10, p. 2306724, 2024.

[10] M. J. Buehler, "Mechgpt, a language-based strategy for mechanics and materials modeling that connects knowledge across scales, disciplines, and modalities," *Applied Mechanics Reviews*, vol. 76, no. 2, p. 021001, 2024.

[11] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–40, 2023.

[12] Y. Tian, H. Chen, X. Xiang, H. Jiang, and X. Zhang, "A comparative study on evolutionary algorithms and mathematical programming methods for continuous optimization," in *Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2022, pp. 1–8.

[13] Z.-H. Zhan, L. Shi, K. C. Tan, and J. Zhang, "A survey on evolutionary computation for complex continuous optimization," *Artificial Intelligence Review*, pp. 1–52, 2022.

[14] P. P. Ray, "Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet of Things and Cyber-Physical Systems*, 2023.

[15] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[16] P. Pošík, W. Huyer, and L. Pál, "A comparison of global search algorithms for continuous black box optimization," *Evolutionary Computation*, vol. 20, no. 4, pp. 509–541, 2012.

[17] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers," in *Proceedings of the 12th International Conference on Learning Representations*, 2024.

[18] J. Gao, H. Xu, H. Shi, X. Ren, L. Philip, X. Liang, X. Jiang, and Z. Li, "Autobert-zero: Evolving bert backbone from scratch," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, 2022, pp. 10 663–10 671.

[19] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *arXiv preprint arXiv:2304.13712*, 2023.

[20] Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, "Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 423–435.

[21] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi *et al.*, "Mathematical discoveries from program search with large language models," *Nature*, pp. 1–3, 2023.

[22] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward, "Genetic improvement of software: a comprehensive survey," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 415–432, 2017.

[23] S. Shyam, G.-D. Miguel, F. Matthias, G. Claire, N. Elias, and R. Sebastian, "Mariogpt: Open-ended text2level generation through large language models," *arXiv preprint arXiv:2302.05981*, 2023.

[24] S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong, "Large language models as evolutionary optimizers," in *Proceedings of the IEEE 2023 Congress on Evolutionary Computation*. IEEE, 2024, pp. 1–6.

[25] Y. G. Woldesenbet and G. G. Yen, "Dynamic evolutionary algorithm with variable relocation," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 500–513, 2009.

[26] L. Araujo, "How evolutionary algorithms are applied to statistical natural language processing," *Artificial Intelligence Review*, vol. 28, pp. 275–303, 2007.

[27] L. Xiao, X. Chen, and X. Shan, "Enhancing large language models with evolutionary fine-tuning for news summary generation," *Journal of Intelligent and Fuzzy Systems*, no. Preprint, pp. 1–13, 2024.

[28] J. Lehman, J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley, "Evolution through large models," in *Handbook of Evolutionary Machine Learning*. Springer, 2023, pp. 331–366.

[29] P.-F. Guo, Y.-H. Chen, Y.-D. Tsai, and S.-D. Lin, "Towards optimizing with large language models," *arXiv preprint arXiv:2310.05204*, 2023.

[30] B. Huang, X. Wu, Y. Zhou, J. Wu, L. Feng, R. Cheng, and K. C. Tan, "Exploring the true potential: Evaluating the black-box optimization capability of large language models," *arXiv preprint arXiv:2404.06290*, 2024.

[31] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," in *Proceedings of the 12th International Conference on Learning Representations*, 2024, pp. 1–10.

[32] E. Meyerson, M. J. Nelson, H. Bradley, A. Moradi, A. K. Hoover, and J. Lehman, "Language model crossover: Variation through few-shot prompting," *arXiv preprint arXiv:2302.12170*, 2023.

[33] R. T. Lange, Y. Tian, and Y. Tang, "Large language models as evolution strategies," *arXiv preprint arXiv:2402.18381*, 2024.

[34] Y. Huang, W. Zhang, L. Feng, X. Wu, and K. C. Tan, "How multimodal integration boost the performance of llm for optimization: Case study on capacitated vehicle routing problems," *arXiv preprint arXiv:2403.01757*, 2024.

[35] S. Brahmachary, S. M. Joshi, A. Panda, K. Koneripalli, A. K. Sagotra, H. Patel, A. Sharma, A. D. Jagtap, and K. Kalyanaraman, "Large language model-based evolutionary optimizer: Reasoning with elitism," *arXiv preprint arXiv:2403.02054*, 2024.

[36] L. Zhang, "Cuda-accelerated soft robot neural evolution with large language model supervision," *arXiv preprint arXiv:2405.00698*, 2024.

[37] M. Chiquier, U. Mall, and C. Vondrick, "Evolving interpretable visual classifiers with large language models," *arXiv preprint arXiv:2404.09941*, 2024.

[38] F. Liu, X. Lin, Z. Wang, S. Yao, X. Tong, M. Yuan, and Q. Zhang, "Large language model for multi-objective evolutionary optimization," *arXiv preprint arXiv:2310.12541*, 2023.

[39] H. Bradley, A. Dai, H. B. Teufel, J. Zhang, K. Oostermeijer, M. Bellagente, J. Clune, K. Stanley, G. Schott, and J. Lehman, "Quality-diversity through ai feedback," in *Proceedings of the 2nd Agent Learning in Open-Endedness Workshop, in 37th Annual Conference on Neural Information Processing Systems*, 2023.

[40] B. Lim, M. Flageat, and A. Cully, "Large language models as in-context ai generators for quality-diversity," *arXiv preprint arXiv:2404.15794*, 2024.

[41] Z. Wang, S. Liu, J. Chen, and K. C. Tan, "Large language model-aided evolutionary search for constrained multiobjective optimization," *arXiv preprint arXiv:2405.05767*, 2024.

[42] M. Pluhacek, A. Kazikova, T. Kadavy, A. Viktorin, and R. Senkerik, "Leveraging large language models for the generation of novel metaheuristic optimization algorithms," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 1812–1820.

[43] A. AhmadiTeshnizi, W. Gao, and M. Udell, "Optimus: Optimization modeling using mip solvers and large language models," *arXiv preprint arXiv:2310.06116*, 2023.

[44] R. Zhong, Y. Xu, C. Zhang, and J. Yu, "Leveraging large language model to generate a novel metaheuristic algorithm with crispe framework," *arXiv preprint arXiv:2403.16417*, 2024.

[45] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang, "Evolution of heuristics: Towards efficient automatic algorithm design using large language model," in *Proceedings of the 41st International Conference on Machine Learning*, 2024, pp. 1–9.

[46] H. Ye, J. Wang, Z. Cao, and G. Song, "Reevo: Large language models as hyper-heuristics with reflective evolution," *arXiv preprint arXiv:2402.01145*, 2024.

[47] E. Hemberg, S. Moskal, and U.-M. O'Reilly, "Evolving code with a large language model," *arXiv preprint arXiv:2401.07102*, 2024.

[48] R. T. Lange, Y. Tian, and Y. Tang, "Evolution transformer: In-context evolutionary optimization," *arXiv preprint arXiv:2403.02985*, 2024.

[49] Y. Yao, F. Liu, J. Cheng, and Q. Zhang, "Evolve cost-aware acquisition functions using large language models," *arXiv preprint arXiv:2404.16906*, 2024.

[50] O. Kramer, "Large language models for tuning evolution strategies," *arXiv preprint arXiv:2405.10999*, 2024.

[51] Y. Huang, S. Wu, W. Zhang, J. Wu, L. Feng, and K. C. Tan, "Autonomous multi-objective optimization using large language model," *arXiv preprint arXiv:2406.08987*, 2024.

[52] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," in *Proceedings of the 12th International Conference on Learning Representations*, 2024.

[53] Z. Chen, L. Cao, S. Madden, T. Kraska, Z. Shang, J. Fan, N. Tang, Z. Gu, C. Liu, and M. Cafarella, "Seed: Domain-specific data curation with large language models," *arXiv preprint arXiv:2310.00749*, 2023.

[54] Y. Fu, Y. Zhang, Z. Yu, S. Li, Z. Ye, C. Li, C. Wan, and Y. C. Lin, "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models," in *Proceedings of the 2023 IEEE/ACM International Conference on Computer Aided Design*. IEEE, 2023, pp. 1–9.

[55] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[56] H. Chen, G. E. Constante-Flores, and C. Li, "Diagnosing infeasible optimization problems using large language models," *arXiv preprint arXiv:2308.12923*, 2023.

[57] J. W. Chinneck and E. W. Dravnieks, "Locating minimal infeasible constraint sets in linear programs," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 157–168, 1991.

[58] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in python," *Mathematical Programming Computation*, vol. 3, pp. 219–260, 2011.

[59] P. Maddigan, A. Lensen, and B. Xue, "Explaining genetic programming trees using large language models," *arXiv preprint arXiv:2403.03397*, 2024.

[60] G. Singh and K. K. Bali, "Enhancing decision-making in optimization through llm-assisted inference: A neural networks perspective," in *Proceedings of the 2024 International Joint Conference on Neural Networks*, 2024, pp. 1–7.

[61] S. Diao, Z. Huang, R. Xu, X. Li, L. Yong, X. Zhou, and T. Zhang, "Black-box prompt learning for pre-trained language models," *Transactions on Machine Learning Research*, 2023.

[62] P. Liang, R. Bommasani, T. Lee, and D. Tsipras, "Holistic evaluation of language models," *Transactions on Machine Learning Research*, 2023, featured Certification, Expert Certification. [Online]. Available: https://openreview.net/forum?id=iO4LZibEqW

[63] H. Xu, Y. Chen, Y. Du, N. Shao, W. Yanggang, H. Li, and Z. Yang, "Gps: Genetic prompt search for efficient few-shot learning," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 8162–8171.

[64] A. Prasad, P. Hase, X. Zhou, and M. Bansal, "Grips: Gradient-free, edit-based instruction search for prompting large language models," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 2023, pp. 3827–3846.

[65] R. Pan, S. Xing, S. Diao, X. Liu, K. Shum, J. Zhang, and T. Zhang, "Plum: Prompt learning using metaheuristic," *arXiv preprint arXiv:2311.08364*, 2023.

[66] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel, "Promptbreeder: Self-referential self-improvement via prompt evolution," *arXiv preprint arXiv:2309.16797*, 2023.

[67] Y. B. Li and K. Wu, "Spell: Semantic prompt evolution based on a llm," *arXiv preprint arXiv:2310.01260*, 2023.

[68] F. Jin, Y. Liu, and Y. Tan, "Zero-shot chain-of-thought reasoning guided by evolutionary algorithms in large language models," *arXiv preprint arXiv:2402.05376*, 2024.

[69] C. Singh, J. X. Morris, J. Aneja, A. M. Rush, and J. Gao, "iprompt: Explaining data patterns in natural language via interpretable autoprompting," *ArXiv preprint*, vol. 2210, 2022.

[70] W. Cui, J. Zhang, Z. Li, H. Sun, D. Lopez, K. Das, B. Malin, and S. Kumar, "Phaseevo: Towards unified in-context prompt optimization for large language models," *arXiv preprint arXiv:2402.11347*, 2024.

[71] H. Yang and K. Li, "Instoptima: Evolutionary multi-objective instruction optimization via large language model-based instruction operators," *arXiv preprint arXiv:2310.17630*, 2023.

[72] J. Baumann and O. Kramer, "Evolutionary multi-objective optimization of large language model prompts for balancing sentiments," in *Proceedings of the International Conference on the Applications of Evolutionary Computation*. Springer, 2024, pp. 212–224.

[73] T. Sun, Y. Shao, H. Qian, X. Huang, and X. Qiu, "Black-box tuning for language-model-as-a-service," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2022, pp. 20 841–20 855.

[74] T. Sun, Z. He, H. Qian, Y. Zhou, X.-J. Huang, and X. Qiu, "Bbtv2: towards a gradient-free future with large language models," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 3916–3930.

[75] Y. Chai, S. Wang, Y. Sun, H. Tian, H. Wu, and H. Wang, "Clip-tuning: Towards derivative-free prompt learning with a mixture of rewards," in *Findings of the Association for Computational Linguistics: EMNLP 2022*, 2022, pp. 108–117.

[76] M. Shen, S. S. Ghosh, P. Sattigeri, S. Das, Y. Bu, and G. Wornell, "Reliable gradient-free and likelihood-free prompt tuning," in *Findings of the Association for Computational Linguistics*, 2023, pp. 2416–2429.

[77] L. Yu, Q. Chen, J. Lin, and L. He, "Black-box prompt tuning for vision-language model as a service," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 2023, pp. 1686–1694.

[78] Z. Fei, M. Fan, and J. Huang, "Gradient-free textual inversion," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 1364–1373.

[79] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang, "Wizardlm: Empowering large language models to follow complex instructions," in *Proceedings of the 12th International Conference on Learning Representations*, 2024.

[80] J. Sun, C. Mei, L. Wei, K. Zheng, N. Liu, M. Cui, and T. Li, "Dial-insight: Fine-tuning large language models with high-quality domain-specific data preventing capability collapse," *arXiv preprint arXiv:2403.09167*, 2024.

[81] X. Liu, N. Xu, M. Chen, and C. Xiao, "Autodan: Generating stealthy jailbreak prompts on aligned large language models," *arXiv preprint arXiv:2310.04451*, 2023.

[82] R. Lapid, R. Langberg, and M. Sipper, "Open sesame! universal black box jailbreaking of large language models," *arXiv preprint arXiv:2309.01446*, 2023.

[83] X. Zou, Y. Chen, and K. Li, "Is the system message really important to jailbreaks in large language models?" *arXiv preprint arXiv:2402.14857*, 2024.

[84] Z. Shi, Y. Wang, F. Yin, X. Chen, K.-W. Chang, and C.-J. Hsieh, "Red teaming language model detectors with language models," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 174–189, 2024.

[85] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992.

[86] R. Storn and K. Price, "Minimizing the real functions of the icec'96 contest by differential evolution," in *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, 1996, pp. 842–844.

[87] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[88] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[89] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[90] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[91] D. Ganguli, L. Lovitt, J. Kernion, A. Askell, Y. Bai, S. Kadavath, B. Mann, E. Perez, N. Schiefer, K. Ndousse *et al.*, "Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned," *arXiv preprint arXiv:2209.07858*, 2022.

[92] N. Jain, A. Schwarzschild, Y. Wen, G. Somepalli, J. Kirchenbauer, P.-y. Chiang, M. Goldblum, A. Saha, J. Geiping, and T. Goldstein, "Baseline defenses for adversarial attacks against aligned language models," *arXiv preprint arXiv:2309.00614*, 2023.

[93] V. Ganesan, G. Ramesh, and P. Kumar, "Supershaper: Task-agnostic super pre-training of bert models with variable hidden dimensions," *arXiv preprint arXiv:2110.04711*, 2021.

[94] Y. Yin, C. Chen, L. Shang, X. Jiang, X. Chen, and Q. Liu, "Autotinybert: Automatic hyper-parameter optimization for efficient pre-trained language models," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, 2021, pp. 5146–5157.

[95] M. Javaheripi, G. de Rosa, S. Mukherjee, S. Shah, T. Religa, C. C. Teodoro Mendes, S. Bubeck, F. Koushanfar, and D. Dey, "Litetransformersearch: Training-free neural architecture search for efficient language models," in *Proceedings of the Advances Conference in Neural Information Processing Systems*, 2022, pp. 24 254–24 267.

[96] A. Klein, J. Golebiowski, X. Ma, V. Perrone, and C. Archambeau, "Structural pruning of large language models via neural architecture search," in *Proceedings of the 2023 AutoML Conference*, 2023. [Online]. Available: https://openreview.net/forum?id=SHlZcInS6C

[97] H. X. Choong, Y.-S. Ong, A. Gupta, and R. Lim, "Jack and masters of all trades: One-pass learning of a set of model sets from foundation models," *IEEE Computational Intelligence Magazine*, vol. 18, no. 3, pp. 29–40, 2023.

[98] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[99] X. Zhou, A. K. Qin, Y. Sun, and K. C. Tan, "A survey of advances in evolutionary neural architecture search," in *Proceedings of the 2021 IEEE congress on evolutionary computation*. IEEE, 2021, pp. 950–957.

[100] D. So, Q. Le, and C. Liang, "The evolved transformer," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2019, pp. 5877–5886.

[101] A. Gupta, Y.-S. Ong, L. Feng, and K. C. Tan, "Multiobjective multifactorial optimization in evolutionary multitasking," *IEEE Transactions on Cybernetics*, vol. 47, no. 7, pp. 1652–1665, 2016.

[102] K. K. Bali, A. Gupta, Y.-S. Ong, and P. S. Tan, "Cognizant multitasking in multiobjective multifactorial evolution: Mo-mfea-ii," *IEEE Transactions on Cybernetics*, vol. 51, no. 4, pp. 1784–1796, 2020.

[103] G. Kim and K. Cho, "Length-adaptive transformer: Train once with length drop, use anytime with search," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021, pp. 6501–6511.

[104] Y. Jiang, R. Yan, X. Yao, B. Chen, and B. Yuan, "Hexgen: Generative inference of foundation model over heterogeneous decentralized environment," *arXiv preprint arXiv:2311.11514*, 2023.

[105] Y. Ding, L. L. Zhang, C. Zhang, Y. Xu, N. Shang, J. Xu, F. Yang, and M. Yang, "Longrope: Extending llm context window beyond 2 million tokens," *arXiv preprint arXiv:2402.13753*, 2024.

[106] T. Akiba, M. Shing, Y. Tang, Q. Sun, and D. Ha, "Evolutionary optimization of model merging recipes," *arXiv preprint arXiv:2403.13187*, 2024.

[107] H. Li, Q. Ai, J. Chen, Q. Dong, Z. Wu, Y. Liu, C. Chen, and Q. Tian, "Blade: Enhancing black-box large language models with small domain-specific models," *arXiv preprint arXiv:2403.18365*, 2024.

[108] Z. Tao, T.-E. Lin, X. Chen, H. Li, Y. Wu, Y. Li, Z. Jin, F. Huang, D. Tao, and J. Zhou, "A survey on self-evolution of large language models," *arXiv preprint arXiv:2404.14387*, 2024.

[109] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[110] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[111] H. Bradley, H. Fan, T. Galanos, R. Zhou, D. Scott, and J. Lehman, "The openelm library: Leveraging progress in language models for novel evolutionary algorithms," in *Genetic Programming Theory and Practice XX*. Springer, 2024, pp. 177–201.

[112] Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang, "Wizardcoder: Empowering code large language models with evol-instruct," in *Proceedings of the Twelfth International Conference on Learning Representations*, 2023.

[113] G. Pinna, D. Ravalico, L. Rovito, L. Manzoni, and A. De Lorenzo, "Enhancing large language models-based code generation by leveraging genetic improvement," in *Proceedings of the European Conference on Genetic Programming*. Springer, 2024, pp. 108–124.

[114] A. E. Narin, "Evolutionary reward design and optimization with multimodal large language models," in *Proceedings of the First Vision and Language for Autonomous Driving and Robotics Workshop*, 2024.

[115] B. Yang, H. Su, N. Gkanatsios, T.-W. Ke, A. Jain, J. Schneider, and K. Fragkiadaki, "Diffusion-es: Gradient-free planning with diffusion for autonomous driving and zero-shot instruction following," *arXiv preprint arXiv:2402.06559*, 2024.

[116] P. Guo, F. Liu, X. Lin, Q. Zhao, and Q. Zhang, "L-autoda: Leveraging large language models for automated decision-based adversarial attacks," *arXiv preprint arXiv:2401.15335*, 2024.

[117] P. Shojaee, K. Meidani, S. Gupta, A. B. Farimani, and C. K. Reddy, "Llm-sr: Scientific equation discovery via programming with large language models," *arXiv preprint arXiv:2404.18400*, 2024.

[118] J. Mao, D. Zou, L. Sheng, S. Liu, C. Gao, Y. Wang, and Y. Li, "Identify critical nodes in complex network with large language models," *arXiv preprint arXiv:2403.03962*, 2024.

[119] F. Wu, X. Liu, and C. Xiao, "Deceptprompt: Exploiting llm-driven code generation via adversarial natural language instructions," *arXiv preprint arXiv:2312.04730*, 2023.

[120] N. Tao, A. Ventresque, and T. Saber, "Program synthesis with generative pre-trained transformers and grammar-guided genetic programming grammar," in *Proceedings of the 9th IEEE Latin American Conference on Computational Intelligence*, 2023, pp. 1–7.

[121] S. Kang and S. Yoo, "Towards objective-tailored genetic improvement through large language models," in *Proceedings of the 12th International Workshop on Genetic Improvement*. University of Melbourne, 2023.

[122] A. E. Brownlee, J. Callan, K. Even-Mendoza, A. Geiger, C. Hanna, J. Petke, F. Sarro, and D. Sobania, "Enhancing genetic improvement mutations using large language models," in *Proceedings of the International Symposium on Search Based Software Engineering*. Springer, 2023, pp. 153–159.

[123] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models," in *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering*, 2023, pp. 919–931.

[124] V. Tawosi, S. Alamir, and X. Liu, "Search-based optimisation of llm learning shots for story point estimation," in *Proceedings of the International Symposium on Search Based Software Engineering*. Springer, 2023, pp. 123–129.

[125] C. Yu, X. Liu, C. Tang, W. Feng, and J. Lv, "Gpt-nas: Neural architecture search with the generative pre-trained model," *arXiv preprint arXiv:2305.05351*, 2023.

[126] M. U. Nasir, S. Earle, J. Togelius, S. James, and C. Cleghorn, "Llmatic: Neural architecture search via large language models and quality-diversity optimization," *arXiv preprint arXiv:2306.01102*, 2023.

[127] A. Chen, D. M. Dohan, and D. R. So, "Evoprompting: Language models for code-level neural architecture search," in *Proceedings of the 37th Anual Conference on Neural Information Processing Systems*, 2023.

[128] C. Morris, M. Jurado, and J. Zutty, "Llm guided evolution-the automation of models advancing models," *arXiv preprint arXiv:2403.11446*, 2024.

[129] M. Zheng, X. Su, S. You, F. Wang, C. Qian, C. Xu, and S. Albanie, "Can gpt-4 perform neural architecture search?" *arXiv preprint arXiv:2304.10970*, 2023.

[130] H. Wang, Y. Gao, X. Zheng, P. Zhang, H. Chen, and J. Bu, "Graph neural architecture search with gpt-4," *arXiv preprint arXiv:2310.01436*, 2023.

[131] J. Zeng, G. Zhou, C. Li, Z. Sun, and Q. Zhao, "Discovering more effective tensor network structure search algorithms via large language models (llms)," *arXiv preprint arXiv:2402.02456*, 2024.

[132] G. Jawahar, M. Abdul-Mageed, L. V. Lakshmanan, and D. Ding, "Llm performance predictors are good initializers for architecture search," *arXiv preprint arXiv:2310.16712*, 2023.

[133] L. Chen, F. Xu, N. Li, Z. Han, M. Wang, Y. Li, and P. Hui, "Large language model-driven meta-structure discovery in heterogeneous information network," *arXiv preprint arXiv:2402.11518*, 2024.

[134] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[135] N. Tao, A. Ventresque, and T. Saber, "Multi-objective grammar-guided genetic programming with code similarity measurement for program synthesis," in *Proceedings of the 2022 IEEE Congress on Evolutionary Computation*. IEEE, 2022, pp. 1–8.

[136] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," *arXiv preprint arXiv:2310.03533*, 2023.

[137] M. Yan, J. Chen, J. M. Zhang, X. Cao, C. Yang, and M. Harman, "Coco: Testing code generation systems via concretized instructions," *arXiv preprint arXiv:2308.13319*, 2023.

[138] G. Fraser and A. Arcuri, "Evolutionary generation of whole test suites," in *Proceedings of the 11th International Conference on Quality Software*. IEEE, 2011, pp. 31–40.

[139] ——, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011, pp. 416–419.

[140] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, S. Yih, L. Zettlemoyer, and M. Lewis, "Incoder: A generative model for code infilling and synthesis," in *Proceedings of the 11th International Conference on Learning Representations*, 2022.

[141] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 122–158, 2017.

[142] A. Aleti, I. Moser, and L. Grunske, "Analysing the fitness landscape of search-based software testing problems," *Automated Software Engineering*, vol. 24, pp. 603–621, 2017.

[143] N. Albunian, G. Fraser, and D. Sudholt, "Causes and effects of fitness landscapes in unit test generation," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 1204–1212.

[144] V. Tawosi, R. Moussa, and F. Sarro, "Agile effort estimation: Have we solved the problem yet? insights from a replication study," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2677–2697, 2022.

[145] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[146] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," in *Proceedings of the 11th International Conference on Learning Representations*, 2023.

[147] E. ZITZLER, "Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization," *EUROGEN 2001, Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece, Septermber 12-21*, 2001.

[148] S. L. Lim, P. J. Bentley, and F. Ishikawa, "Scape: Searching conceptual architecture prompts using evolution," *arXiv preprint arXiv:2402.00089*, 2024.

[149] S. Sudhakaran, M. González-Duque, C. Glanois, M. Freiberger, E. Najarro, and S. Risi, "Prompt-guided level generation," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 179–182.

[150] A. J. Summerville, S. Snodgrass, M. Mateas, and S. Ontanón, "The vglc: The video game level corpus," *Proceedings of the 7th Workshop on Procedural Content Generation*, 2016.

[151] J. Lehman, K. O. Stanley *et al.*, "Exploiting open-endedness to solve problems through the search for novelty," in *Proceedings of the 11th International Conference on the Synthesis and Simulation of Living Systems*, 2008, pp. 329–336.

[152] H. Chefer, Y. Alaluf, Y. Vinker, L. Wolf, and D. Cohen-Or, "Attend-and-excite: Attention-based semantic guidance for text-to-image diffusion models," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–10, 2023.

[153] H. Berger, A. Dakhama, Z. Ding, K. Even-Mendoza, D. Kelly, H. Menendez, R. Moussa, and F. Sarro, "Stableyolo: Optimizing image generation for large language models," in *Proceedings of the International Symposium on Search Based Software Engineering*. Springer, 2023, pp. 133–139.

[154] K. M. Jablonka, Q. Ai, A. Al-Feghali, S. Badhwar, J. D. Bocarsly, A. M. Bran, S. Bringuier, L. C. Brinson, K. Choudhary, D. Circi *et al.*, "14 examples of how llms can transform materials science and chemistry: a reflection on a large language model hackathon," *Digital Discovery*, vol. 2, no. 5, pp. 1233–1250, 2023.

[155] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988.

[156] Y. G. N. Teukam, F. Zipoli, T. Laino, E. Criscuolo, F. Grisoni, and M. Manica, "Integrating genetic algorithms and language models for enhanced enzyme design," 2024. [Online]. Available: https://chemrxiv.org/engage/chemrxiv/article-details/65f0746b9138d23161510400

[157] R. Suzuki and T. Arita, "An evolutionary model of personality traits related to cooperative behavior using a large language model," *Scientific Reports*, vol. 14, no. 1, p. 5989, 2024.

[158] S. Jain and B. C. Wallace, "Attention is not explanation," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 3543–3556.

[159] M. J. Buehler, "Accelerating scientific discovery with generative knowledge extraction, graph-based representation, and multimodal intelligent graph reasoning," *arXiv preprint arXiv:2403.11996*, 2024.

[160] E. L. Buehler and M. J. Buehler, "X-lora: Mixture of low-rank adapter experts, a flexible framework for large language models with applications in protein mechanics and molecular design," *APL Machine Learning*, vol. 2, no. 2, 2024.

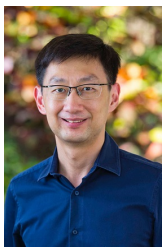
[161] A. Ghafarollahi and M. J. Buehler, "Protagents: protein discovery via large language model multi-agent collaborations combining physics and machine learning," *Digital Discovery*, 2024.
[162] B. Ni and M. J. Buehler, "Mechagents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge," *Extreme Mechanics Letters*, vol. 67, p. 102131, 2024.
[163] C. Qian, S. Liang, Y. Qin, Y. Ye, X. Cong, Y. Lin, Y. Wu, Z. Liu, and M. Sun, "Investigate-consolidate-exploit: A general strategy for inter-task agent self-evolution," *arXiv preprint arXiv:2401.13996*, 2024.
[164] W. Zhang, K. Tang, H. Wu, M. Wang, Y. Shen, G. Hou, Z. Tan, P. Li, Y. Zhuang, and W. Lu, "Agent-pro: Learning to evolve via policy-level reflection and optimization," *arXiv preprint arXiv:2402.17574*, 2024.

Xingyu Wu received the B.Sc degree from University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2018, and the Ph.D degree in the School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei, China, in 2023. Dr. Wu is currently a postdoctoral fellow in the Department of Data Science and Artificial Intelligence, the Hong Kong Polytechnic University (PolyU), Hong Kong SAR, China. His research interests include causality-based machine learning, automatic machine learning, and large foundation model. Dr. Wu has published over 30 papers in prestigious conferences and journals in machine learning and artificial intelligence.

Sheng-Hao Wu (Member, IEEE) received the B.S. degree and the Ph. D. degree in computer science and technology from South China University of Technology, Guangzhou, China, in 2019 and 2023, respectively. He is currently a postdoctoral research fellow at the Department of Data Science and Artificial Intelligence, the Hong Kong Polytechnic University. His research interests mainly include computational intelligence, machine learning, and their applications in real-world problems.

Jibin Wu (Member, IEEE) received the B.E. and Ph.D degree in Electrical Engineering from National University of Singapore, Singapore in 2016 and 2020, respectively. Dr. Wu is currently an Assistant Professor in the Department of Data Science and Artificial Intelligence and the Department of Computing, the Hong Kong Polytechnic University. His research interests broadly include brain-inspired artificial intelligence, neuromorphic computing, computational audition, speech processing, and machine learning. Dr. Wu has published over 30 papers in prestigious conferences and journals in artificial intelligence and speech processing, including NeurIPS, AAAI, TPAMI, TNNLS, TASLP, Neurocomputing, and IEEE JSTSP. He is currently serving as the Associate Editors for IEEE Transactions on Neural Networks and Learning Systems and IEEE Transactions on Cognitive and Developmental Systems. He also serves as the Editor for the Natural Language Processing Journal.

Liang Feng (Senior Member, IEEE) received the Ph.D. degree from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2014. He is currently a Professor with the College of Computer Science, Chongqing University, Chongqing, China. His research interests mainly include computational and artificial intelligence, memetic computing, Big Data optimization and learning, as well as transfer learning and optimization. He has been honored with the 2019 IEEE TEVC Outstanding Article Award, 2023 IEEE TETCI Outstanding Article Award, and 2024 IEEE CIM Outstanding Article Award. He is an Associate Editor for IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, IEEE CIM, Memetic Computing. He is also the founding Chair of the IEEE CIS Intelligent Systems Applications Technical Committee Task Force on Transfer Learning & Transfer Optimization.



Kay Chen Tan (Fellow, IEEE) received the B.Eng. degree (First Class Hons.) and the Ph.D. degree from the University of Glasgow, U.K., in 1994 and 1997, respectively. He is currently a Chair Professor (Computational Intelligence) of the Department of Data Science and Artificial Intelligence, the Hong Kong Polytechnic University. He has published over 300 refereed articles and seven books. Prof. Tan is currently the Vice-President (Publications) of IEEE Computational Intelligence Society, USA. He has served as the Editor-in-Chief of the IEEE Computational Intelligence Magazine from 2010 to 2013 and the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2015 to 2020, and currently serves as the Editorial Board Member for more than ten journals. He is the Chief Co-Editor of Springer Book Series on Machine Learning: Foundations, Methodologies, and Applications.