



The Next Stage of Evolutionary Computation in the Era of Agentic Generative AI

Michael Palk¹, Stefan Voß¹(✉), and Fred Glover²

¹ Institute of Information Systems, University of Hamburg, Hamburg, Germany
{michael.palk,stefan.voss}@uni-hamburg.de

² Entanglement Inc., Miami, FL, USA
fred@entanglement.ai

Abstract. Evolutionary computation (EC) suffers from a lack of standardized open-source frameworks, the need of domain-specific configurations for every new problem, and missing general guidelines for choosing appropriate solution methods. Unlike other areas of artificial intelligence (AI), such as machine learning, which provide ready-to-use programming libraries with reusable components, this degree of standardization is less pronounced in EC. In this paper, a vision for the next stage of EC is proposed, introducing an architecture for an agentic generative AI system designed to unify and automate EC processes, enabling higher levels of integration, autonomy, and self-organization. This represents a step toward a new paradigm for interacting with agentic and generative AI, supporting dynamic, multi-agent collaboration in which problem modeling, algorithmic design, and efficient computation are seamlessly integrated across the entire EC ecosystem. Ultimately, this could reduce the entry barrier and foster a broader adoption of EC.

Keywords: Evolutionary Computation · Agentic AI · Generative AI · Large language models

1 Introduction

Despite being designed to efficiently explore large search spaces and guide heuristics toward “near-optimal” solutions, evolutionary computation (EC), as well as metaheuristics in general, face notable challenges. A recurring issue is the need for problem-specific adaptations, coupled with limited replicability, often hindered by ambiguity and insufficient detail in method descriptions, which makes reproduction difficult. Applying the same evolutionary algorithm across different optimization problems requires domain-specific configurations that account for problem structure, constraints, and unique characteristics [51]. Moreover, compared to data-driven fields in artificial intelligence (AI), such as machine learning, EC suffers from a relative scarcity of ready-to-use libraries and standardized open-source frameworks, increasing implementation complexity and time

requirements [72]. Older approaches to designing reusable optimization software class libraries [78], including EC approaches, often seem to be neglected; exceptions include frameworks such as Pymoo [11]. EC approaches are frequently re-implemented from scratch in various programming languages, making it difficult to build upon existing work or rapidly translate ideas into executable code.

Additional to these practical issues, EC suffers from a lack of reusable, configurable, and extensible algorithmic components. EC approaches are often developed based on intuition, heuristics, and trial-and-error rather than grounded in rigorous mathematical theory. Many evolutionary algorithms draw inspiration from human intelligence, natural phenomena, or biological processes, yet the translation of these concepts into effective algorithms is quite informal. Often such nature-inspired names offer little in terms of deeper algorithmic understanding [6, 13, 70, 75, 79]. Another aspect is that EC algorithms typically require careful parameter tuning to achieve good performance on specific problem instances, with parameter selection relying heavily on practitioner experience and expertise, making it more of an art than a science. These parameters govern the exploration-exploitation trade-off, and finding the right balance is challenging. Moreover, there are few general guidelines to assist practitioners in selecting the most appropriate EC approach for a given problem, leaving choices largely dependent on intuition, prior experience, or personal preference rather than solid scientific guidance [50].

In contrast, reference set-based methods (also referred to as consistent chains or vocabulary building) offer a more structured and transparent search framework. Instead of relying on randomized population dynamics and numerous control parameters, these methods maintain and systematically evolve a reference set of high-quality and diverse solutions, generating new candidates through strategic combinations rather than stochastic variation. In contrast to other evolutionary methods, such as genetic algorithms, scatter search is based on the premise that systematic designs and methods to create new solutions afford significant benefits beyond those derived from the recourse to randomization, using strategies for both search diversification and intensification [49]. Using the notion of reference sets would also allow a better understanding of what “near-optimal” solutions would mean. Furthermore, a different concept relying on specific ideas of reference sets is the idea of the fixed set search metaheuristic [32, 33].

A promising way to address these challenges is through concepts and tools from generative and agentic AI. Generative AI, particularly large language models (LLMs) [39], can help address practical issues by reproducing existing algorithms that are not available in standard libraries and by generating code to reduce manual implementation effort. They can also support automated heuristic design by suggesting or adapting operators based on problem descriptions for optimization problems [45]. Another related paradigm is natural language for optimization (NL4OPT), which leverages LLMs to formulate, analyze, and solve optimization models [1, 3, 27, 40, 86]. Based on these developments, agentic AI can bring structure to the entire EC process by organizing it as a multi-

agent system (MAS), where LLM-based agents handle tasks such as optimization model formulation, heuristic design, computational resource allocation, and result interpretation. Together, generative AI can support algorithmic designers who use agentic frameworks, making EC more systematic, adaptable, and transparent.

The goal of this paper is to present a vision for the next stage of EC that involves leveraging agentic generative AI to unify and automate EC processes, enabling higher integration, autonomy, and self-organization. By combining problem modeling, algorithm modules, efficient computation, and solution interpretation within a MAS, this approach could reduce the reliance on problem-specific configurations and manual fine-tuning. The synergy between agentic generative AI and EC can facilitate automated algorithm generation, adaptive parameter selection, and seamless execution, addressing current limitations in replicability, usability, and infrastructure integration. Ultimately, such a system could lower entry barriers and promote a broader adoption of EC across diverse application domains.

This paper is structured as follows. Section 2 presents core concepts of generative and agentic AI. In Sect. 3, approaches from the literature of synthesizing EC with generative and agentic AI are reviewed. Based on this, an agentic generative AI system architecture for EC is proposed in Sect. 4. The paper ends in Sect. 5 with a conclusion and future outlook.

2 Foundations of Generative and Agentic AI

In this section, foundations necessary for understanding generative and agentic AI systems are provided, including the concepts of LLMs, retrieval-augmented generation (RAG), knowledge graphs (KGs), and MAS.

Human-computer interaction systems powered by LLMs enable the automatic creation of text, images, videos, and code based on user-provided prompts, i.e., descriptions in natural language of the desired output [55]. LLMs built on transformer architectures form the backbone of modern conversational human-computer interaction systems and are often referred to as foundation models. Moreover, the emergence of pre-trained transformer-based models has enhanced LLMs through transfer learning. By leveraging knowledge from pre-training, these models can be fine-tuned for specific downstream tasks, potentially leading to improved performance and adaptability [100].

Prompt engineering involves designing and refining input prompts to guide LLMs toward generating desired responses. As the quality of outputs depend on how prompts are framed, it focuses on crafting concise, structured, and context-specific queries to achieve the desired result. LLMs generate outputs by synthesizing patterns and statistical relationships learned during training, predicting plausible continuations of text or code based on the given context [96].

The rapid advancement of LLMs in recent years has created an ecosystem too vast to be listed comprehensively; naming specific models is often of limited value since they are frequently updated or replaced by newer versions. These include

closed-source models and open-source models, which are often part of larger model families offering multiple versions tailored to various scales and use cases, whereas model merging has become popular recently [90]. Alongside general-purpose models, domain-specific variants have appeared [43], particularly for specific tasks, such as code generation [91]. In addition, techniques, such as distillation have produced smaller, more efficient models that can be deployed in resource-constrained environments [36, 89], e.g., edge LLMs [98]. There is also a growing trend toward the development of small language models (SLMs), which reflect a shift in focus from sheer scale to targeted performance [82].

LLMs face several security, ethical, and explainability challenges that impact their transparent, responsible, and reliable deployment [84]. Issues such as inconsistencies, biases, and the generation of fabricated facts or references, commonly known as hallucinations, remain significant concerns [77, 80]. Another major concern is data poisoning, where malicious or biased data is introduced during training to manipulate the model's behavior [97]. A common vulnerability lies in prompt injection attacks, in which adversarial inputs are crafted to override system instructions, bypass safety filters, generate harmful content, or leak sensitive information [66]. LLMs also suffer from an explainability issue, as their internal decision-making processes are largely opaque [8]. These challenges highlight the need for robust safeguards, adaptable guidelines, regulations, ethical frameworks, disclosure practices, and continual monitoring in the deployment of LLMs.

To mitigate the issues of hallucinations, RAG can enhance LLM capabilities by integrating external knowledge retrieval into the generation process [15]. This hybrid approach is particularly effective in maintaining factual accuracy and contextual relevance in LLM systems, potentially reducing the current issues of hallucinations [28]. Furthermore, KGs can reduce hallucinations and factual inaccuracies by providing structured, reliable, and consistent information. They offer semantic abstraction and enhance explainability, giving a level of structure and reliability to strengthen RAG-based approaches [57, 74]. LLMs can not only assist in building and extracting relevant knowledge but also enable natural language querying and reasoning over KGs [92].

Rather than viewing LLMs as standalone solutions, they can be seen as parts of MAS, equipped with tools and authorized application programming interface (API) access [61]. The Model Context Protocol (MCP) is an open standard that provides a unified way to supply structured context to LLMs and connect them to various tools [62]. MAS can address various problems by enabling multiple autonomous agents to interact and collaborate within a shared environment. Each agent uses local observations, interactions with neighbors, and environmental feedback to learn relevant context and appropriate actions. This knowledge guides decision making, helping agents perform tasks, and contribute to overall system goals [18]. Specifically in the context of LLMs, multiple agents can manage complex interactions and tasks, a concept known as agentic AI [2], or alternatively, agentic generative AI, to emphasize the focus on LLM-based agents. Unlike individual agents handling isolated prompts, agentic generative

AI involves a coordinated system where autonomous agents collaborate, communicate, reason over shared context, and delegate tasks dynamically to achieve broader objectives, as well as coordinated orchestration of workflows [73].

While generative and agentic AI provide powerful capabilities for automation, reasoning, and interaction, their effective use requires careful consideration of their technical, ethical, and operational challenges. Addressing these issues is essential before such systems can be reliably and responsibly integrated into computational frameworks [26].

3 Utilization of Generative and Agentic AI in Evolutionary Computation

This section reviews current approaches to the incorporation of agentic generative AI into EC and metaheuristics.

The work of [88] recognizes two main avenues of synergy between LLMs and evolutionary algorithms: LLMs enhanced by evolutionary algorithms and evolutionary algorithms enhanced by LLMs. The first case considers evolutionary algorithms improving LLMs; the second case concerns LLMs improving search processes. However, high computational demands, concerns about data privacy, and limited explainability can hinder their broader deployment, particularly in sensitive or resource-limited settings [29]. The authors of [81] explore the use of LLMs to solve optimization problems by integrating them with metaheuristics such as simulated annealing for the traveling salesman problem. Similarly in [47], LLM-driven evolutionary algorithms are proposed, an approach that uses LLMs to perform selection, crossover, and mutation in evolutionary search, which includes a self-adaptation mechanism for balancing exploration and exploitation. Also the work of [64] introduces a hybrid approach that uses LLMs as pattern recognition tools to enhance genetic algorithms by guiding the search process on a social network-based optimization problem.

The use of LLMs to generate metaheuristics is investigated in [58] by selecting swarm-based methods, detailing their components for hybridization, and proposing innovative hybrid approaches. The task also involves providing pseudocode for the algorithms, along with justifications for their design. The study of [99] explores the use of generative AI to automatically generate metaheuristics through prompt engineering techniques. While the generated algorithms achieve competitive performance, the authors note that LLMs often lack deep domain-specific expertise and rely heavily on learned patterns, which limits the originality and innovation of the resulting metaheuristics. The authors of [71] propose a framework, which leverages LLMs to generate and refine algorithms for optimization tasks by iteratively mutating and selecting based on performance feedback. A key challenge is the strong dependency on the quality and structure of the prompts used to guide the LLM, as poorly designed prompts can introduce biases or restrict the diversity and creativity of the generated metaheuristics. In [65], a generative AI-driven framework using LLMs for the automatic design and configuration of metaheuristic algorithms is presented, demonstrating its effectiveness in real-world engineering optimization and its potential to lower the barrier

to applying EC in complex domains. A novel mutation control mechanism for LLM-driven evolutionary frameworks is proposed in [94], introducing dynamic, prompt-based mutation rates inspired by genetic algorithm theory and governed by a heavy-tailed power-law distribution to enhance exploration, exploitation, and convergence efficiency in automated metaheuristic design.

FunSearch, an evolutionary method for automated heuristic design, combining an LLM with an evaluator, is investigated in [63], which searches for programs that describe how to solve problems rather than just producing solutions. Similarly, the Evolution of Heuristic procedure is introduced in [44] to provide a method that combines LLMs and EC to automatically design heuristics by evolving prompts into executable code. The study of [93] explores hyper-heuristics powered by LLMs and propose reflective evolution, which combines evolutionary search with LLM reflections to generate sample-efficient solvers across diverse combinatorial optimization problems. The proposed LLM-based memetic framework of [60] dynamically combines LLM-driven exploration with domain-specific local refinement and memory-aware reflection, achieving superior performance across benchmark combinatorial problems, and demonstrating that LLMs can act as adaptive, interpretable heuristic generators for complex real-world optimization tasks. Although such LLM-based approaches can generate superior heuristics, their evaluation may be limited to narrow datasets and their performance gains may not justify the high cost of producing them [68].

Despite these advances, the studies focus mainly on isolated stages of the optimization process, such as heuristic generation or mutation control, rather than providing a unified framework that integrates all phases from problem formulation to solution interpretation. What remains missing is a cohesive agent-based architecture that connects modeling, algorithm design, execution, and insights generation by providing not only a complete system with technical infrastructure, but also an agentic framework controllable through natural language queries, equipped with modular algorithmic components, and capable of incorporating data-specific instance characteristics. Such an approach would enable a more holistic and accessible workflow, reducing the need to repeatedly perform recurring tasks and configurations. Such unified, agent-based, end-to-end frameworks are comparatively more mature in machine learning, where natural-language-driven pipelines, modular components, and automated insight generation are increasingly common [25,37], but remain comparatively scarce and underexplored in the domain of EC.

Going into such a direction, the work of [85] presents a multi-agent LLM-driven framework that automates and coordinates the full evolutionary optimization process. By assigning specialized roles to multiple LLM agents and incorporating a continuous feedback loop, this system streamlines tasks from problem formulation and population initialization to optimization and solution evaluation. This integrated, agentic generative AI approach reduces reliance on domain expertise, enables dynamic refinement of solutions in real time, and provides a more holistic, automated workflow for scheduling problems.

4 An Agentic Generative AI System for Evolutionary Computation

In this section, a possible next stage of EC is proposed, aiming to tackle the aforementioned challenges and limitations in current EC. Rather than relying on isolated solution approaches, an architecture is suggested for an agentic generative AI system designed to unify and automate EC processes, enabling greater levels of integration, autonomy, and self-organization. This approach could represent a step toward a new paradigm for interacting with agentic and generative AI, enabling dynamic, multi-agent collaboration in which problem modeling, algorithmic design, efficient computation, and insights generation are integrated across the entire EC ecosystem. Such a framework could serve as a general blueprint for the use of agentic generative AI in the context of EC, which can then be concretized and tailored to specific problem domains and applications.

By viewing the application of EC as a holistic process rather than focusing solely on the algorithmic component, it can be treated as an end-to-end methodology for solving real-world problems, seamlessly embedded within operational systems. This perspective has the potential to lower the entry barrier for practitioners, simplify adoption, and foster a broader and more effective use of EC across diverse domains.

Such an architecture should be viewed primarily as a conceptual suggestion that provides a bird's-eye view of the entire EC process, divided into its key functional areas. Its purpose is to capture the holistic structure of EC when integrated with agentic and generative AI, rather than to prescribe a fixed implementation. Current research on agent-based systems in software development has introduced both new challenges and opportunities for software architecture, particularly in the design of AI-enabled systems that incorporate autonomous agents [9, 76, 95]. These developments emphasize the need for architectural principles, design patterns, and frameworks that address issues such as system performance, reliability, coordination, and security [7, 83].

The proposed system architecture is illustrated in Fig. 1. The workflow begins with a query from a user, with supervision provided in a human-in-the-loop manner. This supervision represents a critical component, as the system must include mechanisms and entry points for intervention, allowing an oversight of the system's status and the detection of potential malfunctions. Such a system could be implemented within LangGraph [10], providing an open-source, provider-agnostic environment for orchestrating the workflow, supporting multiple LLMs, APIs, and tools without being tied to a single ecosystem. The LLM-based agents themselves remain interchangeable across different providers, helping to avoid reliability issues and vendor lock-in. In resource-constrained environments like local devices, specialized SLMs may offer a suitable alternative, provided similar performance can be achieved for the given task. In addition, Agent2Agent [62] is an open communication protocol that enables AI agents to collaborate across different platforms and frameworks. In the following, the individual components of the system are described in detail.

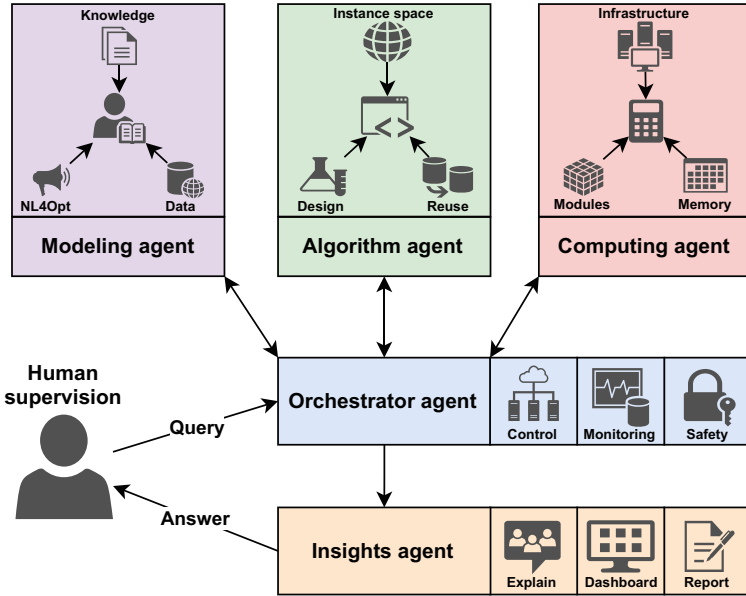


Fig. 1. Architecture of an agentic generative AI system for EC, consisting of an orchestrator, modeling, algorithm, computing, and insights agent.

4.1 Orchestrator Agent

The orchestrator agent coordinates interactions among all other agents in the workflow, ensuring that information flows smoothly throughout the system. Its role is to balance competing requirements, including model performance, response speed, data privacy, and cost efficiency, while maintaining risk controls and safeguards. The orchestrator establishes checkpoints for human supervision, enabling intervention at critical decision stages. It also manages interplay and communication between agents, providing oversight to ensure consistency, coherence, and compliance with safety requirements through established guardrails. Additionally, the orchestrator implements mechanisms to track system behavior over time, detect potential failures, and optimize resource allocation. Through this combination of coordination, oversight, and governance, the orchestrator agent establishes a foundation of reliability and trustworthiness, allowing the multi-agent framework to operate transparently, effectively, and responsibly.

Monitoring and evaluation tools could be integrated to provide observability, logging, and performance tracking of agent interactions, complementing the orchestrator's active coordination with structured oversight and quality control across the system. This can help to avoid cost escalation from infinite loops, complex internal execution, environmental variability, and interactions with untrusted entities. Safety guardrails, activity logs, monitoring systems, resource management, and human-in-the-loop oversight are, therefore, crucial to ensure alignment with goals and prevent unintended behaviors [17]. One pos-

sible implementation would be through the open-source framework LangSmith, thereby avoiding vendor lock-in, as in the case of LangGraph [52].

4.2 Modeling Agent

The modeling agent, whose workflow is illustrated in Fig. 2, is tasked with developing accurate optimization model formulations by integrating domain knowledge retrieved from sources such as academic papers, industry reports, and technical documentation via LLMs [16, 34, 53, 67]. This process can help mitigate false assumptions and better capture real-world complexity, while RAG approaches can further enhance it by reducing hallucinations [30, 41]. The interaction between KGs and LLMs can facilitate semantic understanding, relationship discovery, and context-aware reasoning [42]. Furthermore, the ontological structures embedded within KGs can be translated into optimization model formulations via NL4Opt, which align semantic representations with mathematical constructs to maintain logical coherence and ensure that the derived models embody the encoded domain knowledge [54, 56]. Consequently, this approach reduces the manual effort of defining constraints, which can be both time-consuming and error-prone.

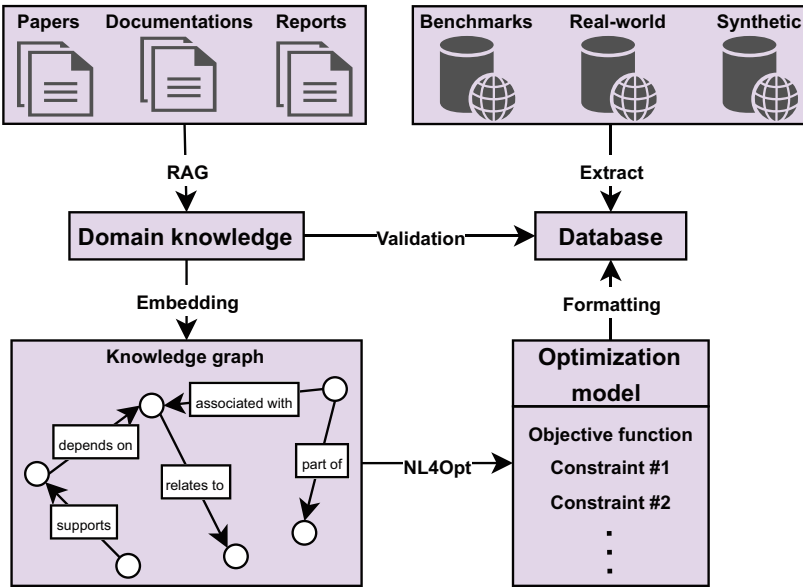


Fig. 2. The workflow of the modeling agent involves retrieving various sources via RAG to construct a KG. Benchmark, real-world, and synthetic data are extracted and validated against the domain knowledge. The KG is read through NL4Opt to generate the optimization model, and the database is formatted to fit the model formulation.

The agent can further validate the model for consistency, completeness, and correctness, identify infeasibilities, and confirm solvability using the chosen optimization methods [14]. A step in such direction is represented by Gurobi AI modeling [38], which can enhance optimization modeling using generative AI and could be accessed by the modeling agent via API calls. Conceptually, this enables an autonomous problem-solving agent with defined permissions and interfaces, capable of recognizing required modeling tasks, assessing the available tools and resources, and selecting or executing the appropriate operations to generate optimization model formulations. It could also retrieve real-world data, as well as benchmark instances from repositories or directly from research papers [59], generate customized instances when benchmarks are insufficient [24], and extend existing datasets to cover realistic scenarios. Validating against the gathered domain knowledge can further ensure that the retrieved or generated data is realistic and representative. Finally, the agent could manage preprocessing, transformation, and formatting of both retrieved and generated instances [46] to meet the requirements for the data formats of the optimization model.

4.3 Algorithm Agent

As illustrated in Fig. 3, the algorithm agent receives information from the orchestrator agent, such as the problem formulation and test instances from the modeling agent (marked in purple to highlight the information flow between the two agents).

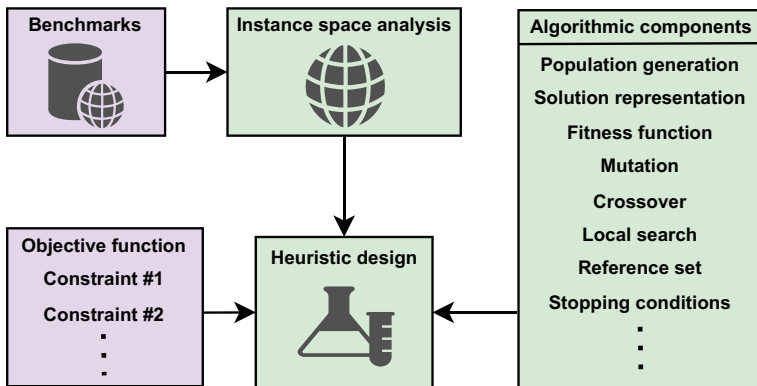


Fig. 3. The workflow of the algorithm agent involves performing instance space analysis based on the received benchmarks, which, alongside the model formulation, serves as a key factor in the automated heuristic design process. During this process, reusable, configurable, and extensible algorithmic components are utilized to cover the main steps of EC, such as defining the fitness function and stopping conditions, generating the population, applying mutation and crossover operations, as well as implementing guidance mechanisms based on reference sets. (Color figure online)

It begins with an instance space analysis [69] to select solution methods based on the available information, assess the diversity of the benchmark data, and identify potential biases in the data. Based on this analysis, it employs an automated heuristic design with reusable, configurable, and extensible algorithmic components, including constructive methods for population generation, choice of solution representation, the definition of stopping conditions, and local search operators. The agent can also configure templates for evolutionary principles such as mutation, crossover, and fitness functions, which may benefit from the integration of reference set-based strategies to systematically guide solution generation, combination, and intensification for a more structured search performance. This approach could enable automatic algorithm configuration, similar to parameter or hyperparameter optimization, reducing the need for domain-specific tuning while adapting to problem structure and instance characteristics. In essence, this setup serves as a test suite or algorithmic sandbox environment, in which various algorithmic strategies such as scatter search [21] can be systematically designed, evaluated, and refined under controlled conditions [20].

Building on this concept, structured combination strategies could allow the agent to generate new solutions by progressively extending partial solutions toward selected elite solutions using constructive or destructive neighborhoods. In this process, guiding solutions act as vectors of votes, which indicate promising components to include and are systematically updated as decisions are made, enabling the creation of trial solutions in a controlled and reproducible manner. Vocabulary-building techniques further enhance this approach by identifying frequently useful partial configurations and reusing them as higher-order components, reducing combinatorial complexity and supporting more efficient exploration of the solution space.

Unlike conventional evolutionary methods that rely primarily on random variation, these strategies incorporate adaptive memory to guide intensification and diversification, systematically exploiting patterns in high-quality solutions [22]. Strategic oscillation complements this framework by deliberately moving across critical boundaries, such as feasible and infeasible regions, with controlled oscillations that balance exploration and exploitation while avoiding entrapment in local optima [23].

4.4 Computing Agent

In Fig. 4, the workflow of the computing agent is depicted which involves executing the generated algorithm (marked in green to highlight the information flow between the two agents), including its deployment in operational systems with real-world data (marked in purple). This process requires selecting appropriate technical infrastructure, such as cloud servers with multithreaded GPUs, to ensure high-performance computation and enable parallelization, since many components of EC can be effectively parallelized [5]. Population evaluations across multiple candidate solutions can be distributed to exploit data-level parallelism; fitness computations for independent instances can run concurrently; and variation operators such as mutation and crossover can be applied in parallel

across subsets of the population. Similar to the algorithm agent, the computing agent generates executable code within a modular framework or programming library, reducing the need to reimplement heuristics across different programming languages and facilitating the reuse and extension of existing work [12]. This interoperability between the two agents allows algorithmic components to be directly transferred into software modules so that the generated algorithm can be implemented and embedded into operational systems.

Memory mechanisms enable progress tracking and state preservation, allowing the agent to maintain contextual information across iterative steps. More specifically, such mechanisms store intermediate results, learned strategies, and partial solution structures, which can be incrementally refined over time. In alignment with evolutionary anytime algorithms [4], this enables the agent to continuously improve its output as additional computational time becomes available, while still being able to produce a satisfactory solution at any intermediate stage. This property is especially important under strict runtime or resource constraints, as encountered in many real-world scenarios, where obtaining a good solution quickly is often more valuable than obtaining an optimal one slowly.

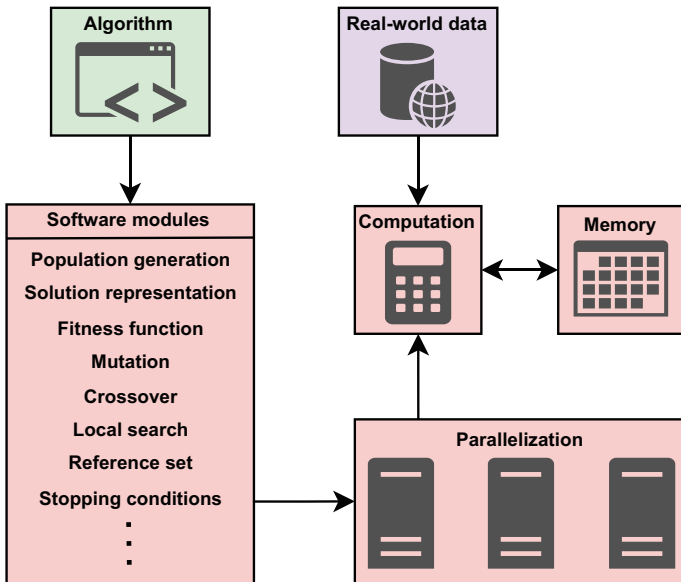


Fig. 4. The workflow of the computing agent involves implementing the generated solution algorithm, utilizing modular software components aligned with the algorithmic components provided by the algorithm agent. The use of parallelization and memory mechanisms enables deployment within operational systems for recurring computations on real-world data.

4.5 Insights Agent

Forwarded by the orchestrator agent, the results of computational studies are sent to the insights agent. It is responsible for transforming analytical outputs into structured answers for the user through reports, visualizations, and explanations. The insights agent contextualizes findings using the domain expertise provided by the modeling agent and integrates algorithmic information from the algorithm agent, allowing users to understand not only the outcomes but also the underlying decision processes and heuristics. Core functions include generating summaries, highlighting the most relevant insights, and tailoring both content and presentation to user needs. Similar to explainable AI, it emphasizes transparency, providing white-box interpretations rather than black-box outputs, and can trace the contribution of specific algorithms, parameters, and evolutionary components to the final results [19]. Such transparency is becoming increasingly important in light of the EU AI Act, which places explicit requirements on governance, regulatory compliance, and human oversight for AI systems [35].

Additionally, it might produce interactive dashboards that allow users to explore evolutionary algorithms, inspect intermediate solutions, and analyze the performance of different methods, as proposed in [31], and increasingly adopted within analytical workflows [48, 87]. By embedding these capabilities within operational systems, the insights agent could support decision making, provide adaptive recommendations, and highlight potential issues, facilitating both understanding and practical adoption of EC. In addition, the insights agent could interact dynamically with other agents to refine its interpretations, triggering new experiments or analyses when inconsistencies or knowledge gaps are detected. Over time, this continuous feedback loop could enhance the overall system's learning capability, enabling more accurate modeling, algorithm design, and computation in future iterations.

5 Conclusion

Despite the long-standing successes of EC in tackling complex, high-dimensional, and real-world problems, significant challenges remain in terms of reproducibility, parameter tuning, and the availability of reusable, extensible algorithmic components. To remain competitive and relevant alongside rapidly advancing AI paradigms such as deep learning, EC should embrace more structured, transparent, and systematic approaches that offer clear guidance, adaptability, and rigor.

The field of AI is increasingly moving beyond providing isolated algorithmic solutions toward approaches that are embedded in operational systems to generate real-world impact. This requires treating problem solving holistically, integrating accurate problem modeling, robust technical infrastructure, and the generation of explainable insights tailored to diverse users throughout the workflow. Agentic generative AI has a significant potential to enhance EC by automating and coordinating complex tasks across the entire ecosystem. By integrating these

tools into a MAS, the proposed system architecture could streamline optimization model formulation, algorithmic design, efficient computation, and solution interpretation, while enabling dynamic collaboration, human-in-the-loop oversight, and real-time execution. This coordinated approach potentially provides benefits for both researchers and practitioners, reducing the need for problem-specific configurations, implementation, and manual fine-tuning of algorithmic components.

Future developments in EC could also benefit from a stronger reliance on structured and mathematically rigorous algorithmic frameworks, such as those based on reference sets. By emphasizing systematic combination, diversification, and intensification strategies rather than only stochastic variations, these methods can provide a transparent foundation for analyzing and improving search behavior. A shift toward such structured designs could enable a more predictable performance, clearer theoretical understanding, and an easier validation of algorithmic mechanisms, while a hybridization of EC with structured reference set-based methods could leverage the strengths of both approaches to achieve more efficient and reliable outcomes.

In such rapidly evolving environment, staying up to date with technological trends, selecting appropriate LLMs, and choosing suitable frameworks remains critical for fully leveraging agentic generative AI in EC. Future work could focus on systematically exploring its limitations and advancing the automation of EC workflows, while ensuring consistent and reliable results and developing cost-efficient models. Research into specialized SLMs tailored to specific EC domains may further improve performance, reduce computational requirements, and enhance explainability, while also lowering the environmental impact and enabling local deployment to achieve greater independence from external providers. These directions aim to increase the robustness, efficiency, and practical applicability of AI-driven support in real-world optimization tasks handled by EC.

While generative and agentic AI hold great promise for advancing EC, they represent only an initial step toward addressing current challenges, establishing guidelines, playbooks, and best practices, exploring system architecture design patterns, as well as integrating these advances into both the theoretical and practical dimensions of EC. Finally, developing concrete applications, open-source programming libraries, and showcase projects in form of operational systems might be necessary to demonstrate practical value to a broad audience and further foster the adoption of EC.

References

1. Abdel-Rahman, M.J., Alslman, Y., Refai, D., Saleh, A., Loha, M.A.A., Hamed, M.Y.: Mathematical programming through the lens of LLMs: systematic evidence and empirical gaps. *IEEE Access* **13**, 180953–180991 (2025). <https://doi.org/10.1109/ACCESS.2025.3618987>

2. Acharya, D.B., Kuppan, K., Divya, B.: Agentic AI: autonomous intelligence for complex goals—a comprehensive survey. *IEEE Access* **13**, 18912–18936 (2025). <https://doi.org/10.1109/ACCESS.2025.3532853>
3. Ahmed, T., Choudhury, S.: LM4OPT: unveiling the potential of large language models in formulating mathematical optimization problems. *INFOR: Inf. Syst. Oper. Res.* **62**(4), 559–572 (2024). <https://doi.org/10.1080/03155986.2024.2388452>
4. Aishwaryaprajna, Rowe, J.E.: Evolutionary anytime algorithms. In: *Evolutionary Computation in Combinatorial Optimization. EvoCOP 2025, Lecture Notes in Computer Science*, vol. 15610, pp. 18–32. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-86849-8_2
5. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. *IEEE Trans. Evol. Comput.* **6**(5), 443–462 (2002). <https://doi.org/10.1109/TEVC.2002.800880>
6. de Armas, J., Lalla-Ruiz, E., Tilahun, S.L., Voß, S.: Similarity in metaheuristics: a gentle step towards a comparison methodology. *Nat. Comput.* **21**, 265–287 (2022). <https://doi.org/10.1007/s11047-020-09837-9>
7. Bandi, A., Kongari, B., Naguru, R., Pasnoor, S., Vilipala, S.V.: The rise of agentic AI: a review of definitions, frameworks, architectures, applications, evaluation metrics, and challenges. *Future Internet* **17**(9), 404 (2025). <https://doi.org/10.3390/fi17090404>
8. Barman, K.G., Wood, N., Pawlowski, P.: Beyond transparency and explainability: on the need for adequate and contextualized user guidelines for LLM use. *Ethics Inf. Technol.* **26**, 47 (2024). <https://doi.org/10.1007/s10676-024-09778-2>
9. Becattini, M., Verdecchia, R., Vicario, E.: SALLMA: A software architecture for LLM-based multi-agent systems. In: *2025 IEEE/ACM International Workshop New Trends in Software Architecture (SATrends)*, pp. 5–8 (2025). <https://doi.org/10.1109/SATrends66715.2025.00006>
10. Biju, S.M.: Implementing multi-agent systems using LangGraph: A comprehensive study. In: Nayak, R., Mittal, N., Khunteta, A., Kumar, M. (eds.) *Recent Advancements in Artificial Intelligence. ICRAAI 2025*, pp. 269–279. *Lecture Notes in Networks and Systems*, Springer, Cham (2026). https://doi.org/10.1007/978-981-96-7760-3_19
11. Blank, J., Deb, K.: Pymoo: multi-objective optimization in Python. *IEEE Access* **8**, 89497–89509 (2020). <https://doi.org/10.1109/ACCESS.2020.2990567>
12. Bradley, H., Fan, H., Galanos, T., Zhou, R., Scott, D., Lehman, J.: The OpenELM library: leveraging progress in language models for novel evolutionary algorithms. In: Winkler, S., Trujillo, L., Ofria, C., Hu, T. (eds.) *Genetic Programming Theory and Practice XX. Genetic and Evolutionary Computation*, pp. 177–201. Springer, Cham (2024). https://doi.org/10.1007/978-981-99-8413-8_10
13. Camacho-Villalón, C.L., Dorigo, M., Stützle, T.: Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors. *Int. Trans. Oper. Res.* **30**(6), 2945–2971 (2023). <https://doi.org/10.1111/itor.13176>
14. Chen, H., Constante-Flores, G.E., Li, C.: Diagnosing infeasible optimization problems using large language models. *INFOR: Inf. Syst. Oper. Res.* **62**(4), 573–587 (2024). <https://doi.org/10.1080/03155986.2024.2385189>
15. Chen, J., Lin, H., Han, X., Sun, L.: Benchmarking large language models in retrieval-augmented generation. *Proc. AAAI Conf. Artif. Intell.* **38**(16), 17754–17762 (2024). <https://doi.org/10.1609/aaai.v38i16.29728>

16. Cheng, J., Dou, Z., Zhu, Y., Li, X.: Descriptive and discriminative document identifiers for generative retrieval. *Proc. AAAI Conf. Artif. Intell.* **39**(11), 11518–11526 (2025). <https://doi.org/10.1609/aaai.v39i11.33253>
17. Deng, Z., Guo, Y., Han, C., et al.: AI agents under threat: a survey of key security challenges and future pathways. *ACM Comput. Surv.* **57**(7), 1–36 (2025). <https://doi.org/10.1145/3716628>
18. Dorri, A., Kanhere, S.S., Jurdak, R.: Multi-agent systems: a survey. *IEEE Access* **6**, 28573–28593 (2018). <https://doi.org/10.1109/ACCESS.2018.2831228>
19. Dwivedi, R., Dave, D., Naik, H., et al.: Explainable AI (XAI): core ideas, techniques, and solutions. *ACM Comput. Surv.* **55**(9), 1–33 (2023). <https://doi.org/10.1145/3561048>
20. Gardner, B.G., Simon, D.: Evolutionary algorithm sandbox: a web-based graphical user interface for evolutionary algorithms. In: 2009 IEEE International Conference on Systems, Man and Cybernetics, pp. 577–582 (2009). <https://doi.org/10.1109/ICSMC.2009.5346657>
21. Glover, F.: Scatter search and star-paths: beyond the genetic metaphor. *OR Spektrum* **17**, 125–137 (1995). <https://doi.org/10.1007/BF01719256>
22. Glover, F.: Scatter search and path relinking. In: Corne, D., Dorigo, M., Glover, F. (eds.) *New Ideas in Optimization*, pp. 297–316. McGraw-Hill (1999)
23. Glover, F., Hao, J.K.: The case for strategic oscillation. *Ann. Oper. Res.* **183**, 163–173 (2011). <https://doi.org/10.1007/s10479-009-0597-1>
24. Goyal, M., Mahmoud, Q.H.: A systematic review of synthetic data generation techniques using generative AI. *Electronics* **13**(17), 3509 (2024). <https://doi.org/10.3390/electronics13173509>
25. Gu, Y., You, H., Cao, J., Yu, M., Fan, H., Qian, S.: Large language models for constructing and optimizing machine learning workflows: a survey. *ACM Trans. Softw. Eng. Methodol.* (2025). <https://doi.org/10.1145/3773084>
26. He, F., Zhu, T., Ye, D., Liu, B., Zhou, W., Yu, P.S.: The emerged security and privacy of LLM agent: a survey with case studies. *ACM Comput. Surv.* **58**(6), 1–36 (2026). <https://doi.org/10.1145/3773080>
27. Huang, C., Tang, Z., Hu, S., et al.: ORLM: a customizable framework in training large models for automated optimization modeling. *Oper. Res.* **73**(6), 2986–3009 (2025). <https://doi.org/10.1287/opre.2024.1233>
28. Huang, L., Yu, W., Ma, W., et al.: A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.* **43**(2), 1–55 (2025). <https://doi.org/10.1145/3703155>
29. Huang, S., Yang, K., Qi, S., Wang, R.: When large language model meets optimization. *Swarm Evol. Comput.* **90**, 101663 (2024). <https://doi.org/10.1016/j.swevo.2024.101663>
30. Hui, Y., Lu, Y., Zhang, H.: UDA: a benchmark suite for retrieval augmented generation in real-world document analysis. *Adv. Neural Inf. Process. Syst.* **37**, 67200–67217 (2024). <https://doi.org/10.52202/079017-2145>
31. Janssen, D.M., Pullan, W., Liew, A.W.C.: Evolutionary computation visualization: ECvis. *IEEE Access* **11**, 16474–16482 (2023). <https://doi.org/10.1109/ACCESS.2023.3244078>
32. Jovanovic, R., Sanfilippo, A.P., Voß, S.: Fixed set search applied to the clique partitioning problem. *Eur. J. Oper. Res.* **309**(1), 65–81 (2023). <https://doi.org/10.1016/j.ejor.2023.01.044>
33. Jovanovic, R., Voss, S.: The fixed set search applied to the power dominating set problem. *Expert. Syst.* **37**(6), e12559 (2020). <https://doi.org/10.1111/exsy.12559>

34. Ke, W., Zheng, Y., Li, Y., et al.: Large language models in document intelligence: a comprehensive survey, recent advances, challenges, and future trends. *ACM Trans. Inf. Syst.* **44**(1), 1–64 (2026). <https://doi.org/10.1145/3768156>
35. Kim, B.J., Jeong, S., Cho, B.K., Chung, J.B.: AI governance in the context of the EU AI Act. *IEEE Access* **13**, 144126–144142 (2025). <https://doi.org/10.1109/ACCESS.2025.3598023>
36. Kim, G.I., Hwang, S., Jang, B.: Efficient compressing and tuning methods for large language models: a systematic literature review. *ACM Comput. Surv.* **57**(10), 1–39 (2025). <https://doi.org/10.1145/3728636>
37. Ko, E., Kang, P.: Evaluating coding proficiency of large language models: an investigation through machine learning problems. *IEEE Access* **13**, 52925–52938 (2025). <https://doi.org/10.1109/ACCESS.2025.3553870>
38. Kyaw, H.T., Alyafei, K., Dai, X.: Harnessing the power of large language model for natural language-driven task scheduling optimization. In: 2025 30th International Conference on Automation and Computing (ICAC), pp. 1–6 (2025). <https://doi.org/10.1109/ICAC65379.2025.11196207>
39. Li, J., Gao, Y., Yang, Y., et al.: Fundamental capabilities and applications of large language models: a survey. *ACM Comput. Surv.* **58**(2), 1–42 (2026). <https://doi.org/10.1145/3735632>
40. Li, J., Wickman, R., Bhatnagar, S., Maity, R.K., Mukherjee, A.: Abstract operations research modeling using natural language inputs. *Information* **16**(2), 128 (2025). <https://doi.org/10.3390/info16020128>
41. Li, X., Jin, J., Zhou, Y., et al.: From matching to generation: a survey on generative information retrieval. *ACM Trans. Inf. Syst.* **43**(3), 1–62 (2025). <https://doi.org/10.1145/3722552>
42. Liang, X., Gu, Z.: Fast Think-on-graph: wider, deeper and faster reasoning of large language model on knowledge graph. *Proc. AAAI Conf. Artif. Intell.* **39**(23), 24558–24566 (2025). <https://doi.org/10.1609/aaai.v39i23.34635>
43. Ling, C., Zhao, X., Lu, J., et al.: Domain specialization as the key to make large language models disruptive: a comprehensive survey. *ACM Comput. Surv.* **58**(3), 1–39 (2026). <https://doi.org/10.1145/3764579>
44. Liu, F., Tong, X., Yuan, M., et al.: Evolution of heuristics: towards efficient automatic algorithm design using large language model. In: Proceedings of the 41st International Conference on Machine Learning, pp. 32201–32223 (2024)
45. Liu, F., Yao, Y., Guo, P., et al.: A systematic survey on large language models for algorithm design. *ACM Comput. Surv.* **58**(8), 1–32 (2026). <https://doi.org/10.1145/3787585>
46. Liu, L., Hasegawa, S., Sampat, S.K., et al.: AutoDW: automatic data wrangling leveraging large language models. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, pp. 2041–2052 (2024). <https://doi.org/10.1145/3691620.3695267>
47. Liu, S., Chen, C., Qu, X., Tang, K., Ong, Y.S.: Large language models as evolutionary optimizers. In: 2024 IEEE Congress on Evolutionary Computation (CEC), vol. 1, pp. 1–8 (2024). <https://doi.org/10.1109/CEC60901.2024.10611913>
48. López-Pernas, S., Song, Y., Oliveira, E., Saqr, M.: LLMs for explainable artificial intelligence: Automating natural language explanations of predictive analytics models. In: Saqr, M., López-Pernas, S. (eds.) *Advanced Learning Analytics Methods*, pp. 261–286. Springer, Cham (2026). https://doi.org/10.1007/978-3-031-95365-1_11
49. Martí, R., Laguna, M., Glover, F.: Principles of scatter search. *Eur. J. Oper. Res.* **169**(2), 359–372 (2006). <https://doi.org/10.1016/j.ejor.2004.08.004>

50. Martí, R., Sevaux, M., Sörensen, K.: Fifty years of metaheuristics. *Eur. J. Oper. Res.* **321**(2), 345–362 (2025). <https://doi.org/10.1016/j.ejor.2024.04.004>
51. Martín-Santamaría, R., López-Ibáñez, M., Stützle, T., Colmenar, J.M.: On the automatic generation of metaheuristic algorithms for combinatorial optimization problems. *Eur. J. Oper. Res.* **318**(3), 740–751 (2024). <https://doi.org/10.1016/j.ejor.2024.06.001>
52. McAvinue, S., Dev, K.: Comparative evaluation of large language models using key metrics and emerging tools. *Expert. Syst.* **42**(2), e13719 (2025). <https://doi.org/10.1111/exsy.13719>
53. Mo, F., Mao, K., Zhao, Z., et al.: A survey of conversational search. *ACM Trans. Inf. Syst.* **43**(6), 1–50 (2025). <https://doi.org/10.1145/3759453>
54. Mostajabdaveh, M., Yu, T.T., Ramamonjison, R., Carenini, G., Zhou, Z., Zhang, Y.: Optimization modeling and verification from problem specifications using a multi-agent multi-stage LLM framework. *INFOR: Inf. Syst. Oper. Res.* **62**(4), 599–617 (2024). <https://doi.org/10.1080/03155986.2024.2381306>
55. Naveed, H., Khan, A.U., Qiu, S., et al.: A comprehensive overview of large language models. *ACM Trans. Intell. Syst. Technol.* **16**(5), 1–72 (2025). <https://doi.org/10.1145/3744746>
56. Ofoghi, B., Yearwood, J.: Knowledge representation of mathematical optimization problems and constructs for modeling. *Knowl.-Based Syst.* **280**, 110980 (2023). <https://doi.org/10.1016/j.knosys.2023.110980>
57. Pan, H., Zhang, Q., Adamu, M., Dragut, E., Latecki, L.J.: Taxonomy-driven knowledge graph construction for domain-specific scientific applications. In: Findings of the Association for Computational Linguistics: ACL 2025, pp. 4295–4320 (2025). <https://doi.org/10.18653/v1/2025.findings-acl.223>
58. Pluhacek, M., Kazikova, A., Kadavy, T., Viktorin, A., Senkerik, R.: Leveraging large language models for the generation of novel metaheuristic optimization algorithms. In: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, pp. 1812–1820 (2023). <https://doi.org/10.1145/3583133.3596401>
59. Polak, M.P., Morgan, D.: Extracting accurate materials data from research papers with conversational language models and prompt engineering. *Nat. Commun.* **15**, 1569 (2024). <https://doi.org/10.1038/s41467-024-45914-8>
60. Qi, F., Wang, T., Zheng, R., Li, M.: A memetic and reflective evolution framework for automatic heuristic design using large language models. *Appl. Sci.* **15**(15), 8735 (2025). <https://doi.org/10.3390/app15158735>
61. Qin, Y., Hu, S., Lin, Y., et al.: Tool learning with foundation models. *ACM Comput. Surv.* **57**(4), 1–40 (2025). <https://doi.org/10.1145/3704435>
62. Radanliev, P., Atefi, K., Santos, O., Maple, C.: Integrating agentic risk signalling in trusted research environments: automating VEX with Agent2Agent protocols and model context protocol (MCP) in SACRO and TRevolution pipelines. *Comput. Stan. Interfaces* **96**, 104079 (2026). <https://doi.org/10.1016/j.csi.2025.104079>
63. Romera-Paredes, B., Barekatin, M., Novikov, A., et al.: Mathematical discoveries from program search with large language models. *Nature* **625**(7995), 468–475 (2024). <https://doi.org/10.1038/s41586-023-06924-6>
64. Sartori, C.C., Blum, C., Bistaffa, F., Rodríguez Corominas, G.: Metaheuristics and large language models join forces: toward an integrated optimization approach. *IEEE Access* **13**, 2058–2079 (2025). <https://doi.org/10.1109/ACCESS.2024.3524176>

65. Senkerik, R., Viktorin, A., Kadavy, T., et al.: Open and closed source models for LLM-generated metaheuristics solving engineering optimization problem. In: Applications of Evolutionary Computation. EvoApplications 2025, Lecture Notes in Computer Science, vol. 15613, pp. 372–385. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-90065-5_23
66. Shi, J., Yuan, Z., Liu, Y., et al.: Optimization-based prompt injection attack to LLM-as-a-judge. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, pp. 660–674 (2024). <https://doi.org/10.1145/3658644.3690291>
67. Shi, X., Liu, J., Liu, Y., Cheng, Q., Lu, W.: Know where to go: make LLM a relevant, responsible, and trustworthy searchers. *Decis. Support Syst.* **188**, 114354 (2025). <https://doi.org/10.1016/j.dss.2024.114354>
68. Sim, K., Renaud, Q., Hart, E.: Beyond the hype: Benchmarking LLM-evolved heuristics for bin packing. In: Applications of Evolutionary Computation. EvoApplications 2025, Lecture Notes in Computer Science, vol. 15613, pp. 386–402. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-90065-5_24
69. Smith-Miles, K., Muñoz, M.A.: Instance space analysis for algorithm testing: methodology and software tools. *ACM Comput. Surv.* **55**(12), 1–31 (2022). <https://doi.org/10.1145/3572895>
70. Sörensen, K.: Metaheuristics-the metaphor exposed. *Int. Trans. Oper. Res.* **22**(1), 3–18 (2015). <https://doi.org/10.1111/itor.12001>
71. van Stein, N., Bäck, T.: LLaMEA: a large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Trans. Evol. Comput.* **29**(2), 331–345 (2025). <https://doi.org/10.1109/TEVC.2024.3497793>
72. Swan, J., Adriaensen, S., Brownlee, A.E.I., et al.: Metaheuristics “in the large.” *Eur. J. Oper. Res.* **297**(2), 393–406 (2022). <https://doi.org/10.1016/j.ejor.2021.05.042>
73. Takagi, H., Moriya, S., Sato, T., Nagao, M., Higuchi, K.: A framework for efficient development and debugging of role-playing agents with large language models. In: Proceedings of the 30th International Conference on Intelligent User Interfaces, pp. 70–88 (2025). <https://doi.org/10.1145/3708359.3712119>
74. Tian, S., Xing, S., Li, X., et al.: A systematic exploration of knowledge graph alignment with large language models in retrieval augmented generation. *Proc. AAAI Conf. Artif. Intell.* **39**(24), 25291–25299 (2025). <https://doi.org/10.1609/aaai.v39i24.34716>
75. Tzanetos, A., Dounias, G.: Nature inspired optimization algorithms or simply variations of metaheuristics? *Artif. Intell. Rev.* **54**, 1841–1862 (2021). <https://doi.org/10.1007/s10462-020-09893-8>
76. Vaidhyanathan, K., Muccini, H.: Software architecture in the age of agentic AI. In: Bianculli, D., et al. (eds.) *Software Architecture. ECSA 2025 Tracks and Workshops. ECSA 2025, Lecture Notes in Computer Science*, vol. 15982, pp. 41–49. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-04403-7_5
77. Voß, S.: Bus bunching and bus bridging: what can we learn from generative AI tools like ChatGPT? *Sustainability* **15**(12), 9625 (2023). <https://doi.org/10.3390/su15129625>
78. Voß, S., Woodruff, D.L. (eds.): *Optimization software class libraries*, Operations Research/Computer Science Interfaces Series, vol. 18. Springer, Cham (2002). <https://doi.org/10.1007/b101931>
79. Wang, C.H., Hu, K., Wu, X., Ou, Y.: Rethinking metaheuristics: unveiling the myth of “novelty” in metaheuristic algorithms. *Mathematics* **13**(13), 2158 (2025). <https://doi.org/10.3390/math13132158>

80. Wang, C., Liu, X., Yue, Y., et al.: Survey on factuality in large language models. *ACM Comput. Surv.* **58**(1), 1–37 (2026). <https://doi.org/10.1145/3742420>
81. Wang, D., Zhang, Z., Teng, Y.: Large language model implemented simulated annealing algorithm for traveling salesman problem. In: 2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 209–214 (2024). <https://doi.org/10.1109/SMC54092.2024.10831145>
82. Wang, F., Zhang, Z., Zhang, X., et al.: A comprehensive survey of small language models in the era of large language models: techniques, enhancements, applications, collaboration with LLMs, and trustworthiness. *ACM Trans. Intell. Syst.* **16**(6), 1–87 (2025). <https://doi.org/10.1145/3768165>
83. Wang, Q., Wang, T., Tang, Z., et al.: MegaAgent: a large-scale autonomous LLM-based multi-agent system without predefined SOPs. In: Findings of the Association for Computational Linguistics: ACL 2025, pp. 4998–5036 (2025). <https://doi.org/10.18653/v1/2025.findings-acl.259>
84. Wang, S., Zhu, T., Liu, B., et al.: Unique security and privacy threats of large language models: a comprehensive survey. *ACM Comput. Surv.* **58**(4), 1–36 (2026). <https://doi.org/10.1145/3764113>
85. Wang, Y., Wang, J., Chu, Z.: Multi-agent large language models as evolutionary optimizers for scheduling optimization. *Comput. Ind. Eng.* **206**, 111197 (2025). <https://doi.org/10.1016/j.cie.2025.111197>
86. Wasserkrug, S., Boussioux, L., Den Hertog, D., et al.: Enhancing decision making through the integration of large language models and operations research optimization. *Proc. AAAI Conf. Artif. Intell.* **39**(27), 28643–28650 (2025). <https://doi.org/10.1609/aaai.v39i27.35090>
87. Weng, L., Wang, X., Lu, J., et al.: InsightLens: augmenting LLM-powered data analysis with interactive insight management and navigation. *IEEE Trans. Visual Comput. Graph.* **31**(6), 3719–3732 (2025). <https://doi.org/10.1109/TVCG.2025.3567131>
88. Wu, X., Wu, S.H., Wu, J., Feng, L., Tan, K.C.: Evolutionary computation in the era of large language model: survey and roadmap. *IEEE Trans. Evol. Comput.* **29**(2), 534–554 (2025). <https://doi.org/10.1109/TEVC.2024.3506731>
89. Xu, M., Cai, D., Yin, W., Wang, S., Jin, X., Liu, X.: Resource-efficient algorithms and systems of foundation models: a survey. *ACM Comput. Surv.* **57**(5), 1–39 (2025). <https://doi.org/10.1145/3706418>
90. Yang, E., Shen, L., Guo, G., et al.: Model merging in LLMs, MLLMs, and beyond: methods, theories, applications, and opportunities. *ACM Comput. Surv.* **58**(8), 1–41 (2026). <https://doi.org/10.1145/3787849>
91. Yang, Z., Shi, J., Devanbu, P., Lo, D.: Ecosystem of large language models for code. *ACM Trans. Softw. Eng. Methodol.* **35**(1), 1–30 (2025). <https://doi.org/10.1145/3731753>
92. Yao, L., Ren, F., Du, K., Du, Q.: From knowledge graph construction to retrieval-augmented generation: a framework for comprehensive earthquake emergency support. *Geo-Spatial Inf. Sci.* **29**(1), 509–529 (2025). <https://doi.org/10.1080/10095020.2025.2514813>
93. Ye, H., Wang, J., Cao, Z., et al.: ReEvo: Large language models as hyper-heuristics with reflective evolution. *Adv. Neural Inf. Process. Syst.* **37**, 43571–43608 (2024). <https://doi.org/10.52202/079017-1381>
94. Yin, H., Kononova, A.V., Bäck, T., van Stein, N.: Controlling the mutation in large language models for the efficient evolution of algorithms. In: Applications of Evolutionary Computation. *EvoApplications 2025, Lecture Notes in Computer*

- Science, vol. 15613, pp. 403–417. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-90065-5_25
95. Yu, C., Cheng, Z., Cui, H., et al.: A survey on agent workflow – Status and future. In: 2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD), pp. 770–781 (2025). <https://doi.org/10.1109/ICAIBD64986.2025.11082076>
 96. Zhang, X., Li, M., Wu, J.: Co-occurrence is not factual association in language models. *Adv. Neural Inf. Process. Syst.* vol. 37, pp. 64889–64914 (2024). <https://doi.org/10.52202/079017-2071>
 97. Zhao, T., Chen, J., Ru, Y., et al.: Exploring knowledge poisoning attacks to retrieval-augmented generation. *Inf. Fusion* **127**, 103900 (2026). <https://doi.org/10.1016/j.inffus.2025.103900>
 98. Zheng, Y., Chen, Y., Qian, B., Shi, X., Shu, Y., Chen, J.: A review on edge large language models: design, execution, and applications. *ACM Comput. Surv.* **57**(8), 1–35 (2025). <https://doi.org/10.1145/3719664>
 99. Zhong, R., Xu, Y., Zhang, C., Yu, J.: Leveraging large language model to generate a novel metaheuristic algorithm with CRISPE framework. *Clust. Comput.* **27**(10), 13835–13869 (2024). <https://doi.org/10.1007/s10586-024-04654-6>
 100. Zhuang, F., Qi, Z., Duan, K., et al.: A comprehensive survey on transfer learning. *Proc. IEEE* **109**(1), 43–76 (2021). <https://doi.org/10.1109/JPROC.2020.3004555>