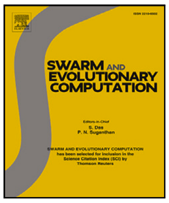
Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo



Survey paper

When large language model meets optimization

Sen Huang [a], Kaixiang Yang [b], Sheng Qi [c], Rui Wang [c,*]

[a] *School of Electronic and Information Engineering, South China University of Technology, China*
[b] *School of Computer Science and Engineering, South China University of Technology, China*
[c] *College of Systems Engineering, National University of Defense Technology, China*

ARTICLE INFO

Keywords:
Large language model
Optimization algorithm
Evolutionary computation

ABSTRACT

Optimization algorithms and large language models (LLMs) enhance decision-making in dynamic environments by integrating artificial intelligence with traditional techniques. LLMs, with extensive domain knowledge, facilitate intelligent modeling and strategic decision-making in optimization, while optimization algorithms refine LLM architectures and output quality. This synergy offers novel approaches for advancing general AI, addressing both the computational challenges of complex problems and the application of LLMs in practical scenarios. This review outlines the progress and potential of combining LLMs with optimization algorithms, providing insights for future research directions.

1. Introduction

Optimization algorithms (OA) is becoming increasingly important as a class of heuristic search algorithms in the broad field of artificial intelligence and machine learning [1–3]. OA draws on the natural mechanisms of biological evolution, including processes such as natural selection, heredity, mutation, and hybridization, for solving complex optimization problems. These algorithms are widely used in many fields due to their global search capability, low dependence on problem structure, and ease of parallelization. Optimization algorithms pivotal in diverse fields such as logistics, finance [4], healthcare [5], and artificial intelligence [6], aim to identify the best solution from available alternatives. They are essential for making decisions efficiently and effectively in an era of rapidly increasing data complexity and volume. The continuous advancement in optimization techniques has resulted in significant enhancements to algorithmic strategies, each customized to address specific types of problems and operational constraints. From deterministic methods addressing linear problems to stochastic approaches for global optimization under uncertainty, optimization algorithms hold promise across a broad spectrum of research and practical applications. With the development of technology, especially when dealing with large-scale, high-dimensional and dynamically changing optimization problems, traditional algorithms often face performance bottlenecks. Evolutionary computing provides effective solutions to these problems with its unique search strategy. In addition, the flexibility and adaptability of evolutionary computation enable it to be combined with a variety of other computational techniques to form hybrid algorithms for further performance enhancement.

In the rapidly evolving field of artificial intelligence, Large Language Models (LLMs) such as GPT (Generative Pre-trained Transformer) [7] provide a significant breakthrough with their advanced natural language understanding and generation capabilities. These models have revolutionized applications ranging from automated writing assistants to sophisticated conversational agents. LLMs have become pivotal in advancing fields like natural language processing, image recognition with their extensive parameters and deep learning capabilities, and machine learning, offering robust solutions for complex data-driven challenges. LLMs have achieved breakthroughs in traditional NLP tasks like text generation and language translation through extensive data training, and they also show promising potential in the emerging fields of algorithm design and optimization. Traditional optimization algorithm design, dependent on human expertise [8], is both time-consuming and potentially limited by the experts' knowledge. The advent of large-scale language models, however, has transformed this arena. These models learn extensive algorithmic patterns and strategies, enabling them to devise new algorithms and tailor solutions to specific challenges.

Furthermore, constructing and training large language models require significant computational resources and large datasets [9], which escalate research and development costs and constrain the applicability and generalization of LLMs. Optimization algorithms are crucial in developing LLMs, enabling researchers to efficiently tailor and refine model structures for specific applications. By enhancing the training process, boosting computational efficiency, and lowering resource consumption, these algorithms facilitate the construction and

* Corresponding author.
E-mail addresses: huangsen@scut.edu.cn (S. Huang), yangkx@scut.edu.cn (K. Yang), qisheng@nudt.edu.cn (S. Qi), ruiwangnudt@gmail.com (R. Wang).

https://doi.org/10.1016/j.swevo.2024.101663
Received 13 May 2024; Received in revised form 15 June 2024; Accepted 10 July 2024
Available online 24 July 2024

Table 1
Glossary of professional terms.

Terminology	Explanatory notes
Large Language Models (LLMs)	Large parameter models are pre-trained on large amounts of data
Pre-trained Language Models (PLMs)	Pre-trained models on large-scale datasets
Generative Pre-Trained Transformer (GPT)	Transformer-based generative pre-training of large language models
Reinforcement Learning (RL)	With feedback and penalties to maximize long-term cumulative rewards
Deep Reinforcement Learning (DRL)	Reinforcement learning based on deep learning methods
Neural Architecture Search (NAS)	Using search algorithm to discover the desired neural network structure
Prompt learning	Using prompt templates to elicit factual knowledge in PLMs
Optimization Algorithms (OAs)	Iterative approximation of the optimal solution of the problem
Evolutionary Algorithms (EAs)	Optimization algorithms inspired by biological evolution
Multi-Objective Optimization (MOO)	Optimization with multiple decision variables and objective functions
Black-Box Optimization (BBO)	Optimization in the absence of an explicit objective function form
Hyperparameter Optimization (HPO)	Optimization methods for finding the optimal hyperparameters of a model
Genetic Algorithms (GAs)	Optimization algorithms inspired by natural selection and genetics theory
Estimation of Distribution Algorithm (EDA)	Heuristic optimization algorithms guided by probabilistic models

application of large-scale language models. These algorithms enhance the models' generalization capabilities and robustness, enabling improved performance amidst real-world complexity and uncertainty. The goal in designing optimization algorithms for LLMs is to enhance their operational efficiency and reduce resource consumption, without compromising, and possibly improving, model performance.

Table 1 summarizes and outlines the main terminology and explanations to provide a clear overview of the relevant methods. To systematically gather relevant literature, we employed Web of Science for retrieval, covering databases such as SpringerLink, ScienceDirect, IEEEXplore, ACM Digital Library, and numerous conference paper. In addition, preprint articles are also included.

Recently, Pallagani [10] explored the prospects of incorporating LLMs into automated planning and scheduling (APS). Wu et al. [11] systematically investigated interdisciplinary research between evolutionary algorithms (EAs) and LLMs, exploring the complementary advantages of LLMs and EAs. Unlike the aforementioned surveys, this review aims to systematically analyze research on developing optimization algorithms (OAs) with LLMs, and optimizing LLMs with optimization algorithms. Compared to APS problems and EAs, OAs constitute a broader category. This review summarizes related research and application scenarios, and explores the diverse aspects of these applications. The contribution of this paper is summarized as follows:

1. We provide a comprehensive review of the development process of large language models and optimization algorithms, and systematically analyze the research on developing OAs using LLMs and optimizing LLMs with OAs. This paper is the latest review that focuses on the development and application of OAs and LLMs.
2. We conduct a thorough investigation of the practical applications of integrating OAs with LLMs, highlighting the significant impact of this interdisciplinary field in real-world scenarios.
3. We summarize the research trends and future research directions of combining LLMs with OAs, demonstrating the practicality and potential of this field to promote further research.

The remainder of this article is organized as follows: Section 2 provides a comprehensive review of large language model development, also known as macromodels, tracing their progression from basic predictive models to sophisticated systems that can comprehend and generate human-like text. Section 3 focus to optimization algorithms, concisely tracing their evolution from basic iterative methods to advanced algorithms essential for efficiently scaling AI models. Section 4 examines research that approaches large language models as optimization problems, highlighting innovative methods employed as search operators and in designing optimization algorithms. Section 5 discusses recent advances in optimization algorithms tailored for refining large-scale models. It highlights how these algorithms can enhance design, boost efficiency, and improve the performance of LLMs. Section 6 examines the practical applications of integrating optimization algorithms with LLMs, highlighting real-world implementations and their benefits. Section 7 is future outlook and research trends, in which summarizes the insights gained from this exploration and provides an outlook on the potential future developments in this exciting intersection of AI research. Section 8 summarizes the contributions of this review and the limitations of combining large models with optimization algorithms.

In summary, we conducted a comprehensive study on the development and application of optimization algorithms for large models, aiming to provide valuable references and insights for future research.

2. Large language models

Language has a crucial role in human cognition, enabling communication and expression from early childhood to adulthood [12]. Teaching robots to imitate human-like language skills is a difficult task due to their intrinsic lack of cognitive capacity for understanding and expressing language. Computational linguistics aims to close this divide by using advanced Artificial Intelligence (AI) algorithms to enable machines to possess reading, writing, and communication skills that like those of humans [13].

The rise of Large Language Models (LLMs) undoubtedly represents an important milestone in the evolution of Natural Language Processing (NLP). These models, such as GPT-3 and GPT-4 of the GPT family [7], are built on the Transformer architecture and have up to billions of parameters. They achieve a deep understanding of natural language and generative capabilities by pre-training on massive text datasets. The evolution of LLMs has gone through several notable stages: from the early days of Statistical Language Models (SLMs) and Neuro-Linguistic Models (NLMs), to Pre-Trained Language Models (PLMs), and ultimately to today's large-scale language models. While PLMs like BERT and GPT-2 have achieved remarkable success in NLP tasks, the emergence of LLMs has revolutionized the game in this field [14]. Not only can they be adapted to a wide range of tasks through large-scale pre-training, but they are also further optimized through fine-tuning, demonstrating a wide range of potential in application scenarios such as chatbots, search engine optimization and office automation The rise of Large Language Models (LLMs) undoubtedly represents an important milestone in the evolution of Natural Language Processing (NLP). These models, such as GPT-3 and GPT-4 of the GPT family [7], are built on the Transformer architecture and have up to billions of parameters. They achieve a deep understanding of natural language and generative capabilities by pre-training on massive text datasets. The evolution of LLMs has gone through several notable stages: from the early days of Statistical Language Models (SLMs) and NLMs, PLMs, and ultimately to today's large-scale language models. While PLMs like BERT and GPT-2 have achieved remarkable success in NLP tasks, the emergence of LLMs has revolutionized the game in this field. Not only can they be adapted to a wide range of tasks through large-scale pre-training, but they are also further optimized through fine-tuning, demonstrating a wide range

of potential in application scenarios such as chatbots, search engine optimization, and office automation [9].

The excellence of Large Language Models (LLMs) in the field of Natural Language Processing (NLP) is due to several key components of their design, which together give LLMs powerful language understanding and generation capabilities [9]. First, “pre-training” is one of the core processes of LLMs. By pre-training on large-scale textual datasets, LLMs are able to learn the basic structures and patterns of language. These datasets typically contain billions of words covering a wide range of topics and language styles, allowing the models to capture the diversity and complexity of the language. Second,“adaptability” is another key characteristic of LLMs. After pre-training, LLMs can be further fine-tuned to adapt to specific downstream tasks, such as text classification, sentiment analysis, or machine translation. This adaptability allows LLMs to optimize their performance for specific tasks, leading to better results in various NLP challenges [15]. In terms of “applications”, the broad applicability of LLMs is another reason for their popularity. Not only do they perform well in traditional NLP tasks, but they can also be applied to a wider range of domains, such as the development of chatbots, the optimization of search engines, the construction of content recommendation systems, and the development of automated office tools. Finally, “performance evaluation” is critical to ensure the reliability and effectiveness of LLMs. Through a series of standardized testing and evaluation protocols, researchers are able to quantify the performance of LLMs and ensure that they work consistently across a range of tasks and conditions. Performance evaluation also includes studies of model bias, fairness, and interpretability, which are key factors in improving model quality and trust.

LLMs are becoming a key driver in the field of AI, and their development and application are attracting widespread attention from industry and academia. LLMs represented by ChatGPT and GPT-4 have not only made significant progress in technology but also promoted in-depth discussions on artificial general intelligence (AGI) conceptually [9]. OpenAI’s technical article proposes that GPT-4 [16] may be an early attempt to move towards AGI, which all indicates the critical position of LLMs in the development of AI [17]. In the field of Natural Language Processing (NLP), LLMs are becoming a common tool for solving various linguistic tasks, changing the previous research and application paradigm. The Information Retrieval (IR) field is also feeling the winds of change, with traditional search engines facing the challenge of emerging information access methods such as AI chatbots, such as New Bing3, which is an attempt to enhance search results based on LLMs [18]. In addition, the field of computer vision (CV) is exploring multimodal models that combine vision and language, and the multimodal input support of GPT-4 is a manifestation of this trend. The rise of LLMs heralds the birth of a whole new ecosystem of apps based on these advanced models. Microsoft 365 leverages LLMs (e.g., Copilot) to automate the office, while OpenAI introduces plug-in functionality in ChatGPT, all of which demonstrate the potential of LLMs to enhance productivity and extend application scenarios [19].

While LLMs have brought about many positive changes, they also present a number of challenges, particularly in terms of the security and accuracy of the generated content. In addition, the training of LLMs requires substantial computational resources, which is a challenge for research institutes as it limits the ability to perform extensive experiments and optimization of the models [20]. We have summarized the relevant restrictions below: (1) **Computational resources**: LLMs demand substantial computational resources for training and inference, which can pose challenges for implementing optimization algorithms at scale. Ensuring access to adequate computational infrastructure remains a significant hurdle. (2) **Data efficiency**: While LLMs have demonstrated impressive performance, they often require large amounts of data for effective training. This reliance on extensive datasets can be a bottleneck for optimization algorithms, especially in scenarios where data availability is limited or costly to obtain. (3) **Interpretability and explainability**: The inherent complexity of LLMs poses challenges for the interpretability and explainability of optimization algorithms. Understanding the decision-making process of these models and interpreting their outputs can be challenging, particularly in critical applications where transparency is essential. (4) **Generalization and Robustness**: Ensuring the generalization and robustness of optimization algorithms trained using LLMs is another key challenge. Over-reliance on specific patterns in the training data may lead to poor generalization performance on unseen data or vulnerability to adversarial attacks.

Optimization algorithms are integrated with LLMs to enhance their efficiency and effectiveness in various ways. This integration involves optimizing the training process, including adjusting batch sizes and learning rates, and employing advanced techniques such as dynamic sampling to minimize data and computational demands. Moreover, the use of regularization techniques and constrained optimization enhances the interpretability of the models and the transparency of their decision-making processes. This is particularly crucial in fields such as healthcare and law, where a high degree of interpretability is essential. Furthermore, optimization algorithms improve the generalization ability and robustness of LLMs by diversifying training and simulating adversarial environments, including the use of cross-validation and model ensembles, to enhance model performance on new data. Overall, the synergy between optimization algorithms and LLMs not only facilitates the practical application of these models but also addresses computational resource limitations and boosts data efficiency.

3. Classic optimization algorithm

3.1. Traditional optimization algorithms for optimization problems

Optimization algorithms have wide applications in fields such as industry, economics, and management. With the rapid development of artificial intelligence, optimization algorithms play a crucial role in achieving intelligence and automation. Table 2 summarizes classic optimization algorithms. Traditional optimization algorithms include deterministic optimization algorithms, approximation algorithms, and heuristic algorithms. Deterministic optimization algorithms include Linear Programming (LP) [21], Integer Programming (IP) [22], Mixed Integer Programming (MIP) [23], Convex Optimization [24], Adaptive Dynamic Programming (ADP) [25], etc. Deterministic optimization algorithms can guarantee finding the global optimal solution, but they are generally not suitable for large-scale problems. Polynomial-time approximation algorithms can find a good solution within a reasonable time frame, but they do not guarantee optimality. In other words, approximation algorithms do not guarantee finding the global optimal solution. For some specific problems, optimality guarantees may not exist at all. Heuristic algorithms employ specially designed functions to intelligently explore the solution space [47]. Heuristic algorithm rely on intuitive rules, trial-and-error strategies, and practical insights to approximate solutions within acceptable bounds. Heuristic algorithms are particularly effective for problems with high-dimensional search spaces or combinatorial complexities where exact solutions are impractical. Heuristic algorithms include greedy algorithm [26], Tabu Search [27–29], genetic algorithm [30,31], differential evolution algorithm [48,49], Cooperative co-evolution algorithm [50–52]. Heuristic algorithms have been widely used due to their excellent computational performance, but they typically require customization and domain expertise for specific problems. Additionally, heuristic algorithms may converge to local optimal solutions and have high time complexity.

Recently, the superiority of using Estimation of Distribution Algorithm (EDA) in solving optimization problems has been demonstrated. EDA is a prominent optimization technique that employs probabilistic models to guide the search process. Unlike traditional evolutionary algorithms, which rely on mutation and recombination operators, EDA focuses on building and updating a probabilistic model of promising solutions. This model is then sampled to generate new candidate solutions, effectively balancing exploration and exploitation. Yang

Table 2
Summary of classic optimization algorithm.

Categories	Algorithms	Application scenario
Deterministic optimization algorithms	Linear Programming (LP) [21], Integer Programming (IP) [22], Mixed Integer Programming (MIP) [23], Convex Optimization [24], Adaptive Dynamic Programming (ADP) [25]	Guaranteed to find a globally optimal solution, not applicable to large-scale problems.
Heuristic algorithms	Greedy algorithm [26], Tabu search [27–29], genetic algorithm [30,31]	Approximate solutions for solving high-dimensional search space or combinatorial complexity problems.
Distribution Estimation Algorithms (EDA)	ACSEDA [32], LS-EDA [33], IDE-EDA [34]	Calculation of covariances for a large number of promising individuals based on a Gaussian distribution model.
Reinforcement learning methods	Q-learning [35], Double Q-learning [36], REINFORCE [37], TRPO [38], PPO [39], AC [40], DPG [41] DQN [42], Double DQN [43], S2V-DQN [44], DDPG [45], A3C [46]	Learning to improve strategies by interacting with the environment to maximize long-term cumulative rewards.

Table 3
Reinforcement learning methods for optimization problems.

	Algorithms	Descriptions	Types
Classic reinforcement learning methods	Q-learning [35]	A classic value-based RL algorithm that extracts the optimal policy by updating the Q-function.	Value-based
	Double Q-learning [36]	An improved version of Q-learning. It alleviates the overestimation problem by using two Q-functions.	Value-based
	REINFORCE [37]	Also known as the Monte Carlo Policy Gradient Reinforcement Learning algorithm, it update parameters along the gradient direction to maximize the objective function.	Policy-based
	TRPO [38]	It ensures that the Kullback–Leibler divergence between the new and the old policy does not exceed a predefined threshold during each policy update.	Policy-based
	PPO [39]	The improved version of TRPO. It is easier to implement and requires less computation.	Policy-based
	AC [40]	It learns both the policy and the value function simultaneously.	Actor–critic
	DPG [41]	It employs deterministic policy function, which greatly reduces the need for sampling and enables the handling of larger action spaces.	Actor–critic
Deep reinforcement learning methods	DQN [42]	The combination of deep learning and Q-learning, using deep neural networks as value function estimators, and introducing experience replay and target networks.	Value-based
	Double DQN [43]	An improved version of DQN. It alleviates the overestimation problem by using two Q-networks.	Value-based
	S2V-DQN [44]	The combination of RL and graph embedding. It utilizes a graph embedding network called structure2vec (S2V) to represent policies, and uses multi-step DQN to learn greedy policies.	Value-based
	DDPG [45]	It employs two different parameterized deep neural networks to represent the value network and the deterministic policy network.	Policy-based
	A3C [46]	By utilizing asynchronous gradient descent to optimize the parameters of deep neural networks (DNNs), it significantly improved the efficiency of policy optimization.	Actor–critic

et al. [32] proposed ACSEDA based on the Gaussian distribution model, and calculates the covariance according to an enlarged number of promising individuals. In contrast to solely relying on solutions from the current generation to estimate the Gaussian model, EDA^2 [53] incorporates a strategy where a set number of high-quality solutions from previous generations are retained in an archive. These historical solutions are then utilized to aid in estimating the covariance matrix of the Gaussian model. Dong et al. [33] introduced a latent space-based EDA (LS-EDA), which converts the multivariate probabilistic model of Gaussian-based EDA into a principal component latent subspace with reduced dimensionality. This transformation effectively decreases the complexity of EDA while preserving its probability model, ensuring that crucial information is retained. Consequently, LS-EDA enhances performance scalability for large-scale global optimization problems. In recent times, hybrid EDAs have emerged as a prominent research focus. Li et al. [34] proposed IDE-EDA, an enhanced version of DE achieved by integrating the EDA. Zhang et al. [54] introduced a new hybrid evolutionary algorithm for continuous global optimization problems, Zhou et al. [55] proposed a fusion of an Estimation of Distribution Algorithm (EDA) with both economical and costly local search (LS) methods. This integration aims to leverage both global statistical insights and individual location-specific information for enhanced optimization performance.

3.2. *Reinforcement learning methods for optimization problems*

Another famous learning paradigm is reinforcement learning. Reinforcement learning learns and improves its strategy by interacting with the environment, aiming to maximize long-term cumulative rewards. The aforementioned heuristic algorithms typically find optimal or approximate solutions by searching the solution space, guided by heuristic information. However, they do not consider interaction with the environment during the search process. Unlike supervised and unsupervised learning, in reinforcement learning, agents can only learn through trial and error, rather than relying on labeled data or searching for the inherent structure of the data. Furthermore, reinforcement learning can be categorized into classical reinforcement learning methods and deep reinforcement learning methods, which will be introduced below. Table 3 summarizes reinforcement learning methods for optimization problems.

3.2.1. *Classic reinforcement learning methods for optimization problems*

According to [56], classic RL methods can be divided into model-based and model-free approaches. Model-free methods can be further categorized into value-based, policy-based, and actor–critic methods. Value-based methods seek the optimal policy by estimating the value function. This approach is suitable for smaller state spaces and discrete action spaces but faces challenges to extend to continuous action spaces and has the “high bias” problem. The error between the estimated value function and the actual value function is difficult to eliminate. Policy-based methods, on the other hand, do not require estimating the value function. Instead, they directly fit the policy function using neural networks. By training and updating the policy parameters, the optimal policy is generated. This method is suitable for continuous action spaces but requires sampling a large number of trajectories, and there is a huge difference between each trajectory, leading to the “high variance” problem. To address the contradiction between high bias and high variance, the actor–critic method emerges. The actor–critic method constructs an agent that can both output policies and evaluate their quality in real-time using the value function. Generally, an actor–critic network consists of two parts: the actor network and the critic network. The actor network is used to generate policies to approximate the policy function, while the critic network is used to evaluate policies to approximate the value function. Representative works of these methods will be introduced below.

Q-learning [35] is a classic value-based RL algorithm and is currently the most widely used model-free RL algorithm. Q-learning first initializes a Q-function, typically represented as a Q-table. It selects an action based on the current state using the ϵ-greedy strategy, performs the selected action, and observes the reward obtained and the next state transitioned to. The Q-function is updated using the Bellman equation, repeat the above steps and update the Q value until the stop condition is reached. Finally, the optimal policy is extracted based on the learned Q-function. Double Q-learning [36] is an improved version of the Q-learning algorithm which alleviates the overestimation problem by using two Q-functions. In each round of interaction with the environment, one of the value function estimators is alternately selected to choose actions, while the other is used for action value estimation. Existing research has demonstrated that the Double Q-learning method achieves higher stability and greater long-term returns.

The REINFORCE algorithm [37], also known as the Monte Carlo Policy Gradient Reinforcement Learning algorithm, is a classical policy gradient method. The goal of the reinforcement is to update parameters along the gradient direction to maximize the objective function. On the basis of the classical policy gradient algorithms, Trust Region Policy Optimization (TRPO) [38] ensures that the Kullback–Leibler Divergence between the new and the old policy does not exceed a predefined threshold during each policy update. This threshold represents the “trust region” of policy updates, indicating the similarity between the new and old policies, thereby ensuring the stability of policy optimization. Proximal Policy Optimization (PPO) [39] is an improvement over TRPO, which is simpler to implement and requires less computation in practical use.

Actor–Critic (AC) [40] algorithm combines the advantages of both policy-based and value-based methods. It learns both the policy and the value function simultaneously. The actor trains the strategy based on the value function feedback from the critic, while the critic trains the value function using the Time Difference Method (TD) for single step updates. The aforementioned REINFORCE algorithm uses a stochastic policy function, which outputs the probability distribution of actions for a given state, and then selects an action based on the probability distribution. In contrast to the REINFORCE algorithm, Deterministic Policy Gradient (DPG) [41] algorithm employs deterministic policy function, which directly outputs a deterministic action for given states and update policy parameters by maximizing expected returns. DPG integrates deterministic policy gradients only in the state space, greatly reducing the need for sampling and enabling the handling of larger action spaces. However, the coupling between policy updates and value estimation in DPG leads to insufficient stability, particularly being highly sensitive to hyperparameters. The difficulty of tuning hyperparameters in actor–critic algorithms and challenges in reproducibility make them hard to apply in practical scenarios. When extended to application fields, the robustness of the algorithm is also one of the most concerned core issues.

The above methods are relatively simple and intuitive, easy to understand and implement, and suitable for smaller-scale problems. In relatively fewer samples, classical RL methods typically achieve good performance, especially in stable environments and reward settings. More importantly, the above method exhibits strong interpretability, allowing for a clear understanding of the agent’s behavior and decision-making process within the environment.

3.2.2. Deep reinforcement learning methods for optimization problems

Although RL has achieved remarkable achievements, previous methods have often struggled to handle high-dimensional data such as images, text, etc. This limitation has constrained its ability to deal with complex tasks and environments. Classical RL methods often find it challenging to strike a good balance between exploration and exploitation, leading to susceptibility to local optima, especially in high-dimensional and complex environments where issues of insufficient exploration are more pronounced. The reason for the aforementioned situation is that RL algorithms, like other algorithms, face challenges such as memory complexity, computational complexity, sample complexity, etc [57]. However, the powerful representation learning capability and function approximation capability of deep learning bring a completely new solution for RL.

Deep learning is a branch of machine learning aimed at using multi-layer neural network models to learn representations and features of data, and solving various tasks through these representations and features. Deep learning models have strong nonlinear function approximation capabilities, allowing them to learn more complex and accurate data representations and features. Deep learning models have strong generalization and representation learning capabilities, enabling them to learn more accurate and effective policies or value functions, thereby allowing RL agents to tackle more complex and high-dimensional tasks and environments, achieving higher performance and accuracy. Deep reinforcement learning (DRL) is the product of integrating both RL and DL. Similarly, DRL methods are also divided into three types: value-based, policy-based, and actor–critic methods. Among them, value-based DRL employs DL to approximate value functions, while policy-based DRL uses DL to approximate policies and solve decision-making policies based on policy gradient rules. The following will introduce representative works in DRL.

In 2013, Mnih et al. from DeepMind combined DL with Q-learning, proposing the groundbreaking Deep Q-network (DQN) [42]. DQN is a DRL algorithm based on Q-learning. On one hand, it utilizes deep neural networks as value function estimators, and on the other hand, it introduces experience replay and target networks. The experience replay mechanism breaks the high dependency between sampled samples, while the target network alleviates the instability of neural networks during training. These two mechanisms work together to enable the DQN algorithm to achieve performance close to or even surpassing human levels in most Atari games. Double DQN [43], based on Double Q-learning, is an improvement over DQN. Similar to how DQN extends Q-learning, DDQN addresses the overestimation issue by using two Q-networks. DDQN achieves better stability and algorithm performance compared to DQN. In 2017, Dai et al. combined RL with graph embedding and proposed S2V-DQN [44]. They utilized a graph embedding network called structure2vec (S2V) to represent the policy in greedy algorithms and employed multi-step DQN to learn greedy policies parameterized by the graph embedding network. The S2V-DQN algorithm generates high-quality solutions faster, sometimes finding better solutions than commercial solvers within a longer timeframe.

In 2015, inspired by the ideas of DQN, Lillicrap et al. combined neural networks with the DPG algorithm to propose DDPG [45]. DDPG employs two different parameterized deep neural networks to represent the value network and the deterministic policy network. The policy network is responsible for updating the policy, while the value network outputs the Q-values for state–action pairs. Similar to DQN, DDPG also utilizes target networks to overcome instability issues during network updates. Zhang et al. [58] considered the feasibility constraints of NP-hard problems, embedding heuristic functions into the transformer architecture, and applying DRL to combinatorial optimization problems with feasibility constraints. Ma et al. [59] introduced a Graph Pointer Network (GPN) to solve the classical TSP problem and combined it with a hierarchical RL framework to address the TSP problem with time window constraints. Multiple Traveling Salesman Problems (MTSP) are more complex, and Hu et al. [60] designed a network consisting of a shared graph neural network and distributed policies to learn a common policy expression suitable for MTSP. Experimental results demonstrated the effectiveness of this approach on large-scale problems.

In 2016, Mnih et al. [46] developed an improved Actor–Critic algorithm called A3C (Asynchronous Advantage Actor–Critic). By utilizing asynchronous gradient descent to optimize the parameters of deep neural networks (DNNs), A3C significantly improved the efficiency of policy optimization. Vinyals et al. [61] proposed the Pointer Network model for solving combinatorial optimization problems, which initiated a series of research studies on utilizing DNN for solving combinatorial optimization problems. This model was inspired by the Seq2Seq model in machine translation. It employs a deep neural network-based encoder to encode the input sequence of the combinatorial optimization problem (such as city coordinates), then utilizes a decoder and attention mechanism to compute the selection probabilities of each node. Finally, it selects nodes in an autoregressive manner until obtaining a complete solution. Due to the supervised nature of the training method proposed by Vinyals et al. [61], the quality of the solutions it obtains will never exceed the quality of the sample solutions. Recognizing this limitation, Bello et al. [62] employed a RL approach to train the Pointer Network model. They treated each problem instance as a training sample, using the objective function of the problem as feedback signals, and trained the model based on REINFORCE. They also introduced a critic network as a baseline to reduce training variance. Furthermore, Nazari et al. [63] extended the Pointer Network to handle dynamic VRP problems. They replaced the LSTM in the Encoder input layer with a simple one-dimensional convolutional layer, effectively reducing computational costs. While maintaining optimization effectiveness, the training time was reduced by 60%.

In recent years, the Transformer [64] has achieved tremendous success in the field of natural language processing. Its multi-head attention mechanism enables better extraction of deep features from problems. In view of this, several recent studies have drawn inspiration from the Transformer for solving combinatorial optimization problems. Deudon et al. [65] improved traditional pointer network models by incorporating ideas from the Transformer. They utilized a similar structure to the Transformer in the encoder, while the decoder employed linear mapping of the decisions from the last three steps to obtain a reference vector, thereby reducing model complexity. The attention calculation method remained the same as in traditional Pointer Network models, and the classic REINFORCE method is still used to train the model. Kool et al. [66] proposed a new method capable of solving multiple combinatorial optimization problems using attention mechanisms. The attention calculation method in this model adopted the self-attention computation method from the transformer, with additional computational layers to enhance performance. They further designed a greedy rollout baseline to replace the Critic network, leading to significant improvements in optimization performance.

Deep reinforcement learning, as one of the most popular research directions in the field of artificial intelligence, has shown great potential in solving complex tasks and addressing various real-world problems. However, DRL also has its limitations, such as data requirements, sample efficiency, computational resources, interpretability, etc. Despite achieving some success in both research and application domains, DRL fundamentally remains constrained to simulated environments with ideal, highly structured experimental data design. They heavily rely on the design and training of specific models. Therefore, there is a growing interest in the design and optimization of automatic algorithms.

4. LLMs as optimization

4.1. LLMs as the black-box optimization search model

There is a strong alignment between Large Language Models (LLMs), which are powerful in generating creative texts, and Evolutionary

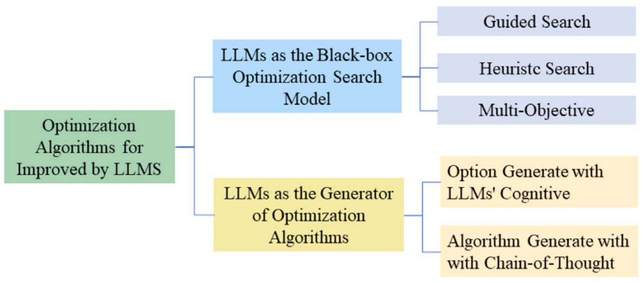

Optimization Algorithms for Improved by LLMS
LLMs as the Black-box Optimization Search Model
Guided Search
Heuristc Search
Multi-Objective
LLMs as the Generator of Optimization Algorithms
Option Generate with LLMs' Cognitive
Algorithm Generate with with Chain-of-Thought

Fig. 1. The split of LLMs assist optimization algorithms.

Algorithms (EAs), capable of discovering diverse solutions to complex real-world problems [67]. With their powerful knowledge storage and generation capabilities, LLMs can support optimization algorithms in problem decomposition, parameter search, and solution generation. As show in Fig. 1, LLMs can be classified into two categories for enhancing optimization algorithms: (1) One category is to use the large model as the search operator of the black-box optimization model, which makes full use of the knowledge storage capacity and experience of LLMs and thus can effectively reduce the input of workforce. (2) The other category of approach is to give full play to the generative capacity of the large model and to make full use of the understanding of the optimization problem by the large model as the input of model and then generate suitable optimization algorithm configurations or generate optimization algorithms for solving specific problems. The use of large models to assist the design of optimization algorithms has achieved preliminary research and widespread attention. How to give full play to the advantages of large models in the design of algorithms and integrate them into the optimization algorithm framework has become the key to the research in this field. A detailed classification of what is the goal of the auxiliary optimization model with LLMs is shown in Table 4.

Yang et al. [68] proposed the Optimization by PROmpting (OPRO), which utilizes natural language descriptions to guide LLMs in searching for solutions for optimization problems. This method is particularly effective for derivative-free optimization scenarios that are common in real-world applications. Technically, OPRO includes previously generated solutions and their values, allowing the LLMs to iteratively improve upon solutions. The framework also intelligently balances benefits and costs, crucial for effective optimization, by adjusting the sampling temperature of the LLMs to encourage both refinement of existing solutions and exploration of new ones. Chen et al. [69] explored cue optimization methods in multi-step tasks to improve task execution efficiency. By constructing a discrete LLMs-based prompt optimization framework, the framework automatically provides suggestions for improvement by integrating human-designed feedback rules and preference alignment. It is noted that while LLMs performs well in single-step tasks, real-world multi-step tasks pose new challenges, such as more complex cueing content, difficulty in evaluating individual step impacts, and the fact that different people may have different task execution preferences. To address these issues, the researchers introduced human feedback, leveraging human expertise in providing input and combining it with a genetic algorithm-style framework to optimize cues. For the current emerging optimization problem of neural architecture search, Zhang et al. [70] designed an automated machine learning (AutoML) system based on large-scale language models (LLMs), AutoML-GPT. AutoML-GPT utilizes a GPT model as a bridge to connect multiple AI models and dynamically uses optimized hyperparameters to train the models. The system automatically generates corresponding prompt paragraphs to search for the optimal model architecture and parameters by dynamically taking inputs from user requests and data cards. It then automatically executes the entire experimental process, from data processing to model architecture, hyperparameter tuning, and prediction of training logs. AhmadiTeshnizi

Table 4
The detailed classification of what is the goal of the auxiliary optimization model with LLMs.

Research field	Category	LLM's goal	Related works
LLMs as the black-box optimization search model	Guided search operator	LLMs combined with natural language descriptions or manual a priori knowledge search for suitable optimization algorithms	[68–71]
	Heuristic search operators	LLMs rely on knowledge storage and problem analysis capabilities to search for suitable optimization algorithms	[72–75]
	Multi-objective optimization	LLMs search for multiple objectives and trade-offs between the importances of the inter-objectives	[76–78]
LLMs as the generator of optimization algorithms	Option generate with the cognitive of LLMs	LLMs analyze the problem and generate suitable optimization algorithm options based on a priori optimization algorithm knowledge	[79–81]
	Algorithm generate with chain-of-thought	Generate optimization algorithms, optimization steps, or code with the help of the chain of thought ability of LLMs to solve complex problems	[82–87]

et al. [71] proposed a large language models (LLMs)-based agent called OptiMUS. OptiMUS is designed to formulate and solve mixed-integer linear programming (MILP) problems from natural language descriptions. It is capable of developing mathematical models, writing and debugging solver code, developing tests, and checking the validity of the generated solutions.

4.1.1. Heuristic search operators

To tackle NP-hard combinatorial optimization problems (COPs) using large language models (LLMs) as Hyper-Heuristics (LHHs), Ye et al. [72] proposed ReEvo, a framework that emulates the reflective design approach of human experts. It leverages the scalable inference capabilities of LLMs, Internet-scale domain knowledge, and powerful evolutionary search strategies. ReEvo operates by generating heuristics through LLMs with minimal human intervention, offering an open-ended heuristic space and the potential for Knowledge and competence beyond that of human experts. The research aims to address the complexity and heterogeneity of COPs by automating the design process of heuristics, which traditionally requires extensive trial and error from domain experts. ReEvo incorporates a dual-level reflection mechanism, where short-term reflections are used to analyze heuristics' relative performance, and long-term reflections are accumulated to guide their evolution. This reflective process allows ReEvo to adapt and improve heuristics over time, leading to smoother fitness landscapes and more effective search results. Zhong et al. [73] introduced a groundbreaking approach to the design of metaheuristic algorithms by utilizing the capabilities of the large language models (LLMs) ChatGPT-3.5. They proposed a animal-inspired metaheuristic algorithm named Zoological Search Optimization (ZSO), which is developed to tackle continuous optimization problems. The ZSO algorithm is designed to mimic the collective behaviors of animals, incorporating two key search operators: the prey–predator interaction operator and the social flocking operator, which together balance exploration and exploitation effectively. A similar approach called Language Model Crossover (LMX) utilizes large pre-trained Language Models (LLMs) to generate new candidate solutions [74]. LMX does this by combining the parent solutions into a prompt and then feeding that prompt into the LLMs to collect offspring from the output. This approach is simple to implement and generates high-quality progeny across various domains, including binary strings, mathematical expressions, English sentences, image generation prompts, and Python code. Liu et al. [75] explore the potential of using Large Language Models (LLMs), such as GPT-4, to generate novel hybrid population intelligence optimization algorithms. The research focuses on using GPT-4 to identify and decompose six population algorithms that perform well in sequential optimization: particle swarm optimization (PSO), cuckoo search (CS), artificial bee colony algorithm (ABC), grey wolf optimizer (GWO), self-organizing migration algorithm (SOMA), and whale optimization algorithm (WOA) by constructing hints to guide the LLMs to search for the optimal from the current population's parent solution. INSTINCT (INSTruction optimization using Neural bandits Coupled with Transformers) [88] is a black-box LLMs cue optimization method. The method employs a novel neural band algorithm, Neural Upper Confidence Bound (NeuralUCB), in place of the Gaussian Process (GP) model in BO. NeuralUCB uses a neural network (NN) as a proxy while retaining the theoretical basis of the trade-off between exploration and exploitation in BO. Theoretical basis for the trade-off between exploration and exploitation. More importantly, NeuralUCB allows for the natural coupling of NN agents with hidden representations learned from pre-trained transformers (i.e., open-source LLMs), significantly improving algorithmic performance.

4.1.2. Multi-objective optimization

Brahmachary et al. [76] introduces an approach to numerical optimization using Large Language Models (LLMs) called Language-Model-Based Evolutionary Optimizer (LEO), which leverages the reasoning capabilities of LLMs to perform zero-shot optimization across a variety of scenarios, including multi-objective and high-dimensional problems. LEO lies in its population-based strategy, which incorporates an elitist framework consisting of separate explore and exploit pools of solutions. This strategy not only harnesses the optimization capabilities of LLMs but also mitigates the risk of getting stuck in local optima. The method is distinct from other auto-regressive, evolutionary, or population-based methods as it uses LLMs to generate new candidate solutions, providing a unique balance between exploration and exploitation. It is shown through a series of test cases that LEO is not only capable of handling single-objective but also of solving multi-objective optimization problems well. Liu et al. [77] leverage the capabilities of large language models (LLMs) to design operators for multi-objective evolutionary algorithms (MOEAs). The research addresses the challenges associated with the manual design of search operators in MOEAs, which often require extensive domain knowledge and can be time-consuming. The authors propose a method for decomposing the multi-objective optimization problem (MOP) into several single-objective subproblems (SOPs). LLMs are employed as search operators for each subproblem through prompt engineering. This allows the LLMs to serve as a black-box search operator in a zero-shot manner without problem-specific training. Bradley et al. [78] introduce a new approach called Quality-Diversity through AI Feedback (QDAIF) that combines evolutionary algorithms and large-scale language models (LLMs) to generate high-quality and diverse candidates for optimization algorithms. The core idea of QDAIF is to use language models to create variants and evaluate the quality and diversity of the candidates. The EA is responsible for maintaining the library of optimization algorithms and replacing the newly generated higher quality and more diverse solutions to the relevant positions in the library based on the evaluation of the LLMs to achieve an iterative optimization search process.QDAIF can find a set of higher quality and more diverse solutions within the search space, which is a successful application of the LLMs in QD problems.

4.2. LLMs as the generator of optimization algorithms

Optimization algorithms usually require the design of suitable optimization schemes based on the specified tasks. Due to the problem-understanding and algorithmic analysis capabilities of LLMs, the design of optimization methods based on LLMs can generate suitable optimization method selection and combination schemes compared to the optimization methods in the pre-LLM era [89,90].

4.2.1. Option generate with the cognitive of LLMs

Zhang et al. [79] explore the use of large language models (LLMs) to generate optimal configurations during Hyperparametric Optimization (HPO). The generation method does not rely on a predefined search space and consists of selecting parameters that can be optimized and specifying bounds for these parameters. Furthermore, the process treats the code of the specified model as hyperparameters to be output by the LLM, which goes beyond the capabilities of existing HPO methods. Liu et al. [80] proposes a system named AgentHPO, which leverages the advanced capabilities of LLMs to streamline the HPO process, traditionally a labor-intensive and complex task that requires significant computational resources and expert knowledge. AgentHPO lies in its unique architecture that incorporates two specialized agents: the Creator and the Executor. The Creator agent interprets task-specific details provided in natural language and generates initial hyperparameters (HPs), emulating the role of a human expert. It utilizes extensive domain knowledge and sophisticated reasoning to propose HP configurations that are expected to yield optimal model performance. Ma et al. [81] explored the effectiveness of Large Language Models (LLMs) as cueing optimizers. They find that LLM optimizers struggle to accurately identify the root cause of errors during reflection and often fail to generate appropriate cues for the target model through a single cueing optimization step, even when semantically valid. Further, they proposed a new paradigm of “automated behavioral optimization” designed to more controllably and directly optimize the behavior of the target model.

4.2.2. Algorithm generate with Chain-of-Thought

Liu et al. [82] proposed a approach called Algorithmic Evolution using Large Language Models (AEL), which aims to automate the generation of efficient algorithms for specific optimization problems.AEL creates and improves algorithms by interacting with Large Language Models (LLMs) within an evolutionary framework, eliminating the need for model training and significantly reducing the need for domain knowledge and expert skills. The constructive algorithms generated by AEL outperform hand-crafted heuristics and algorithms generated directly from LLMs based on the Traveling Provider Problem (TSP) example. Further, they introduce a improved framework for automated algorithm design that combines evolutionary computation and LLMs [83]. By automating algorithm design, combination, and modification, the AEL framework significantly reduces manual effort and eliminates the need for model training. The researchers used the AEL framework to design a Guided Local Search (GLS) algorithm for solving the TSP. The PromptBreeder (PB) system [84] utilizes the Chain-of-Thought Prompting strategy, which can significantly improve the reasoning ability of Large Language Models (LLMs) in different domains. It is a generalized mechanism for self-referential self-improvement of large language models (LLMs). At the heart of the system is self-reference: not only do the task prompts evolve, but the mutation prompts used to generate them also improve over time.PB outperforms existing state-of-the-art prompting strategies, such as Chain-of-Thought and Plan-and-Solve prompts, in several commonly used benchmark tests—and-Solve prompts. Pluhacek et al. [85] identified and decomposed six population algorithms that perform well on sequential optimization problems through LLMs by augmenting the population. Enhanced Swarm Exploration and Exploitation Optimizer(ESEEO) aims to maintain population diversity and effectively balance exploration and exploitation by combining elements of Particle Swarm Optimization (PSO), Cuckoo Search (CS), and Artificial Bee Colony (ABC). Combining Limited Evaluation Population Optimizer (LESO) Designed to solve expensive optimization problems with a limited number of objective function evaluations, LESO combines the features of PSO, Grey Wolf Optimizer (GWO), and ABC to achieve effective exploration and exploitation within a limited number of assessments. LLMs are a challenge in Genetic Programming (GP) approaches. Bradley et al. [86] present an optimized algorithm generation tool for implementing LLMs that converts natural language descriptions into implementation code and automatically repairs program errors. In addition, this paper presents two uses of LLMs as evolutionary operators: difference modeling and LMX crossover. The former is a language model specialized for predicting code discrepancies, and the latter is a technique for generating candidate solutions using multiple parents. Similarly, Hemberg et al. [87] proposed a LLMs-based GP algorithm, called LLMs-GP, for generating optimization algorithm code. Unlike conventional GP algorithms, LLMs-GP harnesses the power of pre-trained pattern matching and sequence complementation capabilities of the LLM. This unique feature allows for the design and implementation of genetic operators, paving the way for more efficient and effective algorithm generation.

5. Optimization algorithms optimize large language models

Large Language Models (LLMs) have exhibited exceptional proficiency in many natural language processing tasks, encompassing text generation and sentiment analysis. Nevertheless, the task of optimizing their performance and efficiency continues to be a crucial obstacle, especially in intricate and unpredictable circumstances. Optimization Algorithms (OA) have emerged as a promising method to improve LLMs. OA utilizes the concepts of natural selection to perform repeated searches inside the parameter space of LLMs [11,67]. It focuses on tasks like prompt engineering, model architecture optimization, hyperparameter setting, and multi-task learning. This holistic strategy seeks to discover optimal solutions for problems that have extensive search spaces, thereby enhancing the effectiveness of LLMs in several fields, such as natural language processing, software engineering, and neural architecture search. In this section, we will delve into the complexities of OA optimization for LLMs, exploring various strategies such as model tuning, prompt tuning and network architecture search to unlock the full potential of these transformative language models. The framework of the optimization algorithm to optimize LLMs is shown in Fig. 2.

5.1. Optimize model tuning

5.1.1. Multi-task learning optimization

Multi-task learning (ML) optimization of large models is an approach that uses optimization methods such as evolutionary algorithms to optimize model architectures for multiple tasks simultaneously. Through multi-task learning, researchers identify optimal architectures for multiple target tasks simultaneously in large pre-trained models [67,95]. The advantage of this approach is the ability to share and interactively learn relevant information between different tasks, thus improving the generalization ability and efficiency of the model. Table 5 summarizes the main optimization model tuning class methods

The optimization strategy of multi-task learning reduces training costs, enhances model performance and improves adaptability across diverse domains and tasks by concurrently identifying optimal model architectures for multiple objectives using evolutionary algorithms [96]. For example, Choong et al. [91] proposed the concept of a diverse set of compact machine learning model sets designed to efficiently address multiple target architectures in large pre-trained models through an evolved multi-task learning paradigm. Baumann et al. [92] present an evolutionary multi-objective approach designed to optimize prompts in large language models (e.g., ChatGPT), demonstrating its effectiveness in crafting prompts in a sentiment analysis task that effectively captures conflicting emotions. Yang et al. [97] treated instruction generation as an evolutionary multi-objective optimization problem, using a large-scale language model to simulate instruction

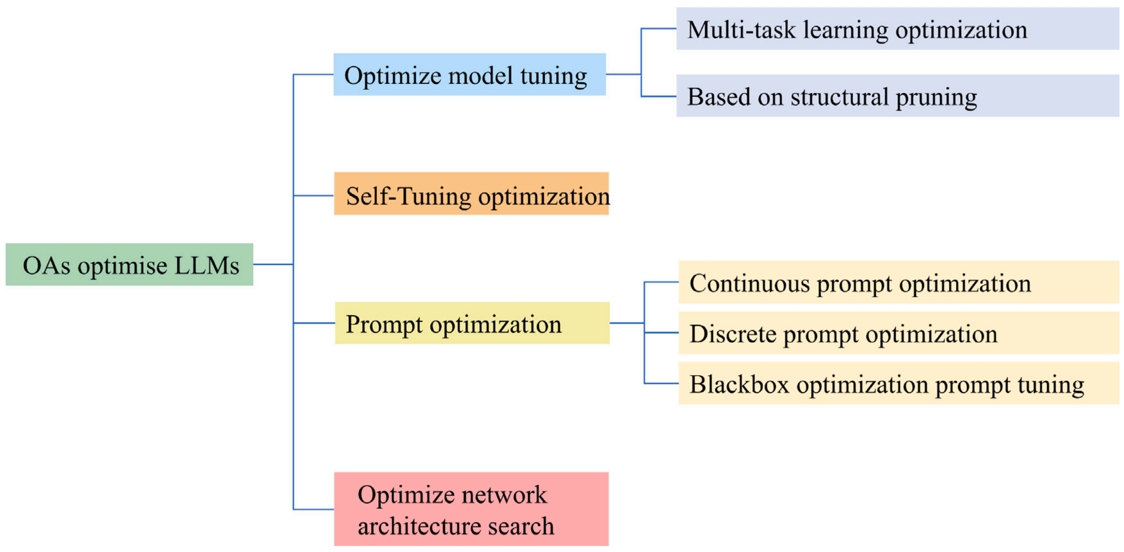

OAs optimise LLMs
Optimize model tuning
Multi-task learning optimization
Based on structural pruning
Self-Tuning optimization
Prompt optimization
Continuous prompt optimization
Discrete prompt optimization
Blackbox optimization prompt tuning
Optimize network architecture search

Fig. 2. OAs methodological framework for optimizing LLMs.

Table 5
Summary of model tuning based on OAs.

Algorithms	Main tasks	Objectives	Using OAs
Multi-task learning [91]	JATs implementation of one-off diverse compact set of sets machine learning models	Multi-tasking and multi-objective optimization	Neuroevolutionary multitasking
EMO-prompts [92]	Generate prompts that lead LLM to produce text with two conflicting emotions	Evolutionary multi-objective	NSGA-II
Structured pruning [93]	Subparts of the network with optimal performance after fine-tuning for model compression	Multi-objective NAS	Weight shared NAS
LLM-pruner [94]	Model compression in a task agnostic manner	Single-objective	Low-rank adaptation

operators in order to improve the quality and diversity of generated instructions.

Meanwhile, Gupta and Bali et al. [98,99] have investigated the use of multi-objective multi-task evolutionary algorithms, such as the MO-MFEA algorithm, in creating task-specific, small-scale models derived from Large Language Models (LLMs) in the field of online search architecture study. These specialized models are created from the LLMs as a general base model and exhibit enhanced performance or more compression in different application fields and neural network designs.

5.1.2. Based on structural pruning

Structural pruning optimizes large language models (LLMs) by selectively removing non-critical coupled structures based on gradient information, effectively reducing model size while preserving functionality and ensuring task-agnosticism. Structural pruning is an essential optimization technique used to enhance pre-trained LLMs for subsequent tasks, such as text categorization and sentiment analysis. Structural pruning, as suggested by Klein [93], seeks to uncover numerous subnetworks of LLMs that achieve a compromise between performance and size, making them easier to use in different real-world applications. This approach employs a multi-objective local search algorithm to identify numerous Pareto-optimal subnetworks effectively. It does this by minimizing evaluation costs via weight sharing. Ma et al. [94] propose the LLM-Pruner approach to optimize large language models by selectively removing non-critical coupling structures based on gradient information to achieve efficient compression while preserving functionality and task agnosticism. Gholami et al. [100] demonstrate that weight pruning can be used as an optimization strategy for the Transfer architecture, proving that judicious pruning can significantly reduce model size without sacrificing performance, thus contributing to bridging the gap between model efficiency and performance.

5.2. Prompt optimization

Prompt tuning, often referred to as optimization for prompt tuning, is a method used to fine-tune large language model (LLMs) prompts. This methodology does not need access to the underlying model parameters and gradients. This strategy is especially beneficial for closed-source models that have limited access. Prompt optimization enhances the efficacy of model creation in fewer or zero-shot scenarios by fine-tuning the input prompts. Optimization algorithms (OAs), renowned for their adaptability and efficiency in situations where the internal workings are unknown, are used to discover prompts that improve job performance just by using the outcomes of LLMs reasoning. Prompt tuning may be categorized into two distinct types: continuous and discrete. Continuous prompt tuning employs continuous optimization algorithms, such as CMAES [101], to improve the quality of embedding prompts [67]. This process involves techniques like partitioning and subspace decomposition to enhance the embedding space. Conversely, discrete prompt tuning uses discrete optimization algorithms to explore the prompt space directly. It utilizes specialized genetic operators to adjust prompts in order to address the problem of combinatorial explosions in the search space. Table 6 summarizes the main methods of optimizing prompts

5.2.1. Continuous prompt optimization

Continuous prompt tuning is used to optimize the performance of LLMs and is often employed to tune the embedding of prompts. The embedding vectors of the prompts are iteratively tuned to maximize the performance of the model on a particular task, thus improving the quality and performance of the model generation [67]. Continuous prompt tuning typically explores different strategies and techniques, such as stochastic embedding, subspace decomposition, and knowledge distillation, in order to improve the embedding quality and the search performance for the optimization of large models. For example, Sun et al. [103] presented BBTv2, an improved version of Black-Box Tuning, which uses a divide-and-conquer gradient-free algorithm to optimize prompts at different layers of pre-trained models, achieving comparable performance under few-shot settings. Fei et al. [105] introduced a gradient-free framework for optimizing continuous textual inversion

Table 6
Summary of optimization prompts.

Algorithms	Main tasks	Types	Using OAs
BBT [102]	Optimizing large pre-trained language models published as a service	Blackbox optimization	Derivative-free optimization, CMAES
BBTv2 [103]	Adjustment of cues by gradient-independent methods	Blackbox optimization	CMAES
BBT-VLMS [104]	Black-box optimization of visual language models	Blackbox optimization	Derivative-free optimization, CMAES
Textual inversion [105]	Optimizing the embedded representation of specific concepts in text	Continuous	Iterative evolutionary strategy, CMAES
Clip-tuning [106]	Enabling optimization of prompts without model weight access rights	Continuous	Derivative-free optimization, CMAES
RGLF [107]	Effective hint tuning for LLMs in resource-constrained black-box API settings	Continuous	CMAES
GrIPS [108]	Prompts for improving large language models	Discrete	Edit-based search, Genetic algorithms
GPS [109]	Automatically search for high-performance prompts to optimize model performance for specific tasks	Discrete	Genetic algorithm
Plum [110]	Optimizing and customizing large pre-trained language models	Discrete	Metaheuristics

in an iterative evolutionary strategy. It accelerates the optimization process with minimal performance loss and compares performance with gradient-based models with variant GPU/CPU platforms. Pryzant et al. [111] proposed Automatic Prompt Optimization (APO) using numerical gradient descent techniques to automatically improve the prompts of Large Language Models (LLMs), obtaining significant performance gains in various NLP tasks and jailbreak detection. Zheng et al. [112] Black-box prompt optimization using subspace learning (BSL) enhances the versatility of prompt optimization across tasks and LLMs by identifying common subspaces through meta-learning, ensuring competitiveness across a variety of downstream tasks.

Recently, some researchers have used techniques such as knowledge distillation, variational reasoning, and federated learning to improve search efficiency, generalization, and security. For example, Shen et al. [107] presented techniques for adapting large pre-trained language models (PLMs) to downstream tasks using only black-box API access, achieving competitive performance with gradient-based methods while also considering predictive uncertainty in prompts. Sun et al. [113] present a set of techniques for improving the efficiency and performance of black-box optimization (BBT-RGB) for tuning large language models without access to gradient and hidden representations, demonstrating its effectiveness in a variety of natural language understanding tasks. Han et al. [114] proposed GDFO, which ensembles gradient descent and derivative-free optimization for optimizing task-specific successive prompts of large pre-trained language models in black-box tuning scenarios, obtaining significant performance gains over previous state-of-the-art approaches. Sun et al. [115] proposed FedBPT for federated black-box prompt tuning, a framework for efficient and privacy-preserving fine-tuning of pre-trained language models through collaborative prompt optimization without access to model parameters, reducing communication and memory costs while maintaining competitive performance. Chai et al. [106] introduced the Clip-Tuning technique to enhance search efficiency and offer more detailed and diverse evaluation feedback throughout the black-box tuning procedure. Clip-tuning differs from utilizing a single random projection matrix to reduce dimensionality in BBT. Instead, it utilizes pre-trained sampling on dropout models during inference. This process generates many subnetworks that act as predictive projections of samples in the original high-dimensional space. The search technique achieves faster convergence to the ideal solution by aggregating rewards from predictions made by several subnetworks.

Continuous prompt tuning is a method for optimizing the performance of large language models where prompts are represented as continuous vectors that exist in a continuous embedding space. These vectors are optimized in a continuous embedding space. The search space is continuous and can be differentiated, allowing the use of gradient-based optimization techniques. Continuous prompt tuning is suitable for black-box optimization scenarios where the internal structure of the model and gradient information are not accessible. However, this approach may require more computational resources due to the complex mathematical operations and gradient calculations involved.

5.2.2. *Discrete prompt optimization*

Discrete prompt optimization is a method for finding optimal prompts in a pre-trained language model, where the prompts are represented as discrete text sequences. These methods typically use genetic algorithms, particle swarm optimization, or other heuristic-based search methods to search in a discrete prompt space to find the optimal prompt sequence [11,67]. Unlike continuous prompt tuning, discrete prompt optimization focuses more on tuning prompts at the level of text sequences and is suitable for tasks based on text sequences, such as text generation or classification. Before the emergence of large language models, researchers have been inclined to investigate the application of optimization methods to enhance the performance of pre-trained language models. For instance, Greedy Teaching Prompt Search (GrIPS) [108] uses a stepwise search approach without gradients, whereas Genetic Prompt Search (GPS) [116] is grounded in the ideas of genetic algorithms. The majority of these research employ evolutionary algorithms (EAs) as the primary search engine, whereas the language model is tasked with generating and assessing potential prompts [109]. Within the discrete prompt space, specialized genetic operators are employed to fine-tune heuristics and directly identify the most optimal prompts. This, in turn, enhances the quality of the model's response to a given task.

Typically, these studies employ optimization algorithms as a search framework, where Large Language Models (LLMs) are used to generate and evaluate prompts. Nevertheless, it is important to acknowledge that this research mainly concentrates on particular rapid engineering situations and has restricted breadth. To fully harness the potential of discrete optimization in black-box prompt optimization, it is necessary to tackle the issue of combinatorial explosion in discrete search spaces. For example, Zhou et al. [117] proposed a simple black-box search method called ClaPS, which achieves state-of-the-art performance on a variety of tasks and LLMs while significantly reducing the search cost by clustering and pruning the search space to focus on key prompting tokens that affect LLMs prediction. Yu et al. [104] proposed a black-box prompt tuning framework for visual-verbal models, which optimizes visual and verbal prompts in an intrinsic parameter subspace through an evolutionary strategy, enabling task-relevant prompt learning without back-propagation. Pan et al. [110] introduced meta-heuristic algorithms as a generic prompt learning method, and demonstrated their effectiveness in black-box prompt learning and Chain-of-Thought prompt tuning by testing six typical methods and were able to discover more understandable prompts, opening up more possibilities for prompt optimization. Lapid et al. [118] proposed a method for attacking large language models using genetic algorithms to reveal the vulnerability of the models to malicious manipulation and to provide a diagnostic tool for assessing and enhancing the consistency of language models with human intentions. Guo et al. [119] presented EvoPrompt, a discrete prompt optimization framework for the automatic optimization of LLMs prompts using evolutionary algorithms. By linking LLMs and EAs, the method achieved significant performance improvements over manually designed prompts and existing automatic prompt generation methods on 31 datasets, demonstrating the potential of combining LLMs and EAs. Pinna et al. [120] present a method for improving the generation of code for large language models using genetic improvement techniques, which significantly improves the quality of the generated code through user-supplied test cases, demonstrating the potential of combining LLMs with evolutionary techniques.

Generally, discrete prompt optimization is a method to optimize the performance of large language models in a black-box environment, which searches for optimal prompt words or phrases in a discrete prompt space through techniques such as genetic algorithms, heuristic search, and clustering pruning without the need for internal gradient information of the model. The advantages include effective performance improvement without relying on the internal information of the model in a black-box environment, and adaptability to tasks with a small number of samples or zero samples, while the disadvantages include the possibility of facing a huge search space, the tendency to fall into local optimums, the sensitivity of hyper-parameters, the limited ability of generalization, the difficulty of interpreting the results, and the high dependence on the choice of evaluation metrics.

5.2.3. Blackbox optimization prompt tuning

Black-box optimization of large models refers to the process of optimizing and tuning large pre-trained models (e.g., large language models) within a black-box optimization framework. Compared to traditional black-box optimization, black-box optimization of large models is more challenging because the complexity of large pre-trained models and a large number of parameters makes the optimization process more complex and time-consuming. In black-box optimization of large models, optimization algorithms typically optimize the performance of pre-trained models by interacting with them and adjusting their inputs or parameters step-by-step without having direct access to the internal structure or parameters of the model.

In recent years, a number of researchers have focused on the application of black-box optimization to large-scale language models (LLMs) and visual-linguistic models, proposing a variety of methods to optimize model performance without accessing the model's internal parameters or gradients. Yu et al. [121] optimize a visual language model using a dialogue-feedback-based approach. Guo et al. [122] introduced the collaborative black-box tuning (CBBT) technique. Sun et al. [102] develop a black-box tuning framework for Language Models as a Service (LMaaS). Diao et al. [123] proposed a black-box discrete prompt learning (BDPL) algorithm. And the work of Yu et al. [104] introduces a black-box prompt tuning framework for visual language models. These studies demonstrate that in black-box scenarios where model weights cannot be directly modified, external prompt learning and optimization are used to effectively improve model performance in image classification, text-to-image generation, and adaptation to different downstream tasks.

5.3. Self-tuning optimization

Compared to the initialization method under the EvoPrompt framework, which relies on hand-prompted optimization. In recent years some researchers have proposed automated prompting methods. Use as a gene operator in EAs to automatically create high-quality prompts for yourself and others. Table 7 summarizes the main Self-Tuning optimization methods. For example, Singh et al. [124] applied an interpretable auto-prompt (iPrompt) to generate a natural language string that explains the data. Fernando et al. [84] propose a self-improving mechanism for PromptBreder that evolves and adapts cues for different domains, outperforming existing strategies on arithmetic, common-sense reasoning, and hate speech classification tasks. Pryzant et al. [125] proposed a simple and non-parametric solution, Automated Prompt Optimization (APO), which automatically performs fast improvement prompts by using techniques inspired by numerical gradient descent. Li et al. [126] proposed SPELL, a black-box evolutionary algorithm that uses a large language model to automatically optimize text style cues, demonstrating rapid improvements for a variety of text tasks.

Furthermore, LLMs can serve as a flexible prompt selector for jobs that are not inside the domain it was trained on. Self-tuning can operate inside a versatile language domain without being dependent on parameter updates [67]. For example, Zhang et al. [127] proposed Auto-Instruct, an approach that utilizes the generative power of LLMs to automatically improve the quality of instructions for a variety of tasks, going beyond manually written instructions and existing baselines in a variety of out-of-domain tasks, with significant generalizability to other LLMs.

5.4. Optimize network architecture search

Prompt-based optimization tools improve the quality of model output by optimizing the input format. Another approach known as LLMs Network Architecture Search (NAS) focuses on directly optimizing the architecture of LLMs, and in the context of Large Language Models (LLMs), NAS can take a different form by optimizing the architecture of the model directly rather than by tuning the parameters of the model. Table 8 summarizes the main optimized network architecture search methods

As the complexity of neural network models increases, manually designing efficient network architectures becomes time-consuming and challenging. NAS eases the burden on researchers by automating the design process, allowing efficient exploration of the vast search space to discover more efficient, generalized and less resource-intensive model architectures [11,128,129]. Previously NAS was optimized by simulating the process of natural selection. It involves the steps of randomly generating an initial population, selection, crossover (or recombination, as it is called), and mutation until termination conditions are met [130]. With the development of deep learning techniques and the increase of arithmetic power, the NAS field is also exploring new optimization strategies to optimize large models. With the increase of computational resources and the proposal of new algorithms, the efficiency and effectiveness of NAS have been significantly improved. Nasir et al. [131] proposed a new NAS algorithm that effectively combines the advantages of LLMs and Quality Diversity (QD) algorithms to automate the search and discovery of high-performance neural network architectures. So et al. [132] proposed an evolutionary Transformer discovered through evolutionary architectural search in multilingual tasks superior Transformer that achieves better performance with fewer parameters and maintains high quality even at smaller sizes.

More sophisticated and effective search strategies have been proposed by researchers in recent years to improve the performance of large models. For example, Gao et al. [133] proposed an automatic method (AutoBERT-Zero) for discovering the backbone structure of a general-purpose language models (LLMs) using a well-designed search space and an operation-first evolutionary strategy, as well as a two-branch weight-sharing training strategy, to improve search efficiency and performance. Ganesan et al. [134] perform task-independent pre-training of BERT models while generating differently shaped sub-networks by varying the hidden dimensions in the Transformer layer. Rather than optimizing for a specific task, it generates a series of different-sized models by varying the hidden dimensions of the network, which can be fine-tuned for various downstream tasks. Yin et al. [135] proposed the use of one-shot Neural Architecture Search (one-shot NAS) to automatically search for architectural hyperparameters. A large SuperPLM is obtained through one-shot learning, which can be used as a proxy for all potential sub-architectures. An evolutionary algorithm is also used to search for the best architectures on the SuperPLM, and then the corresponding sub-models are extracted based on these architectures and further trained. Javaheripi et al. [136] proposed a no-training Neural Architecture Search (NAS) algorithm for finding Transformer architectures that have an optimal

Table 7
Summary of self-tuning optimization.

Algorithms	Main tasks	Objectives	Using OAs
SPELL [126]	Combining evolutionary algorithms and text generation capabilities of LLMs to optimize prompts for LLMs in a black box environment	Classification accuracy	Variants based on evolutionary algorithms
Promptbreeder [84]	Improves LLMs performance on specific tasks by automating the cue search and optimization process without the need to manually engineer prompts	Performance score	Genetic algorithm
Interpretable autoprompting [124]	Iteratively use LLM to generate explanations and reorder them based on their performance when used as prompts	Accuracy and interpretability	Iterative local search algorithm
APO [125]	Automatically improves prompts to reduce the heavy trial and error required to write prompts manually	Performance of initial prompts	Textual gradient descent
Auto-Instruct [127]	Automatic generation and optimization of instructions for LLMs	Classification accuracy	Metaheuristic algorithms

Table 8
Summary of OAs-based network architecture for LLMs.

Algorithms	Main tasks	Objectives	Using OAs
Autobert-zero [133]	Automatic exploration of new self-attentive structures and overall efficient pre-trained language model backbone architecture	Classification performance	Operation-priority NAS
AutoTinyBERT [135]	Automatic hyperparameter optimization for efficient compression of pre-trained language models	Classification performance	One-shot NAS
Llmatic [131]	Discover diverse and powerful neural network architectures	Network architecture	Quality-diversity optimization
SuperShaper [134]	Discovering networks with effective trade-offs between accuracy and model size	Classification accuracy	Evolutionary algorithm
EvoPrompting [6]	Exploring the use of LLMs as generalized adaptive variation and crossover operators for NAS algorithms	Classification accuracy	Evolutionary algorithm

balance between task performance (perplexity) and hardware constraints (e.g., peak memory usage and latency). Zhou et al. [137] proposed a Transformer architecture search method called T-Razor, which uses zero-cost agent-guided evolution to improve the search efficiency and evaluates and ranks Transformers by introducing metrics such as synaptic diversity and synaptic saliency to efficiently find optimized architectures in the Transformer search space. Klein et al. cites klein2023structural proposed Neural Architecture Search (NAS) based on weight sharing as a structural pruning method for finding the optimal balance between optimization efficiency and generalization performance to achieve compression of large language models (LLMs) in order to reduce the model size and inference latency.

Overall, the main advantage of NAS for optimizing large models is its ability to automate the exploration and discovery of efficient network architectures for specific tasks, significantly improving model performance while reducing manual design and tuning efforts. With intelligent search strategies, NAS helps save computational resources and time. However, this approach also faces challenges, including the large search space, the possibility of falling into local optimal solutions, and the large amount of computational resources required in the initial search and training phases. In addition, the selection and tuning of optimization algorithms require expertise, and the generalization ability of the network architecture obtained from the search still needs further validation. Future research may focus on improving the search efficiency, reducing the computational cost, and enhancing the generalizability and adaptability of the model.

6. Application of LLMs-based optimization algorithms

As show in Fig. 3, optimization algorithms are pivotal in various applications, broadly categorized into software programming, neural architecture search and content generation. LLM-based optimization algorithms are becoming increasingly important in artificial intelligence, especially in machine learning. They are used for software programming and neural architecture search to help design efficient network architectures. Furthermore, these algorithms are employed as innovative tools in content generation, optimizing the creation process to produce relevant and engaging content. This bifurcation in application highlights the versatility and evolving role of optimization algorithms in addressing both conventional challenges and pioneering technological advancements.

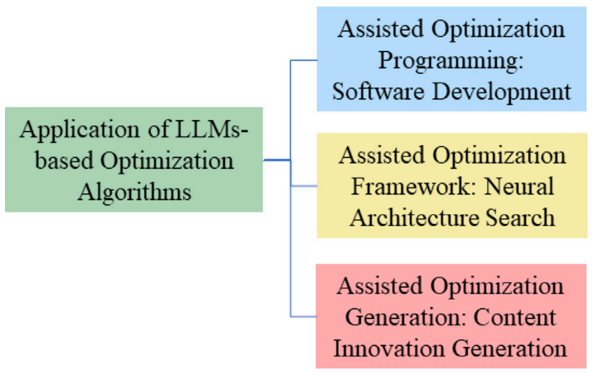

Application of LLMs-based Optimization Algorithms
Assisted Optimization Programming: Software Development
Assisted Optimization Framework: Neural Architecture Search
Assisted Optimization Generation: Content Innovation Generation

Fig. 3. The split of LLMs assist optimization algorithms.

6.1. *Assisted optimization programming: Software programming*

In the wave of artificial intelligence, optimization algorithms based on LLMs have gradually become an important research area to promote code generation and software development. With many parameters and deep learning capabilities, LLMs have shown powerful capabilities in many fields, such as natural language processing, image recognition, etc. Especially in the process of software development, the application of large models can improve the efficiency of code generation and further enhance the model performance through optimization algorithms. By automatically generating code for training models, non-professionals can also easily train efficient machine learning models, greatly reducing the technical threshold and expanding the audience for machine learning technology. Meanwhile, code integration practices in model development, such as static code integration and dynamic code integration, also play a key role in improving the efficiency and quality of software development.

Weyssow et al. [140] explore using Large Language Models (LLMs) for code generation tasks, focusing on Parameter-Efficient Fine-Tuning (PEFT) optimization techniques. The methodology aims to optimize the fine-tuning process of LLMs by updating a small portion of the model's parameters instead of all of them using the PEFT technique to achieve efficient fine-tuning in resource-constrained environments.

Table 9
The detailed classification of what is the goal of the auxiliary optimization model with LLMs.

Fields of application	Category	LLM's goal	Related works
Assisted optimization programming	Code generation and optimization	LLMs facilitate automatic code generation, reducing human input in the programming process and increasing programming efficiency	[138–141]
	Security and testing	Evaluates the security risks of code generated by LLMs and also generates adversarial natural language instructions that would direct LLMs to produce functionally correct but vulnerability-laden code	[120,142–144]
Neural architecture search via assisted optimization framework	NAS algorithms based on LLMs	Generate new architectural variants and discover diverse and robust solutions using LLMs	[131,145,146]
	Evolutionary optimization and performance prediction methods	Enhancing contextual prompts and architectural design with evolutionary search and soft prompt tuning methods	[6,147]
Optimized content innovation with LLMs	Content innovation and generation	Improving the quality and diversity of content innovation and promoting innovation in science, technology and the arts	[148–151]
	Scientific discovery and drug analysis	Providing users with intelligent tools for analyzing drug compounds that mimic human behavior and improve the efficiency and accuracy of scientific research	[152–154]

Cassano et al. [143] present a system called MultiPL-E, which is a system for translating code generation benchmark tests from Python to other programming languages. Further, Pinna et al. [120] point out the application of automatic code generation based on problem descriptions and that even the most efficient LLMs often fail to generate correct code. Therefore, to address the question of how to enhance code generation based on Large Language Models (LLMs) through a Genetic Improvement (GI) approach, an evolutionary algorithm-based approach is proposed that uses Genetic Improvement (GI) to improve LLMs-generated code using a collection of user-supplied test cases. Program synthesis (PS), a form of automation, aims to reduce the time and effort required for software development while improving code quality. Although Genetic Programming (GP) is a competing approach to solving the program synthesis problem, it has limitations in evolving syntactically correct and semantically meaningful programs. Tao et al. [139] used a combination of Generative Pre-trained Transformers (GPTs) and Grammar-Guided Genetic Programming (G3P) to solve the program synthesis problem. GPTs) and Grammar-Guided Genetic Programming (G3P) to address the program synthesis problem. The OpenAI team has developed a language model called CodeX [138], which has been fine-tuned using publicly available code on GitHub and investigated for its ability to write Python code. A special production version of Codex is GitHub Copilot, a programming aid. Brownlee et al. [141] explored how large language models (LLMs) can be applied to variation operations in Genetic Improvement (GI) to improve the efficiency of the search process.GI is a search-based technique used to improve the non-functional attributes, such as execution time, and functional attributes, such as fixing defects, of existing software. Ji et al. [144] specifically presents a research overview of the assessment and interpretation of code generation capabilities based on large language models (LLMs), including two main phases: data collection and analysis. In the data collection phase, the prompts' features are quantified by extracting their linguistic features and performance metrics from the generated code. In the analysis phase, causal diagrams are constructed using causal discovery algorithms and further analyzed to identify principles of hint design.

Codex [138] was fine-tuned using publicly available GitHub code to enhance its Python coding capabilities. The method evaluated Codex using a new test set, HumanEval, which assesses the functional correctness of programs synthesized from documentation strings. In this dataset, Codex successfully solved 28.8% of the problems, compared to GPT-3, which solved none, and GPT-J, which solved 11.4%. CODAMOSA [142] integrates a pre-trained Codex with Search-Based Software Testing (SBST) to enhance test case code coverage. SBST generates high-coverage test cases by combining test case generation with mutation for programs under test. However, SBST may face stagnant coverage, meaning it struggles to produce new test cases that increase coverage. When SBST's coverage improvement stagnates, the CODAMOSA algorithm aids in relocating to more advantageous search space areas by using Codex to generate example test cases for functions with lower coverage. Wu et al. [155] highlight the advances in code generation by large-scale language models (LLMs) and the associated security risks, particularly the critical vulnerabilities in the generated code. Although some LLM providers have sought to mitigate these issues through human guidance, their efforts have yet to yield robust and reliable code LLMs in practical applications. They introduce the DeceptPrompt algorithm, designed to generate adversarial natural language instructions that prompt code LLMs to produce functionally correct yet vulnerable code. DeceptPrompt employs a systematic, evolution-based algorithm with a fine-grained lossy approach. The algorithm uniquely excels at identifying natural prefixes/suffixes with benign, non-directional semantics and effectively induces code LLMs to generate vulnerable code. This feature enables researchers to conduct near-worst-case red-team tests on these LLMs in real-world scenarios through natural language.

In summary, in software programming, large language models (LLMs) enhance code generation efficiency via optimization algorithms and reduce the complexity of machine learning technology. This simplification allows non-experts to train efficient models, thereby broadening the reach of machine learning.

6.2. Assisted optimization framework: Neural architecture search

Neural Architecture Search (NAS), an important technique for the automatic design of neural networks, is undergoing a transformation driven by combining LLMs and optimization algorithms. With their massive parameters and deep learning capabilities, LLMs show unprecedented potential in handling complex tasks. At the same time, the combination of well-designed optimization algorithms can further accelerate the Neural Architecture Search process, improving the search efficiency and the performance of the resulting models. With the continuous application of big models in NAS, we see their great potential in the automatic search and optimization of neural network structures, which not only greatly saves labor costs but also improves the innovation and diversity of model design.

Nasir et al. [131] present LLMatic, a large model-based Neural Architecture Search (NAS) algorithm that uses two QD archives to search for competitive networks, which combines the code generation capabilities of Large Language Models (LLMs) with the diversity and robustness of Quality Diversity (QD) algorithms. LLMatic utilizes the LLMs to generate new architectural variants and combines the QD algorithms (especially the MAP-Elites algorithm) to discover diverse and robust solutions. Chen et al. [6] found that an approach combining evolutionary prompt engineering and soft prompt tuning, EvoPrompting, consistently discovers diverse and high-performance models. A method for creating and curating data using evolutionary search to improve in-context prompting examples for LM is presented. While focused on neural architecture design tasks, this approach is equally applicable to

LM tasks that rely on in-context learning (ICL) or cue tuning. Jawahar et al. [147] build new uses for Performance Predictors (PP) by using Large Language Models (LLMs) that predict the performance of specific Deep Neural Network (DNN) architectures on downstream tasks. A hybrid search algorithm (HS-NAS) is proposed, which uses LLM-Distill-PP in the initial phase of the search and a baseline predictor for the remainder of the search.HS-NAS reduces the search time by about 50% with performance comparable to that of the SOTA NAS and sometimes improves latency, GFLOPs, and model size. Jawahar et al. [147] introduced LLM-PP, a precise performance predictor developed using LLM for few-shot prompting. It achieves a mean absolute error (MAE) comparable to the state of the art (SOTA). LLMDistill-PP, developed as a more cost-effective predictor, caters to applications like Neural Architecture Search (NAS) that require numerous predictions. Additionally, the new HS-NAS algorithm is introduced. It leverages the strengths of LLMDistill-PP and the state-of-the-art performance estimator, reducing NAS search times by half and identifying more efficient architectures.

Zheng et al. [145] explored the potential of GPT-4 models for the Neural Architecture Search (NAS) task of designing effective neural network architectures. At the same time, they propose an approach called GPT-4 Enhanced Neural archItectUre Search (GENIUS), which leverages the generative power of GPT-4 as a black-box optimizer to navigate the architectural search space quickly, identify promising candidate architectures, and iteratively refine these candidate architectures to improve performance. EvoPrompting [6] employs advanced Language Models (LMs) for code-level Neural Architecture Search (NAS). This approach integrates evolutionary prompt engineering with soft-prompt tuning. It aims to iteratively refine contextual prompts and enhance prompt tuning on LMs, thereby boosting their capacity to generate innovative and diverse solutions for complex reasoning tasks. Radford et al. [156] describe a method to enhance Natural Language Understanding (NLU) using Generative Pre-Training. They show that this approach significantly boosts performance across various NLU tasks by initially pre-training a language model on a vast corpus of unlabeled text, then applying supervised fine-tuning for particular tasks. Chowdhery et al. [146] proposed PaLM (Pathways Language Model), a large-scale language model, PaLM has demonstrated excellent performance on a variety of Natural Language Processing (NLP) tasks, and PaLM also has very good performance on network structure design, structure search.

In summary, integrating LLMs with optimization algorithms in neural network architecture search (NAS) enhances search efficiency, fosters innovation, diversifies model designs, and opens new avenues for the automated design of complex neural architectures.

6.3. Assisted optimization generation: Content innovation generation

Innovative content generation has become a key driver for developing media, entertainment, arts, and scientific discovery. Applying big artificial intelligence models combined with optimization algorithms is increasingly important in this process. In summary, using optimization algorithms based on large models in innovative content generation is not only about the innovation and diversity of content but also promotes and facilitates scientific and technological innovation development.

Xiao et al. [148] proposed a pattern-centric text generation framework, PatternGPT, to address the error-prone nature of Large Language Models (LLMs) and the inability to use external knowledge in text generation tasks directly. The framework uses algorithms to search for or generate high-quality patterns based on judgmental criteria. It leverages the pattern extraction capabilities of LLMs to develop a diverse set of structured and formalized patterns, which can help to bring in external knowledge for computation. Chen et al. [149] enhance the performance of Large Language Models (LLMs) in language generation tasks through Model-Adaptive Prompt Optimization (MAPO), a prompt optimization method that can be widely applied to various downstream generation tasks. Similarly, they propose a new paradigm for news summary generation that uses Large Language Models (LLMs) to improve the quality of news summary generation through evolutionary fine-tuning [150]. The method uses LLM to extract multiple structured event patterns from news passages, evolves a population of event patterns via a genetic algorithm, and selects the most adapted event patterns to input into LLM to generate news summaries. PanGu Drug Model [153] is a graph-to-sequence asymmetric conditional variational autoencoder designed to improve molecular property representation and performance in drug discovery tasks. The model is inspired by conversions between molecular formulas and structural formulae in the chemistry classroom and can appropriately characterize molecules from both representations. Liang et al. [152] presented a prototype of a DrugChat system designed to provide ChatGPT-like capabilities for drug compound analysis.DrugChat, by combining graph neural networks (GNNs), large language models (LLMs), and adapters, enables users to upload molecular maps of compounds and ask various questions during multiple rounds of interaction. Diagrams and ask different questions in numerous rounds of interaction, which the system then answers. To break the bottleneck of literate graph technology, Berger et al. [151] proposes the framework of StableYolo, which aims to optimize the image generation quality of large language models (LLMs) by applying evolutionary computation to the Stable Diffusion model while adjusting the prompts and model parameters. The core idea of StableYolo is to improve the image generation quality of photo-realistic styles by combining visual evaluation with multi-objective search. The core concept of StableYolo is to enhance the quality of image generation in photo-realistic style by combining visual evaluation with multi-objective search. The system uses the confidence estimate of the Yolo model as a fitness function and searches for the optimal combination of cue words and model parameters using a Genetic Algorithm (GA).

To explore additional research related to LLMs, including cognitive functions of LLMs, behavior and learning in game-theoretic environments, and Big Five personality traits, Suzuki et al. [157] propose a model for the evolution of personality traits based on Large Language Models (LLMs), specifically those related to cooperative behavior. The approach demonstrates how LLMs can enhance the study of human behavioral evolution and is based on evolutionary game theory by using an evolutionary model that assumes that human behavioral choices in game-theoretic situations can be simulated by providing LLMs with high-level psychological and cognitive trait descriptions. De et al. [158] explored the phenomenon of the self-organized formation of scale-free networks in social interactions between large language models (LLMs). Scale-free networks are a typical emergent behavior in complex systems, especially in online social media, where users can follow each other and form social networks with specific structural features. Lu et al. [154] propose a novel learning framework, SELF (Self-Evolution with Language Feedback), which aims to continuously enable large-scale language models (LLMs) to improve themselves through self-feedback and self-improvement. The SELF framework is inspired by the human self-driven learning process, which consists of an initial attempt, reflective feedback, and The SELF framework is inspired by the human self-driven learning process, which involves a cycle of initial attempts, reflective feedback, and behavioral improvement to improve the model's capabilities. The ELF framework also enables smaller LLMs to improve themselves, which can be reversed to facilitate the development of larger predictive models.

In summary, the proposed systems and frameworks, including DrugChat and SELF, illustrate the development of personalized, intelligent tools for analyzing drug compounds, generating news summaries, and mimicking human behaviors. These tools continuously improve their performance through self-learning and feedback mechanisms, enhancing efficiency and accuracy in related fields. The detailed classification of what is the goal of the auxiliary optimization model with LLMs is shown in Table 9.

7. Future outlook and research trends

In the previous sections, we have examined recent advances in the fields of long-term memory models (LLMs) and optimization algorithms (OAs). Nonetheless, there are still many challenges and unresolved issues between these two fields. Therefore, the aim of this section is to explore directions for future research in order to provide scholars with the opportunity to explore new areas beyond the boundaries of current knowledge, to ask new research questions, and to reinvigorate the field.

Theoretical Foundations and Methodologies. Experimental studies have confirmed the effectiveness of combining large-scale language models (LLMs) with OAs in solving small-scale problems [74,75]. However, the motivation for their interaction has not yet been clarified. To further promote the performance of algorithms, we need to deeply explore the mechanism of mutual reinforcement between LLMs and OAs in theoretical studies and analyze their complementary advantages and potential problems in practical applications in detail through large-scale empirical studies. In addition, it is crucial to conduct in-depth theoretical analyses of algorithms combining LLMs and OAs, which includes evaluating their convergence, time complexity and space complexity. Also, investigating the impact of algorithmic parameter settings on performance, as well as performance guarantees or theoretical limitations of the algorithms on different problem types, are key steps in advancing the algorithms. Further, exploring optimization theory [95], such as clarifying the definition and characterization of the objective function, dealing with constraints, and analyzing the feasible solution space of a problem, will provide a solid theoretical foundation for the design and application of algorithms to achieve better algorithmic performance in solving more complex problems.

Automated Intelligent Optimization. In the optimization context, large language models (LLMs) show significant potential, especially in enhancing the automation and intelligence of optimization algorithms (OAs). Learning from multimodal data during the pre-training phase allows LLMs to understand and generate cross-modal content [159]. This provides a new search and mutation strategy for OAs when performing cross-modal operations. This capability of LLMs can facilitate OAs in achieving a more efficient global search in multimodal optimization problems. At the same time, as the technology of LLMs continues to advance, it is expected that they will drive the performance of OAs in modeling complex evolutionary mechanisms, especially when dealing with optimization problems with large-scale search spaces [160]. However, current research has yet to explore the potential of LLMs in evolutionary optimization, and there remain challenges, such as how to combine LLMs and OAs better and how to handle complex search spaces.

In addition, the pre-training of LLMs on large amounts of textual data embeds them with rich domain knowledge, which provides a robust knowledge base for OAs.LLMs can assist OAs in better integrating domain-specific knowledge in the optimization process, thus improving the efficiency of optimization and the quality of solutions. For example, LLMs can generate high-quality initial solutions, improve problem formulation, and provide solution coding and definition of solution spaces. In addition, LLMs can provide guiding principles for algorithm design, enabling EAs to handle complex optimization problems such as multi-objective, discrete and dynamic more effectively [161]. With the rapid development of LLMs technology, they are expected to play an even more critical role in the future evolutionary optimization field, driving the field toward higher levels of automation and intelligence.

Robustness and Prompt Engineering. Utilizing optimization techniques is a crucial method for improving the capabilities of LLMs in engineering applications. Common approaches involve utilizing LLMs as optimization operators within EIA frameworks to consistently produce fresh prompt. This technique has consistently shown efficacy and superiority in numerous investigations. Nevertheless, certain obstacles persist. Firstly, it is crucial to pay close attention to the initialization of the optimization process as it will have a substantial impact on the outcomes [6,126]. It is crucial to have cue templates that are generic and customizable in order to provide accurate and valid prompts. Random initialization may not be capable of using existing information, and manual seeding may add bias. In addition, when confronted with issues that contain a significant amount of previous knowledge, the range of possible prompts to consider increases exponentially as the length of the cue and the size of the vocabulary grow. This can result in over fitting or becoming trapped in local optimal solutions. Furthermore, these approaches lack stability and strongly depend on the capabilities of the LLM, rendering them susceptible to stochasticity [126]. If the LLM lacks the ability to comprehend and efficiently employ the cues, it may undermine the effectiveness of the approach.

Further research should strive to tackle these obstacles by creating more resilient techniques. For instance, in the context of initialization, the technique of multisource seeding can be investigated to automatically improve the size and quality of the initial population utilizing LLM. When dealing with intricate search spaces, it is essential to develop efficient optimization algorithms. This may involve combining more comprehensive sets of optimization operators, using the advantages of different evolutionary algorithms, and utilizing adaptive optimization techniques.

Generality and Architecture Search. The combined efforts of Large Language Models (LLMs) and OAs have accelerated progress in the field of code generation, leading to notable improvements in downstream applications such as software engineering and OAs design. An commonly used method in this collaboration involves employing LLMs to create large training datasets, and then refining the LLMs using reinforcement learning approaches [162,163]. Nevertheless, this approach has challenges with the variety and quantity of training data, which could result in a failure to cover all possible scenarios. An alternate approach involves utilizing the strong code generation capabilities of LLMs in combination with the powerful search architecture of OAs to continually improve the code generation process. However, this method has difficulties when it comes to generating code for sophisticated algorithmic logic that may require the combined work of numerous code snippets. In order to overcome these obstacles, it is possible to develop a modular strategy that breaks down large activities into smaller, more manageable sub-tasks. An interactive interface might be added to allow users to clearly define the breakdown of tasks [163]. This would enable LLMs and OAs to generate code for each sub-task in a coordinated manner.

Neural Architecture Search (NAS) is an important application scenario that arises from the combination of LLMs and OAs. Although LLMs have shown remarkable effectiveness in other tasks, they have not been specifically designed for NAS [6]. The performance of current LLM models varies significantly when used for NAS tasks, and there is a clear difference between LLM-based approaches and conventional NAS methods in terms of their application area and ability to generalize [164]. In order to enhance the overall effectiveness of LLMs and EAs in NAS projects, a comprehensive strategy could be implemented. This involves assessing the effectiveness of various LLM models in NAS tasks, enhancing LLM's NAS skills by incorporating more training data, optimizing the structure of LLMs during the fine-tuning stage, and investigating the utilization of past search knowledge to speed up future searches and provide clearly defined search spaces for LLMs.

Interdisciplinary Applications and Innovations. The incorporation of Large Language Models (LLMs) with optimization algorithms (OAs) shows potential in several interdisciplinary domains, providing a powerful synergy to stimulate innovation and improve performance in intricate jobs.

In the realm of computational creativity and generative design, LLMs are adept at generating creative content, such as artwork, music, and literary pieces. The collaboration with OA brings methods of variation and selection, which can promote creative diversity and ignite innovation. This collaborative approach can result in the production of unique and groundbreaking artistic and design works, thereby promoting innovation and fostering the growth of creativity. Within the domain of robotics, intelligent and adaptable robot systems can be produced through the collaboration of OAs, which have the ability to refine control strategies and action sequences, and LLMs, which are capable of generating instructive dialogues and task-oriented directives [11]. These systems have enhanced capabilities to adjust to various tasks and participate in complex interactions with humans, enhancing collaboration between humans and robots and establishing the foundation for advanced robotic applications.

Moreover, in the field of drug design, the ability of LLMs to produce new chemical structures, combined with the multi-objective optimization capabilities of OAs, can accelerate the process of discovering new drugs. This comprehensive technique has the capability to recognize drug candidates with higher potential, so decreasing the time and expenses linked to conventional trial-and-error procedures and promoting progress in pharmaceutical research and development. The combination of LLMs and OAs offers a versatile instrument that has the ability to transform various fields by offering inventive solutions and improving efficiency in problem-solving. As research explores the joint capabilities of LLMs and OAs, it is expected that more significant advancements and innovative uses will arise, revolutionizing industries and expanding the limits of human accomplishment.

8. Conclusion

The practical limitations, ethical considerations, and potential biases of combining large models (LLMs) with optimization algorithms in review articles are manifold. Firstly, from the perspective of practical limitations, LLMs require large amounts of computational resources, which may limit their application in resource-constrained environments. At the same time, they rely on a large amount of data for training, which not only poses data acquisition and processing challenges but may also raise privacy protection concerns. In addition, the decision-making process of LLMs often needs more transparency, which may pose a problem in applications that require a high degree of interpretability. Their ability to generalize and robustness are also key challenges, especially in the face of adversarial attacks and unseen data. When we delve into the ethical considerations of training and applying LLMs, we enter a realm of significant concern. The handling of large amounts of personal data demands stringent data privacy protection. Algorithmic fairness, a crucial topic, is at risk as LLMs may inherit and amplify biases in the training data, potentially leading to unjust outcomes. Furthermore, as the decision-making process becomes automated, the complexity of determining responsibility attribution and the opacity of LLMs may compromise users' trust in the system. Potential bias is another key issue.LLMs may exhibit biases in training data, algorithm design, cue engineering, performance evaluation, and application scenarios. These biases may stem from imbalances in the dataset or cultural or linguistic factors that affect the model output and performance.

Addressing these limitations and challenges is not a one-time task but a commitment to continuous research and improvements in algorithm design, data management, transparency enhancement, ethical review, and user education. It is not enough to optimize at the technical level; we must also adhere to ethical principles and proactively prevent potential risks. This ongoing effort, which requires the collective contribution of researchers, scholars, and professionals, is crucial to ensure that the combination of LLMs and optimization algorithms can be developed within an ethical and socially responsible framework.

We explore the progress and potential of combining Large Language Models (LLMs) with Optimization Algorithms (OAs), particularly for enhanced decision-making in dynamic environments. LLMs possess extensive domain knowledge that contributes to strategic decision-making in intelligent modeling and optimization, while Optimization Algorithms improve the architectures and quality of outputs from LLMs. This synergy provides new ways to advance general-purpose Artificial Intelligence (AI), addressing both the computational challenges of complex problems and the application of LLMs in real-world scenarios. Combining Large Language Models (LLMs) and Optimization Algorithms (OAs) is crucial for solving large-scale, high-dimensional, and dynamically changing optimization problems. Future research should deeply explore the mechanisms by which LLMs and OAs interact and analyze in detail their complementary advantages and potential issues in real-world applications through large-scale empirical studies. Integrating LLMs and OAs has demonstrated its potential to facilitate innovation and enhance performance in solving complex problems across various fields, including computational creativity, generative design, robotics, and drug design. With the rapid advancements in LLM technology, they are expected to play an even more critical role in evolutionary optimization, driving the field toward higher levels of automation and intelligence. By systematically analyzing the research progress in combining LLMs with OAs, this paper provides valuable references and insights for future research directions. With continuous technological progress, the synergy between LLMs and OAs will undoubtedly yield more innovations and breakthroughs in artificial intelligence.

CRediT authorship contribution statement

Sen Huang: Writing – review & editing, Writing – original draft. **Kaixiang Yang:** Investigation. **Sheng Qi:** Methodology. **Rui Wang:** Project administration.

Declaration of competing interest

We authors declare that they have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript.

Data availability

No data was used for the research described in the article.

References

[1] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092.
[2] M. Abdel-Basset, W. Ding, D. El-Shahat, A hybrid Harris Hawks optimization algorithm with simulated annealing for feature selection, Artif. Intell. Rev. 54 (1) (2021) 593–637.
[3] N. Singh, S. Singh, E.H. Houssein, Hybridizing salp swarm algorithm with particle swarm optimization algorithm for recent optimization functions, Evol. Intell. 15 (1) (2022) 23–56.
[4] M. Lv, J. Wang, S. Wang, J. Gao, H. Guo, Developing a hybrid system for stock selection and portfolio optimization with many-objective optimization based on deep learning and improved NSGA-III, Inform. Sci. (2024) 120549.
[5] Y. Massim, F. Yalaoui, E. Châtelet, A. Yalaoui, A. Zeblah, Efficient immune algorithm for optimal allocations in series-parallel continuous manufacturing systems, J. Intell. Manuf. 23 (2012) 1603–1619.
[6] A. Chen, D. Dohan, D. So, Evoprompting: Language models for code-level neural architecture search, Adv. Neural Inf. Process. Syst. 36 (2024).
[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Adv. Neural Inf. Process. Syst. 33 (2020) 1877–1901.
[8] E. Mirsadeghi, S. Khodayifar, Hybridizing particle swarm optimization with simulated annealing and differential evolution, Cluster Comput. 24 (2021) 1135–1163.
[9] W.X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al., A survey of large language models, 2023, arXiv preprint arXiv:2303.18223.

[10] V. Pallagani, B.C. Muppasani, K. Roy, F. Fabiano, A. Loreggia, K. Murugesan, B. Srivastava, F. Rossi, L. Horesh, A. Sheth, On the prospects of incorporating large language models (llms) in automated planning and scheduling (aps), in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 34, 2024, pp. 432–444.

[11] X. Wu, S.-h. Wu, J. Wu, L. Feng, K.C. Tan, Evolutionary computation in the era of large language model: Survey and roadmap, 2024, arXiv preprint arXiv:2401.10034.

[12] G. Lupyan, The centrality of language in human cognition, Lang. Learn. 66 (3) (2016) 516–553.

[13] A.M. Turing, Computing machinery and intelligence (1950), 2021.

[14] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al., A survey on evaluation of large language models, ACM Trans. Intell. Syst. Technol. 15 (3) (2024) 1–45.

[15] B. Min, H. Ross, E. Sulem, A.P.B. Veyseh, T.H. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, ACM Comput. Surv. 56 (2) (2023) 1–40.

[16] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, 2023, arXiv preprint arXiv:2303.08774.

[17] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y.T. Lee, Y. Li, S. Lundberg, et al., Sparks of artificial general intelligence: Early experiments with gpt-4, 2023, arXiv preprint arXiv:2303.12712.

[18] Y. Cao, S. Li, Y. Liu, Z. Yan, Y. Dai, P.S. Yu, L. Sun, A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt, 2023, arXiv preprint arXiv:2303.04226.

[19] C. Wu, S. Yin, W. Qi, X. Wang, Z. Tang, N. Duan, Visual chatgpt: Talking, drawing and editing with visual foundation models, 2023, arXiv preprint arXiv:2303.04671.

[20] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al., A survey on evaluation of large language models, ACM Trans. Intell. Syst. Technol. (2023).

[21] A.M. Newman, M. Weiss, A survey of linear and mixed-integer optimization tutorials, INFORMS Trans. Educ. 14 (1) (2013) 26–38.

[22] M. Savelsbergh, A branch-and-price algorithm for the generalized assignment problem, Oper. Res. 45 (6) (1997) 831–841.

[23] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, et al., MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library, Math. Program. Comput. 13 (3) (2021) 443–490.

[24] B. Wang, Q. Ye, Stochastic gradient descent with nonlinear conjugate gradient-style adaptive momentum, 2020, ArXiv, abs/2012.02188 URL https://api.semanticscholar.org/CorpusID:227255095.

[25] D. Liu, S. Xue, B. Zhao, B. Luo, Q. Wei, Adaptive dynamic programming for control: A survey and recent advances, IEEE Trans. Syst. Man Cybern.: Syst. 51 (1) (2020) 142–160.

[26] C. Cerrone, R. Cerulli, B. Golden, Carousel greedy: A generalized greedy algorithm with applications in optimization, Comput. Oper. Res. 85 (2017) 97–112.

[27] E. Alba, R. Martí, Metaheuristic Procedures for Training Neural Networks, vol. 35, Springer Science & Business Media, 2006.

[28] A. Amuthan, K.D. Thilak, Survey on tabu search meta-heuristic optimization, in: 2016 International Conference on Signal Processing, Communication, Power and Embedded System, SCOPES, IEEE, 2016, pp. 1539–1543.

[29] T.K. Gupta, K. Raza, Optimizing deep feedforward neural network architecture: A tabu search based approach, Neural Process. Lett. 51 (3) (2020) 2855–2870.

[30] H. Wang, C. He, Z. Li, A new ensemble feature selection approach based on genetic algorithm, Soft Comput. 24 (2020) 15811–15820.

[31] S. Lee, J. Kim, H. Kang, D.-Y. Kang, J. Park, Genetic algorithm based deep learning neural network structure and hyperparameter optimization, Appl. Sci. 11 (2) (2021) 744.

[32] Q. Yang, Y. Li, X.-D. Gao, Y.-Y. Ma, Z.-Y. Lu, S.-W. Jeon, J. Zhang, An adaptive covariance scaling estimation of distribution algorithm, Mathematics 9 (24) (2021) 3207.

[33] W. Dong, Y. Wang, M. Zhou, A latent space-based estimation of distribution algorithm for large-scale global optimization, Soft Comput. 23 (2019) 4593–4615.

[34] Y. Li, T. Han, S. Tang, C. Huang, H. Zhou, Y. Wang, An improved differential evolution by hybridizing with estimation-of-distribution algorithm, Inform. Sci. 619 (2023) 439–456.

[35] C.J.C.H. Watkins, Learning from delayed rewards, 1989.

[36] H. Hasselt, Double Q-learning, Adv. Neural Inf. Process. Syst. 23 (2010).

[37] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (1992) 229–256.

[38] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning, PMLR, 2015, pp. 1889–1897.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, arXiv preprint arXiv:1707.06347.

[40] V. Konda, J. Tsitsiklis, Actor-critic algorithms, Adv. Neural Inf. Process. Syst. 12 (1999).

[41] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: International Conference on Machine Learning, Pmlr, 2014, pp. 387–395.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013, arXiv preprint arXiv:1312.5602.

[43] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, 2016.

[44] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, Adv. Neural Inf. Process. Syst. 30 (2017).

[45] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2015, arXiv preprint arXiv:1509.02971.

[46] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, PMLR, 2016, pp. 1928–1937.

[47] S. Desale, A. Rasool, S. Andhale, P. Rane, Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey, Int. J. Comput. Eng. Res. Trends 351 (5) (2015) 2349–7084.

[48] F. Neri, V. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artif. Intell. Rev. 33 (2010) 61–106.

[49] S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution–an updated survey, Swarm Evol. Comput. 27 (2016) 1–30.

[50] Y.-j. Shi, H.-f. Teng, Z.-q. Li, Cooperative co-evolutionary differential evolution for function optimization, in: Advances in Natural Computation: First International Conference, ICNC 2005, Changsha, China, August 27-29, 2005, Proceedings, Part II 1, Springer, 2005, pp. 1080–1088.

[51] Z. Cao, L. Wang, Y. Shi, X. Hei, X. Rong, Q. Jiang, H. Li, An effective cooperative coevolution framework integrating global and local search for large scale optimization problems, in: 2015 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2015, pp. 1986–1993.

[52] G.A. Trunfio, P. Topa, J. Wąs, A new algorithm for adapting the configuration of subcomponents in large-scale optimization with cooperative coevolution, Inform. Sci. 372 (2016) 773–795.

[53] Y. Liang, Z. Ren, X. Yao, Z. Feng, A. Chen, W. Guo, Enhancing Gaussian estimation of distribution algorithm by exploiting evolution direction with archive, IEEE Trans. Cybern. 50 (1) (2018) 140–152.

[54] Q. Zhang, J. Sun, E. Tsang, J. Ford, Hybrid estimation of distribution algorithm for global optimization, Eng. Comput. 21 (1) (2004) 91–107.

[55] A. Zhou, J. Sun, Q. Zhang, An estimation of distribution algorithm with cheap and expensive local search methods, IEEE Trans. Evol. Comput. 19 (6) (2015) 807–822.

[56] K. Lei, P. Guo, Y. Wang, X. Wu, W. Zhao, Solve routing problems with a residual edge-graph attention neural network, Neurocomputing 508 (2022) 79–98.

[57] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Process. Mag. 34 (6) (2017) 26–38.

[58] R. Zhang, A. Prokhorchuk, J. Dauwels, Deep reinforcement learning for traveling salesman problem with time windows and rejections, in: 2020 International Joint Conference on Neural Networks, IJCNN, IEEE, 2020, pp. 1–8.

[59] Q. Ma, S. Ge, D. He, D. Thaker, I. Drori, Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning, 2019, arXiv preprint arXiv:1911.04936.

[60] Y. Hu, Y. Yao, W.S. Lee, A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs, Knowl.-Based Syst. 204 (2020) 106244.

[61] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, Adv. Neural Inf. Process. Syst. 28 (2015).

[62] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, 2016, arXiv preprint arXiv:1611.09940.

[63] M. Nazari, A. Oroojlooy, L. Snyder, M. Takác, Reinforcement learning for solving the vehicle routing problem, Adv. Neural Inf. Process. Syst. 31 (2018).

[64] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. 30 (2017).

[65] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, L.-M. Rousseau, Learning heuristics for the tsp by policy gradient, in: Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, the Netherlands, June 26–29, 2018, Proceedings 15, Springer, 2018, pp. 170–181.

[66] W. Kool, H. Van Hoof, M. Welling, Attention, learn to solve routing problems!, 2018, arXiv preprint arXiv:1803.08475.

[67] W. Chao, J. Zhao, L. Jiao, L. Li, F. Liu, S. Yang, A match made in consistency heaven: when large language models meet evolutionary algorithms, 2024, arXiv preprint arXiv:2401.10510.

[68] C. Yang, X. Wang, Y. Lu, H. Liu, Q.V. Le, D. Zhou, X. Chen, Large language models as optimizers, 2023, arXiv preprint arXiv:2309.03409.

[69] Y. Chen, J. Arkin, Y. Hao, Y. Zhang, N. Roy, C. Fan, Prompt optimization in multi-step tasks (PROMST): Integrating human feedback and preference alignment, 2024, arXiv preprint arXiv:2402.08702.
[70] S. Zhang, C. Gong, L. Wu, X. Liu, M. Zhou, Automl-gpt: Automatic machine learning with gpt, 2023, arXiv preprint arXiv:2305.02499.
[71] A. AhmadiTeshnizi, W. Gao, M. Udell, OptiMUS: Optimization Modeling Using mip Solvers and large language models, 2023, arXiv preprint arXiv:2310.06116.
[72] H. Ye, J. Wang, Z. Cao, G. Song, ReEvo: Large language models as hyper-heuristics with reflective evolution, 2024, arXiv preprint arXiv:2402.01145.
[73] R. Zhong, Y. Xu, C. Zhang, J. Yu, Leveraging large language model to generate a novel metaheuristic algorithm with CRISPE framework, 2024, arXiv preprint arXiv:2403.16417.
[74] E. Meyerson, M.J. Nelson, H. Bradley, A. Gaier, A. Moradi, A.K. Hoover, J. Lehman, Language model crossover: Variation through few-shot prompting, 2023, arXiv preprint arXiv:2302.12170.
[75] S. Liu, C. Chen, X. Qu, K. Tang, Y.-S. Ong, Large language models as evolutionary optimizers, 2023, arXiv preprint arXiv:2310.19046.
[76] S. Brahmachary, S.M. Joshi, A. Panda, K. Koneripalli, A.K. Sagotra, H. Patel, A. Sharma, A.D. Jagtap, K. Kalyanaraman, Large language model-based evolutionary optimizer: Reasoning with elitism, 2024, arXiv preprint arXiv:2403.02054.
[77] F. Liu, X. Lin, Z. Wang, S. Yao, X. Tong, M. Yuan, Q. Zhang, Large language model for multi-objective evolutionary optimization, 2023, arXiv preprint arXiv:2310.12541.
[78] H. Bradley, A. Dai, H. Teufel, J. Zhang, K. Oostermeijer, M. Bellagente, J. Clune, K. Stanley, G. Schott, J. Lehman, Quality-diversity through AI feedback, 2023, arXiv preprint arXiv:2310.13032.
[79] M.R. Zhang, N. Desai, J. Bae, J. Lorraine, J. Ba, Using large language models for hyperparameter optimization, in: NeurIPS 2023 Foundation Models for Decision Making Workshop, 2023.
[80] S. Liu, C. Gao, Y. Li, Large language model agent for hyper-parameter optimization, 2024, arXiv preprint arXiv:2402.01881.
[81] R. Ma, X. Wang, X. Zhou, J. Li, N. Du, T. Gui, Q. Zhang, X. Huang, Are large language models good prompt optimizers? 2024, arXiv preprint arXiv:2402.02101.
[82] F. Liu, X. Tong, M. Yuan, Q. Zhang, Algorithm evolution using large language model, 2023, arXiv preprint arXiv:2311.15249.
[83] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, Q. Zhang, An example of evolutionary computation+ large language model beating human: Design of efficient guided local search, 2024, arXiv preprint arXiv:2401.02051.
[84] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, T. Rocktäschel, Promptbreeder: Self-referential self-improvement via prompt evolution, 2023, arXiv preprint arXiv:2309.16797.
[85] M. Pluhacek, A. Kazikova, T. Kadavy, A. Viktorin, 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.
[86] H. Bradley, H. Fan, T. Galanos, R. Zhou, D. Scott, 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.
[87] E. Hemberg, S. Moskal, U.-M. O'Reilly, Evolving code with a large language model, 2024, arXiv preprint arXiv:2401.07102.
[88] X. Lin, Z. Wu, Z. Dai, W. Hu, Y. Shu, S.-K. Ng, P. Jaillet, B.K.H. Low, Use your INSTINCT: instruction optimization using neural bandits coupled with transformers, 2023, arXiv preprint arXiv:2310.02905.
[89] Y. Tian, L. Si, X. Zhang, K.C. Tan, Y. Jin, Local model-based Pareto front estimation for multiobjective optimization, IEEE Trans. Syst. Man Cybern.: Syst. 53 (1) (2022) 623–634.
[90] Y. Tian, R. Cheng, X. Zhang, F. Cheng, Y. Jin, An indicator-based multiobjective evolutionary algorithm with reference point adaptation for better versatility, IEEE Trans. Evol. Comput. 22 (4) (2017) 609–622.
[91] H.X. Choong, Y.-S. Ong, A. Gupta, C. Chen, R. Lim, Jack and masters of all trades: One-pass learning sets of model sets from large pre-trained models, IEEE Comput. Intell. Mag. 18 (3) (2023) 29–40.
[92] J. Baumann, O. Kramer, Evolutionary multi-objective optimization of large language model prompts for balancing sentiments, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2024, pp. 212–224.
[93] A. Klein, J. Golebiowski, X. Ma, V. Perrone, C. Archambeau, Structural pruning of large language models via neural architecture search, 2023.
[94] X. Ma, G. Fang, X. Wang, Llm-pruner: On the structural pruning of large language models, Adv. Neural Inf. Process. Syst. 36 (2023) 21702–21720.
[95] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig, Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, ACM Comput. Surv. 55 (9) (2023) 1–35.
[96] T. Wei, S. Wang, J. Zhong, D. Liu, J. Zhang, A review on evolutionary multitask optimization: Trends and challenges, IEEE Trans. Evol. Comput. 26 (5) (2021) 941–960.
[97] H. Yang, K. Li, InstOptima: Evolutionary multi-objective instruction optimization via large language model-based instruction operators, 2023, arXiv:2310.17630.
[98] K.K. Bali, A. Gupta, Y.-S. Ong, P.S. Tan, Cognizant multitasking in multiobjective multifactorial evolution: MO-MFEA-II, IEEE Trans. Cybern. 51 (4) (2020) 1784–1796.
[99] A. Gupta, Y.-S. Ong, L. Feng, K.C. Tan, Multiobjective multifactorial optimization in evolutionary multitasking, IEEE Trans. Cybern. 47 (7) (2016) 1652–1665.
[100] S. Gholami, M. Omar, Can pruning make large language models more efficient? 2023, arXiv preprint arXiv:2310.04573.
[101] N. Hansen, S.D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES), Evol. Comput. 11 (1) (2003) 1–18.
[102] T. Sun, Y. Shao, H. Qian, X. Huang, X. Qiu, Black-box tuning for language-model-as-a-service, in: International Conference on Machine Learning, PMLR, 2022, pp. 20841–20855.
[103] T. Sun, Z. He, H. Qian, Y. Zhou, X. Huang, X. Qiu, Bbtv2: Towards a gradient-free future with large language models, 2022, arXiv preprint arXiv:2205.11200.
[104] L. Yu, Q. Chen, J. Lin, 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.
[105] Z. Fei, M. Fan, J. Huang, Gradient-free textual inversion, in: Proceedings of the 31st ACM International Conference on Multimedia, 2023, pp. 1364–1373.
[106] Y. Chai, S. Wang, Y. Sun, H. Tian, H. Wu, H. Wang, Clip-tuning: Towards derivative-free prompt learning with a mixture of rewards, 2022, arXiv preprint arXiv:2210.12050.
[107] M. Shen, S. Ghosh, P. Sattigeri, S. Das, Y. Bu, G. Wornell, Reliable gradient-free and likelihood-free prompt tuning, 2023, arXiv preprint arXiv:2305.00593.
[108] A. Prasad, P. Hase, X. Zhou, M. Bansal, Grips: Gradient-free, edit-based instruction search for prompting large language models, 2022, arXiv preprint arXiv:2203.07281.
[109] J. Zhao, Z. Wang, F. Yang, Genetic prompt search via exploiting language model probabilities, in: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI, 2023, pp. 5296–5305.
[110] R. Pan, S. Xing, S. Diao, X. Liu, K. Shum, J. Zhang, T. Zhang, Plum: Prompt learning using metaheuristic, 2023, arXiv preprint arXiv:2311.08364.
[111] R. Pryzant, D. Iter, J. Li, Y.T. Lee, C. Zhu, M. Zeng, Automatic prompt optimization with" gradient descent" and beam search, 2023, arXiv preprint arXiv:2305.03495.
[112] Y. Zheng, Z. Tan, P. Li, Y. Liu, Black-box prompt tuning with subspace learning, 2023, arXiv preprint arXiv:2305.03518.
[113] Q. Sun, C. Han, N. Chen, R. Zhu, J. Gong, X. Li, M. Gao, Make prompt-based black-box tuning colorful: Boosting model generalization from three orthogonal perspectives, 2023, arXiv preprint arXiv:2305.08088.
[114] C. Han, L. Cui, R. Zhu, J. Wang, N. Chen, Q. Sun, X. Li, M. Gao, When gradient descent meets derivative-free optimization: A match made in black-box scenario, 2023, arXiv preprint arXiv:2305.10013.
[115] J. Sun, Z. Xu, H. Yin, D. Yang, D. Xu, Y. Chen, H.R. Roth, FedBPT: Efficient federated black-box prompt tuning for large language models, 2023, arXiv preprint arXiv:2310.01467.
[116] H. Xu, Y. Chen, Y. Du, N. Shao, Y. Wang, H. Li, Z. Yang, GPS: Genetic prompt search for efficient few-shot learning, 2022, arXiv preprint arXiv:2210.17041.
[117] H. Zhou, X. Wan, I. Vulić, A. Korhonen, Survival of the most influential prompts: Efficient black-box prompt search via clustering and pruning, in: The 2023 Conference on Empirical Methods in Natural Language Processing, 2023.
[118] R. Lapid, R. Langberg, M. Sipper, Open sesame! universal black box jailbreaking of large language models, 2023, arXiv preprint arXiv:2309.01446.
[119] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, Y. Yang, Connecting large language models with evolutionary algorithms yields powerful prompt optimizers, in: The Twelfth International Conference on Learning Representations, 2024.
[120] G. Pinna, D. Ravalico, L. Rovito, L. Manzoni, A. De Lorenzo, Enhancing large language models-based code generation by leveraging genetic improvement, in: European Conference on Genetic Programming (Part of EvoStar), Springer, 2024, pp. 108–124.
[121] S. Yu, S. Liu, Z. Lin, D. Pathak, D. Ramanan, Language models as black-box optimizers for vision-language models, 2023, arXiv preprint arXiv:2309.05950.
[122] Z. Guo, Y. Wei, M. Liu, Z. Ji, J. Bai, Y. Guo, W. Zuo, Black-box tuning of vision-language models with effective gradient approximation, 2023, arXiv preprint arXiv:2312.15901.
[123] S. Diao, Z. Huang, R. Xu, X. Li, Y. Lin, X. Zhou, T. Zhang, Black-box prompt learning for pre-trained language models, 2022, arXiv preprint arXiv:2201.08531.
[124] C. Singh, J.X. Morris, J. Aneja, A.M. Rush, J. Gao, Explaining data patterns in natural language with language models, in: Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP, 2023, pp. 31–55.

[125] R. Pryzant, D. Iter, J. Li, Y.T. Lee, C. Zhu, M. Zeng, Automatic prompt optimization with "Gradient Descent" and beam search, in: Conference on Empirical Methods in Natural Language Processing, 2023.

[126] Y.B. Li, K. Wu, SPELL: Semantic prompt evolution based on a LLM, 2023, arXiv preprint arXiv:2310.01260.

[127] Z. Zhang, S. Wang, W. Yu, Y. Xu, D. Iter, Q. Zeng, Y. Liu, C. Zhu, M. Jiang, Auto-instruct: Automatic instruction generation and ranking for black-box language models, in: Findings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, Association for Computational Linguistics, 2023.

[128] Y. Liu, Y. Sun, B. Xue, M. Zhang, G.G. Yen, K.C. Tan, A survey on evolutionary neural architecture search, IEEE Trans. Neural Netw. Learn. Syst. 34 (2) (2021) 550–570.

[129] X. Zhou, A.K. Qin, Y. Sun, K.C. Tan, A survey of advances in evolutionary neural architecture search, in: 2021 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2021, pp. 950–957.

[130] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, J. Mach. Learn. Res. 20 (55) (2019) 1–21.

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

[132] D. So, Q. Le, C. Liang, The evolved transformer, in: International Conference on Machine Learning, PMLR, 2019, pp. 5877–5886.

[133] J. Gao, H. Xu, H. Shi, X. Ren, L. Philip, X. Liang, X. Jiang, Z. Li, Autobert-zero: Evolving bert backbone from scratch, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, 2022, pp. 10663–10671.

[134] V. Ganesan, G. Ramesh, P. Kumar, SuperShaper: Task-agnostic super pre-training of BERT models with variable hidden dimensions, 2021, arXiv preprint arXiv:2110.04711.

[135] Y. Yin, C. Chen, L. Shang, X. Jiang, X. Chen, Q. Liu, Autotinybert: Automatic hyper-parameter optimization for efficient pre-trained language models, 2021, arXiv preprint arXiv:2107.13686.

[136] M. Javaheripi, G. de Rosa, S. Mukherjee, S. Shah, T. Religa, C.C. Teodoro Mendes, S. Bubeck, F. Koushanfar, D. Dey, Litetransformersearch: Training-free neural architecture search for efficient language models, Adv. Neural Inf. Process. Syst. 35 (2022) 24254–24267.

[137] Q. Zhou, K. Sheng, X. Zheng, K. Li, Y. Tian, J. Chen, R. Ji, Training-free transformer architecture search with zero-cost proxy guided evolution, IEEE Trans. Pattern Anal. Mach. Intell. (2024).

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

[139] N. Tao, A. Ventresque, T. Saber, Program synthesis with generative pre-trained transformers and grammar-guided genetic programming grammar, in: 2023 IEEE Latin American Conference on Computational Intelligence (la-CCI), IEEE, 2023, pp. 1–6.

[140] M. Weyssow, X. Zhou, K. Kim, D. Lo, H. Sahraoui, Exploring parameter-efficient fine-tuning techniques for code generation with large language models, 2023, arXiv preprint arXiv:2308.10462.

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

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

[143] F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C.J. Anderson, M.Q. Feldman, A. Guha, M. Greenberg, A. Jangda, MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation, IEEE Trans. Softw. Eng. 49 (7) (2023) 3675–3691.

[144] Z. Ji, P. Ma, Z. Li, S. Wang, Benchmarking and explaining large language model-based code generation: A causality-centric approach, 2023, arXiv preprint arXiv:2310.06680.

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

[146] 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, J. Mach. Learn. Res. 24 (240) (2023) 1–113.

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

[148] L. Xiao, X. Shan, X. Chen, PatternGPT: A pattern-driven framework for large language model text generation, in: Proceedings of the 2023 12th International Conference on Computing and Pattern Recognition, 2023, pp. 72–78.

[149] Y. Chen, Z. Wen, G. Fan, Z. Chen, W. Wu, D. Liu, Z. Li, B. Liu, Y. Xiao, MAPO: Boosting large language model performance with model-adaptive prompt optimization, in: Findings of the Association for Computational Linguistics: EMNLP 2023, 2023, pp. 3279–3304.

[150] L. Xiao, X. Chen, X. Shan, Enhancing large language models with evolutionary fine-tuning for news summary generation, J. Intell. Fuzzy Systems (Preprint) 1–13.

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

[152] Y. Liang, R. Zhang, L. Zhang, P. Xie, DrugChat: towards enabling ChatGPT-like capabilities on drug molecule graphs, 2023, arXiv preprint arXiv:2309.03907.

[153] X. Lin, C. Xu, Z. Xiong, X. Zhang, N. Ni, B. Ni, J. Chang, R. Pan, Z. Wang, F. Yu, et al., PanGu Drug Model: learn a molecule like a human, Biorxiv (2022) 2022-2003.

[154] J. Lu, W. Zhong, W. Huang, Y. Wang, F. Mi, B. Wang, W. Wang, L. Shang, Q. Liu, Self: Language-driven self-evolution for large language model, 2023, arXiv preprint arXiv:2310.00533.

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

[156] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al., Improving language understanding by generative pre-training, 2018.

[157] R. Suzuki, T. Arita, An evolutionary model of personality traits related to cooperative behavior using a large language model, Sci. Rep. 14 (1) (2024) 5989.

[158] G. De Marzo, L. Pietronero, D. Garcia, Emergence of scale-free networks in social interactions among large language models, 2023, arXiv preprint arXiv:2312.06619.

[159] J. Wu, W. Gan, Z. Chen, S. Wan, S.Y. Philip, Multimodal large language models: A survey, in: 2023 IEEE International Conference on Big Data (BigData), IEEE, 2023, pp. 2247–2256.

[160] C. Cui, Y. Ma, X. Cao, W. Ye, Y. Zhou, K. Liang, J. Chen, J. Lu, Z. Yang, K.-D. Liao, et al., A survey on multimodal large language models for autonomous driving, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, pp. 958–979.

[161] K.C. Tan, L. Feng, M. Jiang, Evolutionary transfer optimization-a new frontier in evolutionary computation research, IEEE Comput. Intell. Mag. 16 (1) (2021) 22–33.

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

[163] Z. CHen, L. Cao, S. Madden, J. Fan, N. Tang, Z. Gu, Z. Shang, C. Liu, M. Cafarella, T. Kraska, Seed: Simple, efficient, and effective data management via large language models, 2023, arXiv preprint arXiv:2310.00749.

[164] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, X. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, ACM Comput. Surv. 54 (4) (2021) 1–34.