AI4AI Toward AI-discovered Superintelligence

Siyuan Liu[1,2†] Yifei Wang[3†] Zheng Hu[2†] Tong Hai[2,4] Ruiqing Ye[2] Jinghan Yang[5]

Jiamin Shi[6] Zhiqin Yang[7] Mingyuan Meng[2] Chang Liu[2] Tao Li[1]

Jianfei Yang[5] He Zhang[2✉] Nanning Zheng[3✉] Hongkun Dou[2✉] Jingwen Fu[2✉]

[1] Nankai University [2] Zhongguancun Academy [3] Xi'an Jiaotong University

[4] Shanghai Jiao Tong University [5] Nanyang Technological University

[6] National University of Singapore [7] The Hong Kong University of Science and Technology

[†] *Core contribution*, [✉] *Corresponding authors*

Contact: Siyuan Liu (s-lsyuan25@bza.edu.cn)

Artificial intelligence (AI) is reshaping how AI itself is developed. From designing learning algorithms to automating research workflows and evaluation, this expanding role has fueled the rapid growth of AI for AI (AI4AI). Yet a fundamental question remains: as AI takes on more research tasks, which decisions does it actually control? Even systems that execute entire research workflows may still rely on humans to define research goals, allowable modifications, and criteria for success. Existing taxonomies, typically organized by technique or target artifact, offer limited insight into these differences in research responsibility. This survey presents a unified, control-centered perspective on AI4AI, viewing its evolution as a progressive transfer of control over three key decisions in the discovery loop: search direction, search space, and evaluation signal. We introduce an L0–L3 control-source framework, complemented by full responsibility profiles, to characterize different forms of human–AI research responsibility without conflating control with overall capability or end-to-end autonomy. This perspective unifies research across AI4AI Learning, Inference, Data, and Infrastructure, while motivating an evaluation framework that goes beyond downstream performance to assess decision quality, discovery efficiency, transfer, evaluator independence, and robustness under optimization. Looking further ahead, AI4AI may enable recursive self-improvement, where advances in AI enhance the process of making subsequent discoveries. If sustained and independently validated, such improvements could offer a pathway toward superintelligence. By connecting research autonomy with the conditions for reliable and cumulative progress, this survey provides a unified foundation for understanding the evolution and long-term potential of AI4AI.

🎯 **Keywords**: AI for AI, Automated AI Research, Research Autonomy.
Github: https://ai4ai-research.github.io

Contents

1 Introduction 3

2 Key Concepts and Related Surveys 5
2.1 Key Concepts . . . 5
2.2 Related Surveys . . . 7

3 Preliminaries 9
3.1 The AI4AI Discovery Process . . . 9
3.2 Search-Space Provision . . . 10
3.3 Direction Selection within a Search Space . . . 11
3.4 Evaluation as the Feedback Signal . . . 11
3.5 Responsibility Profiles and Control Strength . . . 12
3.6 The Control-Source Ladder, Assignment, and Reporting . . . 13

4 L1: AI Provides the Search Direction 14
4.1 History-Conditioned Direction Selection . . . 15
4.2 Evolutionary Program Improvement . . . 17
4.3 Prompt, Pipeline, and Workflow Optimization . . . 19
4.4 Reflection, Memory, and Search over Trajectories . . . 20
4.5 Experiment-Driven Research Direction . . . 21
4.6 Direction Quality, Failure Modes, and the Boundary with L2 . . . 22

5 L2: AI Provides the Search Space 23
5.1 Formalization and evaluation of AI-defined search spaces . . . 23
5.2 Evaluation and Minimum Evidence for L2 . . . 27
5.3 Composition with Direction and Evaluation Control . . . 28

6 L3: AI Provides the Evaluation Signal 29
6.1 The Object of Attribution: the Operative Evaluation Signal . . . 29
6.2 Executable Evaluation Artifacts . . . 31
6.3 Learned Preference Evaluation and Model Judges . . . 32
6.4 Generated Objectives and Self-Evaluation . . . 34
6.5 Adaptive Oversight, Robustness, and Recursive Evidence . . . 34

7 Applications and Benchmarks 36
7.1 AI4AI Learning . . . 36
7.2 AI4AI Inference . . . 40
7.3 AI4AI Data . . . 42
7.4 AI4AI Infrastructure . . . 44
7.5 Research Workflows and Shared Evaluation . . . 45

8 Open Questions and Future Works 49
8.1 Open Questions . . . 49
8.2 Future Work . . . 51

9 Conclusion and Limitations 53

1. Introduction

The development of artificial intelligence (AI) has traditionally been guided by human researchers. They decide what kinds of solutions to explore, which experiments to conduct, and how to evaluate the results. As AI systems and design spaces grew more complex, this fully human-directed process became a scalability bottleneck: too many consequential research decisions still depended on limited human attention and manual experimentation.

Early automation began to relieve this bottleneck by automating candidate search, but it did not fundamentally change who controlled the research process. Hyperparameter optimization, AutoML, and neural architecture search replaced manual trial-and-error with systematic exploration of model configurations, development pipelines, and architectures [19, 230, 71, 104, 336, 210]. Subsequent advances extended automation beyond model selection to the learning process itself, enabling systems to adapt training schedules, learn optimization rules, and even discover complete learning algorithms from primitive operations [106, 10, 211]. More recently, systems such as FunSearch and AlphaEvolve have demonstrated the discovery and improvement of executable algorithms and programs [214, 190], while language-model-based approaches have made prompts and agent architectures objects of automated optimization [204, 69, 100]. Across these developments, the scope of automation has expanded from selecting numerical configurations to modifying increasingly complex artifacts. Yet this growing flexibility does not necessarily imply greater research autonomy: humans may still determine the problems to solve, the boundaries of allowable modifications, and the criteria for success.

Automation is now extending beyond individual AI artifacts to the research process itself. Systems such as The AI Scientist and The AI Scientist-v2 integrate idea generation, experimentation, and reporting into increasingly automated research workflows [158, 292]. In parallel, AI is taking a more active role in evaluation, generating feedback that guides subsequent improvements. Constitutional AI and RLAIF use model-generated critiques or preferences to guide training [15, 126]; Eureka synthesizes executable reward functions [168]; and self-rewarding language models use their own judgments to guide further training [303]. Although these approaches often operate under human-defined objectives, AI-generated feedback can influence which candidates are selected and how models are improved. This growing role creates opportunities for more adaptive discovery, while underscoring the importance of independent validation, as model-generated judgments can be biased and improvements under imperfect reward signals may not translate into genuine progress [322, 265, 80]. Together, these developments show that automating a research workflow and controlling its key decisions are fundamentally different. An AI system may execute an entire research workflow while humans still determine which problems to pursue, what can be modified, and what counts as success. Conversely, AI may construct or substantively revise the evaluation signal that governs candidate selection, even when humans carry out the experiments and implement the changes. Research autonomy should therefore be characterized by which decisions AI makes or revises, rather than by how many research tasks it executes.

These developments reflect the growing field of *AI for AI* (**AI4AI**), in which AI contributes to the development and improvement of AI systems themselves. Yet despite its rapid expansion, how control over the discovery process is distributed between humans and AI remains insufficiently characterized. To address this gap, we view AI development as a discovery loop in which candidate improvements are proposed, evaluated, and selected or revised based on feedback. We distinguish three key decisions in this loop: search direction (D), which determines which avenues to pursue; search space (S), which defines what can be changed; and evaluation signal (E), which determines what counts as progress.

Existing taxonomies typically organize AI4AI research by technique or target artifact. While useful for mapping the field, they do not fully capture how responsibility for these decisions is divided between humans

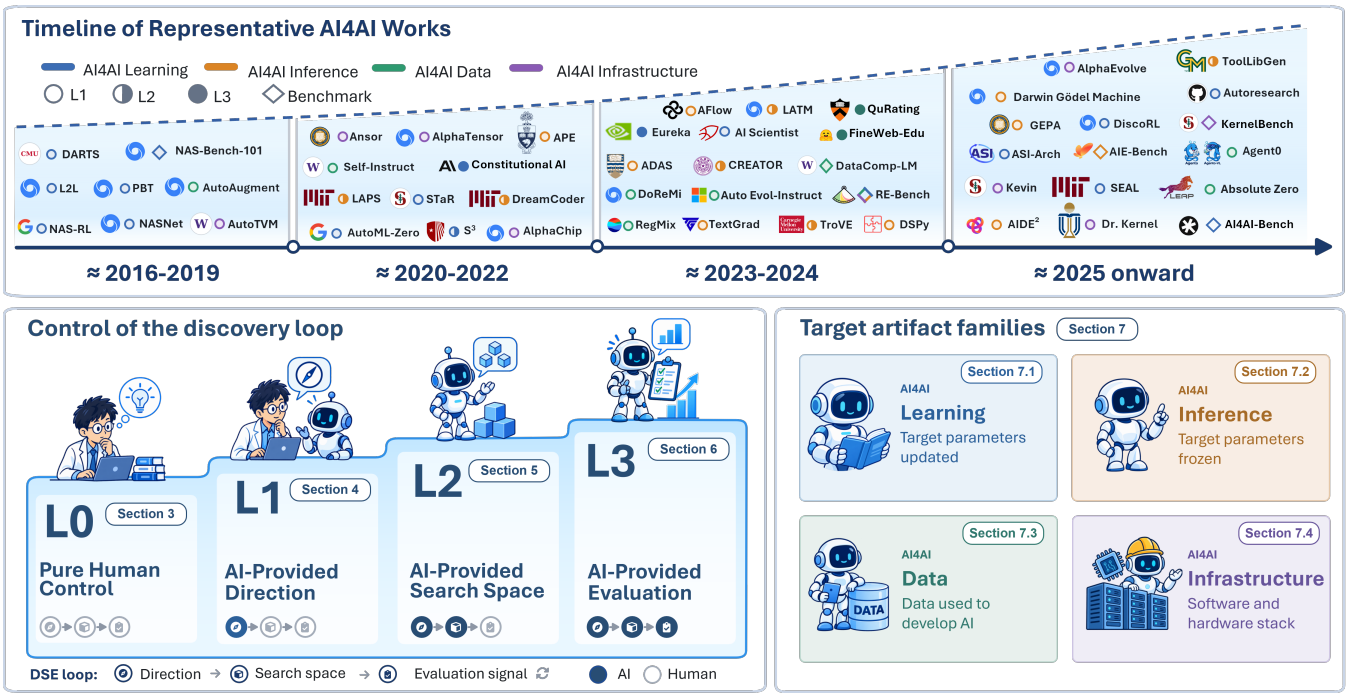


Figure 1: The progressive transfer of control over the AI4AI discovery loop.

and AI. Indeed, systems that improve similar AI artifacts may differ fundamentally in which decisions they control. This motivates the central perspective of our survey:

> *We view the evolution of AI4AI as a progressive shift in control over the discovery loop, from humans deciding how to explore, where to explore, and what counts as progress, toward AI systems increasingly making or revising these decisions.*

Identifying these three decisions is only the first step; understanding research autonomy also requires determining who actually controls them. Generating candidate solutions does not mean deciding where to explore, just as running an evaluation does not mean defining what counts as success. We therefore attribute control to AI only when it originates or substantively revises a decision that shapes the subsequent discovery process.

To characterize how these responsibilities are distributed between humans (H) and AI (A), we assign each system a responsibility profile:

$$\mathbf{a} = \langle a_D, a_S, a_E \rangle, \qquad a_i \in \{\mathrm{H}, \mathrm{A}\}, \tag{1}$$

where the coordinates indicate whether each decision is human-specified or externally fixed, or substantively AI-provided. Based on the responsibility profile, we distinguish four control-source levels, where $* \in \{\mathrm{H}, \mathrm{A}\}$ denotes either source:

- **L0** ($\langle \mathrm{H}, \mathrm{H}, \mathrm{H} \rangle$): Humans provide all three decisions, serving as the reference case.
- **L1** ($\langle \mathrm{A}, \mathrm{H}, \mathrm{H} \rangle$): AI provides the search direction, while the search space and evaluation remain human-defined.
- **L2** ($\langle *, \mathrm{A}, \mathrm{H} \rangle$): AI constructs or substantively revises the search space, while evaluation remains human-defined.

- **L3** ($\langle *, *, \mathrm{A} \rangle$): AI provides or substantively revises the evaluation signal, regardless of who determines direction or space.

Levels are assigned by the precedence $E > S > D$, which determines the classification label rather than implying cumulative autonomy or greater capability. The level label provides a concise classification, while the full responsibility profile preserves how control over direction, space, and evaluation is distributed between humans and AI.

The ladder captures the source of research control rather than the extent of automation or a system's overall capability. For example, FunSearch and AlphaEvolve guide program search using evaluation histories while operating within human-defined edit boundaries and evaluators [214, 190]. Similarly, The AI Scientist and the Darwin Gödel Machine automate research or self-revision processes without necessarily controlling the underlying search space or success criteria [158, 312]. These distinctions matter because greater control does not guarantee better outcomes: AI may pursue ineffective directions, construct unproductive search spaces, or generate evaluation signals that reward misleading improvements. Independent validation is therefore essential, especially when AI helps determine what counts as progress.

Our main contributions are threefold. First, we synthesize AI4AI research through the lens of control over discovery decisions, offering a unified perspective on techniques and applications that are otherwise fragmented. Second, we formalize this perspective in a Direction–Space–Evaluation framework that pairs L0–L3 control-source levels with full responsibility profiles, distinguishing the different forms of human–AI research responsibility without conflating them with end-to-end autonomy. Third, we identify key open questions and outline promising directions for future research.

2. Key Concepts and Related Surveys

2.1. Key Concepts

AI4AI. Artificial intelligence for artificial intelligence (AI4AI) refers to the use of AI models or feedback-driven search to discover and improve AI systems and their development processes. Within the broader landscape of AI for Science, AI4AI specifically targets the advancement of AI itself. A defining requirement is that AI substantively contributes to discovery rather than merely executing a predetermined workflow.

Automated Machine Learning. Automated machine learning (AutoML) automates the optimization of model configurations, architectures, learning algorithms, and training pipelines, typically within human-defined search spaces and evaluation criteria [104, 62, 93]. It provides an important foundation for AI4AI by replacing manual design choices with systematic search.

AI Scientists. AI scientists extend automation from individual optimization tasks to broader research processes, including hypothesis generation, experiment design, implementation, analysis, and reporting [164, 324, 246]. Such systems fall within AI4AI when their research aims to improve AI systems or the processes used to develop them.

Self-Evolving AI. Self-evolving AI systems improve their own parameters, instructions, memories, tools, or behaviors through feedback and accumulated experience [241, 79, 66, 213]. However, self-modification

does not necessarily imply control over the improvement process, as humans may still define the objectives, allowable changes, and evaluation criteria.

Recursive Self-Improvement. Recursive self-improvement (RSI) refers to a process in which improvements to an AI system enhance its ability to produce further improvements [31]. Unlike repeated optimization of a fixed target, RSI requires improvements to the discovery mechanism itself. Demonstrating RSI therefore requires sustained gains in subsequent improvement cycles under independent evaluation. Such gains may provide a path toward superintelligence, although sustained recursive improvement remains an open challenge.

Search Direction. The search direction (D) determines which possibilities to pursue within a search space, such as which bottleneck to address, hypothesis to test, or candidate region to explore. It guides the allocation of exploration effort based on research objectives and previous results.

Search Space. The search space (S) defines the set of admissible solutions through editable components, candidate representations, available operations, and constraints. It determines what can be explored, while generating candidates within a predefined space does not itself constitute control over that space.

Evaluation Signal. The evaluation signal (E) determines what counts as progress through metrics, tests, rewards, or acceptance criteria that guide candidate selection. Applying a predefined evaluation rule does not imply control over the criteria themselves. Together, D, S, and E provide three distinct dimensions for characterizing how control over AI discovery is distributed between humans and AI.

Figure 2 brings together these three control variables and shows how evidence connects candidate generation, evaluation, updating, and iterative improvement.

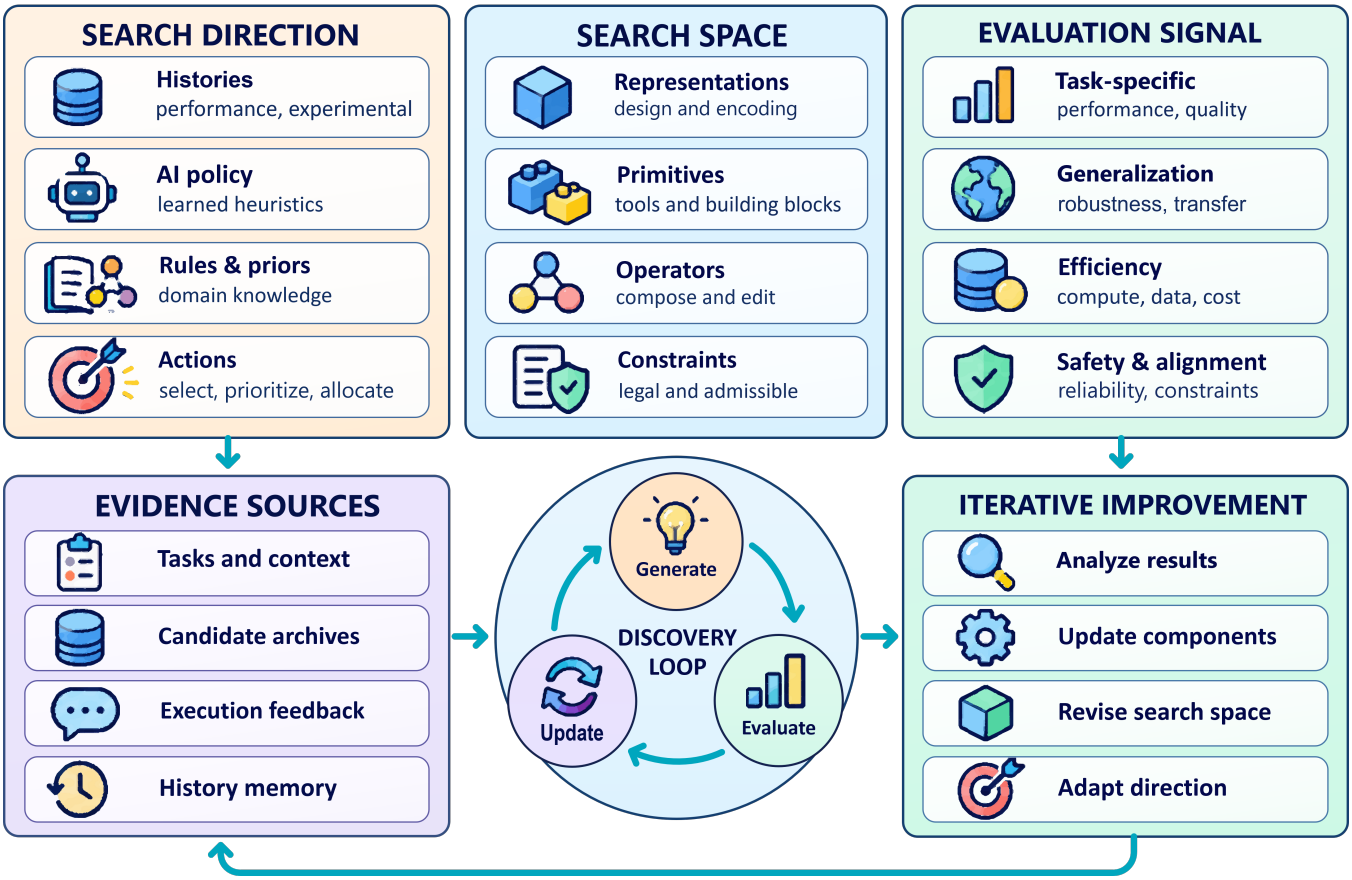


Figure 2: AI4AI control framework. The upper band separates the three control variables—direction, search space, and evaluation—while the lower discovery loop connects evidence sources to candidate generation, evaluation, updating, and iterative improvement. The discovery process is formalized in Section 3; the source of each variable is assigned separately by the L0–L3 responsibility framework.

2.2. Related Surveys

Existing surveys offer detailed accounts of automated optimization, evolving agents, autonomous research, and the artifacts produced by these systems. However, these research traditions focus on different aspects of AI-driven progress. Table 1 groups existing surveys into five lenses accordingly. We summarize how each lens understands the field, identify the limitation they share, and explain the different starting point of this survey.

Perspectives in existing surveys. Search-centric surveys, spanning AutoML, neural architecture search, meta-learning, and LLM-based algorithm design, organize methods around the representations being searched, the optimization strategy, and the procedure for estimating candidate quality [62, 99, 144]. This decomposition is valuable for explaining how search works, but does not necessarily track whether those components are specified by humans or constructed by AI. Self-evolution surveys instead ask what can be updated, when adaptation occurs, and which feedback mechanisms support it [79, 66, 213]. Research-lifecycle surveys emphasize the integration of ideation, experimentation, review, and reporting into increasingly automated workflows [324, 246, 245]. Component-oriented reviews examine particular technologies, such

Table 1: Survey perspectives on AI-built AI.

Survey lens	Primary unit	Discovery control			Cross-domain coverage
		D	*S*	*E*	
Search-centric [104, 93, 62, 272, 99, 197, 249, 88, 144, 281, 208, 304]	Search procedure	●	◆	○	◆
Self-evolution [241, 79, 66, 55, 213]	Model or agent	●	◆	◆	◆
Research lifecycle [164, 33, 324, 266, 246, 245, 118, 52]	Research stage or workflow	●	○	◆	○
Component and artifact [87, 129, 26, 9, 157, 300, 326, 262]	Target artifact	○	○	◆	○
Loop closure [31, 182, 68]	Capability, autonomy, or improvement loop	◆	○	●	◆
This survey	Discovery control	●	●	●	●

● central to the lens; ◆ partial or present in only some surveys; ○ absent.
D, *S*, and *E* indicate whether the lens analyzes AI control over search direction, search space, and evaluation signal, respectively. Cross-domain coverage refers to learning, inference, data, and infrastructure within one framework.

as model judges, data selection, and electronic design automation [87, 9, 326]. Finally, work on recursive improvement and levels of intelligence or agent autonomy offers broader conceptual accounts of independence and capability [31, 182, 68]. These perspectives address distinct, legitimate questions; their limitation for our purpose is the absence of a common unit for attributing consequential research decisions across all four AI4AI artifact families. The perspectives differ in axis but share one limitation: they describe what AI *does* rather than what it *decides*. As Table 1 shows, each lens brings at most one of the three decisions in the AI4AI discovery loop into focus.

Positioning of this survey. We begin from a different premise on who holds the authority to decide. A discovery process depends on three decisions: which directions to explore, what solutions can be considered, and how progress is evaluated. Existing surveys typically organize AI4AI systems by their techniques, outputs, or workflows. While useful, these approaches often overlook changes in who controls the research process. We therefore view the development of AI4AI through the transfer of three key decisions from humans to AI: search direction, search space, and evaluation signal. These transfers represent a progression in research control: from choosing what to explore, to defining what can be explored, and ultimately to determining what counts as progress. This perspective also provides a common way to compare different AI4AI systems. A hyperparameter optimizer, a kernel-search agent, and an AI scientist can all be assessed by the decisions they control, regardless of their underlying techniques. Claims of research autonomy should therefore be supported by evidence of who makes each decision, rather than by how much of the research workflow is automated.

3. Preliminaries

This section establishes the formal framework for analyzing and classifying AI4AI systems. We first formalize the discovery process, then examine the provenance of its three control decisions: search direction (D), search space (S), and evaluation signal (E). Based on these decisions, we introduce responsibility profiles and a protocol for assigning L0–L3 levels. The levels follow a classification precedence of $E > S > D$, rather than representing an ordinal scale of intelligence, capability, or autonomy. Table 2 summarizes the notation.

3.1. The AI4AI Discovery Process

We assess control provenance at the level of individual system configurations, rather than assigning fixed control responsibilities to models or methods. Each configuration is characterized by its target AI artifact, search procedure, editable boundaries, available tools, evaluator, and human-intervention policy. This scope encompasses classical AutoML and neural architecture search [104, 62], as well as more recent AI4AI applications such as visual-token pruning and test-time capability transfer [154, 207].

Under our operational criterion, a system qualifies as AI4AI when an AI model or feedback-adaptive search procedure originates or substantively revises at least one control decision (D, S, or E), and that decision causally influences candidate generation, selection, updating, or acceptance. Merely executing a predefined workflow or applying a fixed evaluation rule does not, by itself, establish such control. Consequently, the same model or method may assume different control responsibilities depending on its specific configuration.

Three boundary conditions follow. Generic AI-assisted coding is included only when the generated code changes the target AI artifact or discovery procedure; code that merely implements a human-fixed design is excluded. AI for science is adjacent but not automatically AI4AI: a mathematical or scientific discovery enters scope only when the resulting artifact or discovery mechanism materially contributes to AI development. Automated experimentation is likewise insufficient when the experiment plan, editable objects, and acceptance criteria remain human-authored. Benchmarks for algorithmic design are especially informative because they test whether an agent can alter learning procedures rather than only tune familiar parameters [40]. The unit of analysis is a *demonstrated configuration*: the model and software versions, prompts, tools, data, search budget, evaluator, orchestration, random seeds, and human-intervention policy used in a particular result. The same model can therefore receive different profiles in different configurations. A language model may construct an evaluator in one loop but merely execute a human rubric in another. We consequently do not assign an intrinsic level to a foundation model, method name, or paper title.

Let x_t denote the target AI system at discovery cycle t, h_t the accumulated research history, and B_t the available resource budget. A discovery cycle is written as

$$S_t = P_S(x_t, h_t), \tag{2}$$
$$D_t = P_D(x_t, h_t, S_t), \tag{3}$$
$$\mathcal{C}_t = G_t(S_t, D_t; B_t) = \{c_{t,i}\}_{i=1}^{n_t}, \tag{4}$$
$$q_{t,i} = E_t(c_{t,i}, x_t, S_t, D_t), \quad i = 1, \ldots, n_t, \tag{5}$$
$$\mathbf{q}_t = (q_{t,1}, \ldots, q_{t,n_t}), \tag{6}$$
$$x_{t+1} = U_t(x_t, \mathcal{C}_t, \mathbf{q}_t), \tag{7}$$
$$h_{t+1} = H_t(h_t, \mathcal{C}_t, \mathbf{q}_t). \tag{8}$$

The procedure P_S first specifies the editable objects, admissible candidates, constraints, and operators. Conditioned on S_t, P_D determines how the system should search inside that space. The generator G_t realizes

Table 2: Notation used for the AI4AI discovery process and responsibility framework.

Symbol	Meaning
x_t	Target AI system at discovery cycle t
h_t	Accumulated research history available at cycle t
B_t	Resource budget for candidate generation and evaluation
S_t	Search space: editable objects, representations, constraints, and operators
$\mathcal{K}_t$	Validity or admissibility constraints defining the active search space
D_t	Search direction selected within S_t
$\mathcal{C}_t$	Realized set of candidate interventions
n_t	Number of candidates in $\mathcal{C}_t$
E_t, $\mathbf{q}_t$	Evaluation procedure and the candidate-level evidence vector it produces
$q_{t,i}$	Evidence assigned to candidate $c_{t,i}$
U_t	Target-system update rule that selects, aggregates, or implements an intervention
H_t	History update rule that records candidates and evidence for later cycles
$\mathbf{a}$	Binary responsibility profile $\langle a_D, a_S, a_E \rangle$
$\mathbf{r}$	Descriptive control-strength vector $\langle r_D, r_S, r_E \rangle$

a candidate set $\mathcal{C}_t$, the evaluator E_t produces candidate-level evidence $\mathbf{q}_t$, the updater U_t executes the selection or update implied by that evidence, and H_t records the history used by later cycles. Selection, aggregation, thresholds, and stopping rules that determine acceptance belong to the evaluation signal; U_t only executes those decisions. Resource allocation within a fixed evaluation protocol may be part of D_t, whereas a learned or revised pruning, acceptance, or stopping criterion is part of E_t. If an approval or update rule in U_t changes operative acceptance, that change is recorded as part of E or as residual human control. Each procedure may be implemented by people, algorithms, learned models, or hybrids.

For the schematic batch cycle used here, the main dependency is

$$S \longrightarrow D \longrightarrow \mathcal{C} \longrightarrow E. \tag{9}$$

This arrow is a batch-processing abstraction that separates the space, direction, candidate, and evaluation roles; it is not a strict temporal law for every system. Online systems may instead revise S_t and D_t alternately as new evidence arrives. Each procedure may be human-provided, AI-provided, or jointly specified, and the source of one procedure does not determine the sources of the others. Hyperparameter optimization can generate thousands of configurations under a human-defined task, domain, and metric [19, 230, 71]; program search can produce novel solutions while retaining a human-provided problem and executable evaluator [214, 190]. Operational scale therefore does not by itself establish responsibility for the discovery loop.

3.2. Search-Space Provision

The search space S_t defines *where to explore and what the candidates can be*. It contains the editable objects, their representation, admissible primitives, constraints, and operators that transform one candidate into another. It is distinct from the realized set $\mathcal{C}_t$: sampling or generating candidates from a fixed human-provided space does not amount to constructing that space. L2 control requires AI to substantively construct or revise at least one defining component of S_t, such as the representation, primitive library, legal operators, or validity constraints.

Neural architecture search illustrates this distinction. Reinforcement learning, regularized evolution, and differentiable relaxation search architectures effectively, but their results remain conditioned on human-designed operator spaces and validation objectives [336, 210, 145]. Population-based training adapts schedules within a chosen setup [106]; learned optimization and meta-learning broaden the editable learning process while retaining human-selected task distributions and meta-objectives [10, 72].

Later work increases the expressive depth of S_t. AutoML-Zero composes learning algorithms from primitive mathematical operations [211], while evolved reinforcement-learning losses treat computational graphs as candidate learning rules [42]. AlphaTensor and AlphaDev search algorithmic and assembly-level representations [67, 170]. Language models make instructions, critiques, and programs editable: APE, ProTeGi, OPRO, and Promptbreeder search or evolve prompts [331, 204, 293, 69]; Eureka generates reward programs, which belong to E_t rather than S_t [168]; Automated Design of Agentic Systems searches agent scaffolds [100]; and STOP and the Darwin Gödel Machine modify improver or agent code over repeated cycles [310, 312]. These examples are not automatically L2: a method receives L2 only when the demonstrated configuration makes the search space substantively AI-provided while E_t remains human- or externally provided. If E_t is AI-controlled, the precedence rule assigns L3 and the AI-provided space remains in the full profile. In the cited configurations, prompt- and agent-code systems remain L1 direction-selection systems when E_t is fixed; a variant becomes L2 if AI also changes the representation, primitive library, legal operators, or validity constraints that define S_t. Reward-program systems such as Eureka are classified by their control of E_t and may be L3.

3.3. Direction Selection within a Search Space

The search direction D_t defines *how to explore* within S_t. It may specify a capability gap to address, failure to diagnose, bottleneck to remove, research question to pose, or hypothesis to test. Direction is not merely a textual idea: it must causally change which interventions are generated, prioritized, or tested within the available space. A direction is AI-provided only when the configuration gives AI substantive responsibility for selecting or revising that direction in response to the target and its accumulated evidence.

Literature-grounded ideation systems begin to operationalize this responsibility by proposing and revising research questions [14]. Experimentation benchmarks test whether agents can convert observations into subsequent plans [103]. The AI Scientist and AI Scientist-v2 combine ideation with implementation and reporting, whereas Agent Laboratory organizes literature review, experimentation, and reporting from a human-provided idea [158, 292, 217]. A generated proposal counts as AI-provided direction only when it is more than a restatement of the human prompt and when changing that proposal would materially change the ensuing search.

The dependence $D_t = P_D(x_t, h_t, S_t)$ is important. A direction is meaningful only relative to the interventions that the current space permits. A human may specify the direction while AI constructs a new search space for pursuing it, producing an L2 profile such as $\langle \mathrm{H}, \mathrm{A}, \mathrm{H} \rangle$. Conversely, AI may identify a direction while people retain the editable space and evaluator, producing the L1 profile $\langle \mathrm{A}, \mathrm{H}, \mathrm{H} \rangle$.

3.4. Evaluation as the Feedback Signal

The evaluation signal E_t defines *what counts as progress*. It may include metrics, tests, verifiers, reward models, critics, statistical protocols, cost models, aggregation rules, and stopping conditions. A fixed human-authored rubric or benchmark remains externally fixed even when a model executes it; producing a score under that fixed criterion is not by itself AI control of E_t. L3 requires AI to generate or substantively revise an

operative component of $(\mathcal{T}_t, J_t, A_t, \tau_t)$—for example, a test distribution, judgment mapping, aggregation rule, or stopping condition—and that component must govern candidate selection, training, resource allocation, or acceptance in the discovery loop.

Human-preference learning converts judgments into trainable rewards [41], while process supervision assigns credit to intermediate reasoning steps [137]. Constitutional AI and RLAIF transfer parts of critique and preference production to models while retaining human-authored principles, prompts, or targets [15, 126]. LLM-as-a-judge systems make open-ended evaluation scalable but introduce position and verbosity biases [322], as well as self-preference bias when a model favors its own outputs or identity [265]; self-rewarding models further couple improvement and reward production [303].

Evaluation control is identified by provenance; independence is a quality check on the resulting signal rather than a prerequisite for the L3 label. Self-feedback methods such as Self-Refine and Reflexion can provide useful feedback while leaving the criterion externally specified in the cited configurations [169, 223]. L3 requires AI to construct or substantively revise the operative criterion, but it does not require AI-provided direction or search space. Such signals should be tested under optimization pressure because maximizing an imperfect proxy can eventually reduce performance under an independent reference metric [80].

3.5. Responsibility Profiles and Control Strength

For each control variable $i \in \{D, S, E\}$, let $a_i \in \{\mathrm{H}, \mathrm{A}\}$ denote whether the demonstrated configuration treats the operative decision as human- or AI-provided. The binary responsibility profile is

$$\mathbf{a} = \langle a_D, a_S, a_E \rangle. \tag{10}$$

Here H denotes human-authored or externally fixed, non-AI-originated control, while A denotes AI-originated substantive control. A fixed model that only executes a human-authored rubric or benchmark remains an external implementation; a learned or generated mapping is AI-provided only when it has operative authority and passes the counterfactual test. A shared or partial component is recorded separately in the provenance audit and control-strength vector; it receives $a_i = \mathrm{A}$ when the AI component originates or revises the operative decision and changes the downstream loop, even if a human anchor remains. The profile records information that a single level label suppresses. A program-search system with a human problem, AI-provided program space, and human executable evaluator has profile $\langle \mathrm{H}, \mathrm{A}, \mathrm{H} \rangle$ and is L2. A system whose AI constructs the operative evaluation signal for human-selected candidates may have profile $\langle \mathrm{H}, \mathrm{H}, \mathrm{A} \rangle$ and is L3. A shared decision should not be rounded up to full AI control without evidence; its mixed provenance must be recorded in the accompanying audit and, where useful, in the control-strength vector. Neither representation should be read as end-to-end AI control.

Binary labels conceal shared control. We therefore also recommend a descriptive control-strength vector

$$\mathbf{r} = \langle r_D, r_S, r_E \rangle, \qquad r_i \in [0, 1], \tag{11}$$

where r_i summarizes the degree of AI responsibility. This vector is a reporting aid rather than a validated scalar metric or a threshold for assigning L0–L3. Evidence for a high value should cover four properties:

1. **Origination:** AI produces substantive content not explicitly supplied by a human specification;
2. **Task specificity:** the decision adapts to evidence from the present target rather than replaying a generic template;
3. **Causal efficacy:** changing the AI-produced decision changes candidate generation, selection, or the resulting artifact; and

4. **Revisability:** the system can update the decision when feedback reveals that it is ineffective or invalid.

These conditions distinguish substantive control from transcription, execution, or superficial linguistic variation. They also prevent a system from receiving credit for a nominally AI-generated component that the actual search trajectory ignores.

3.6. The Control-Source Ladder, Assignment, and Reporting

The three variables connect through the operational source of control. The resulting labels identify the focal AI-provided variable; they do not assign provenance to every other component. Because a configuration can combine human and AI sources across the three variables, the label is assigned by a fixed precedence rule rather than by an assumption that later labels are more autonomous.

L0. Human-authored or externally fixed discovery loop. Humans or external fixed procedures provide the search space, select the direction within it, and define the evaluation signal. Computation may execute training or measurement, but none of the three control variables is originated or substantively revised by AI. L0 is the human-controlled reference profile.

L1. AI-provided search direction. AI determines how to explore, while people or external fixed procedures still provide the search space and evaluation signal. Typical evidence includes target-specific bottleneck diagnosis, hypothesis generation, or direction prioritization that causally changes the ensuing research plan. This label applies only when S and E remain human-authored or externally fixed.

L2. AI-provided search space. AI determines where exploration occurs and which candidate interventions become available, while evaluation remains human-authored or externally fixed. The direction may be human- or AI-provided, so both $\langle \mathrm{H}, \mathrm{A}, \mathrm{H} \rangle$ and $\langle \mathrm{A}, \mathrm{A}, \mathrm{H} \rangle$ are L2.

L3. AI-provided evaluation signal. AI constructs or substantively revises what counts as progress. Direction and space may be human- or AI-provided; equivalently, L3 includes every profile with $a_E = \mathrm{A}$. Only $\langle \mathrm{A}, \mathrm{A}, \mathrm{A} \rangle$ places all three control variables under AI control.

A label identifies a different source of operational control, not a greater degree of intelligence, usefulness, reliability, or alignment. We assign one label using the precedence $E > S > D$: if $a_E = \mathrm{A}$, assign L3; otherwise, if $a_S = \mathrm{A}$, assign L2; otherwise, if $a_D = \mathrm{A}$, assign L1; assign L0 only when all three coordinates are human-authored or externally fixed. The full profile must accompany the label whenever the remaining allocation of control matters.

Level assignment and reporting. We use the following protocol when coding a paper or system:

1. **Fix the unit of analysis.** Record the exact model and software versions, prompts, tools, orchestration, dataset, budget, evaluator version, random seeds, and human-intervention policy.
2. **Trace provenance.** For D, S, and E, identify the source of the operative decision in the reported configuration. Separate human-authored constraints from AI-originated choices within those constraints, and distinguish a human seed from an AI revision that changes the decision used by the loop. If an AI component changes the operative accept/reject decision while a human anchor remains, code the coordinate as AI-provided and record the anchor separately.

3. **Apply the counterfactual test.** Ask whether removing or changing the AI-produced component would materially alter the downstream search trajectory. If not, the component does not establish control.
4. **Test adaptation and revision.** Record whether the system changes the decision in response to target-specific evidence and whether it can recover when that decision proves misleading. The absence of adaptation does not erase one-shot substantive control.
5. **Assign the full profile first.** Code $\mathbf{a} = \langle a_D, a_S, a_E \rangle$. If $a_E = \mathrm{A}$, assign L3; otherwise, if $a_S = \mathrm{A}$, assign L2; otherwise, if $a_D = \mathrm{A}$, assign L1; assign L0 only when all coordinates are human-authored or externally fixed.
6. **Report residual human control.** List human seeds, templates, primitive libraries, task distributions, rubrics, hidden tests, approval gates, rescue interventions, and stopping rules.

Candidate generation shows that the system acts within a space; by itself it establishes neither AI-provided direction nor AI-provided space. The system is L2 only when AI substantively constructs or revises the representation, primitives, operators, or constraints that define which candidates are expressible, even if a person chose the direction. A generated score establishes L3 only when AI constructs or revises the operative criterion that governs selection, rather than merely executing a fixed prompt or benchmark. Self-modification likewise does not establish L2 by itself: it qualifies only when the modification changes the space-defining objects or operators. Recursive self-improvement additionally requires evidence that the modification improves the process producing later improvements.

Every AI4AI result should report both an outcome and a responsibility audit. The outcome report includes target capability, transfer, resource use, variance, failed trials, and comparisons with human-designed and automated baselines. The responsibility audit records $\mathbf{a}$, optional control strengths $\mathbf{r}$, human inputs, evaluator visibility, candidate/evaluator model overlap, contamination controls, intervention frequency, and the number of completed discovery generations. When provenance changes across cycles, report the profile for each cycle and state the reporting window rather than collapsing the trajectory into one unqualified label.

Evidence should match the claimed control variable. Direction quality should be assessed with independent judgments of importance, novelty, feasibility, actionability, and whether the proposed trajectory changes the next interventions. Search-space quality requires validity, coverage, diversity, expressivity, and matched-compute comparisons. Evaluation-signal quality requires calibration, adversarial robustness, resistance to reward hacking, and agreement with hidden external criteria. Claims of recursive benefit require cross-generation evidence under a frozen or held-out evaluator: an improvement at cycle t must increase the productivity or capability of the improver at $t+1$, not merely the score of the current target.

4. L1: AI Provides the Search Direction

The control transferred at L1. L1 transfers control over *how to explore*. Its defining capability is substantive AI control of the search direction: this may be expressed in a single target-specific intervention or through history-conditioned direction selection. In the latter case, given a human-specified search space and evaluation rule, the AI reads previous candidates, scores, critiques, failures, and partial discoveries, then decides which improvement trajectory should be pursued next. Section 4.1 formalizes this view. Section 4.2 studies evolutionary program improvement, including FunSearch and OpenEvolve. Section 4.3 covers prompt and workflow optimization. Section 4.4 connects direction discovery to reflection, memory, and explicit search over reasoning trajectories. Section 4.5 examines experiment-driven research agents, and Section 4.6 summarizes evaluation criteria, failure modes, and the boundary with L2.

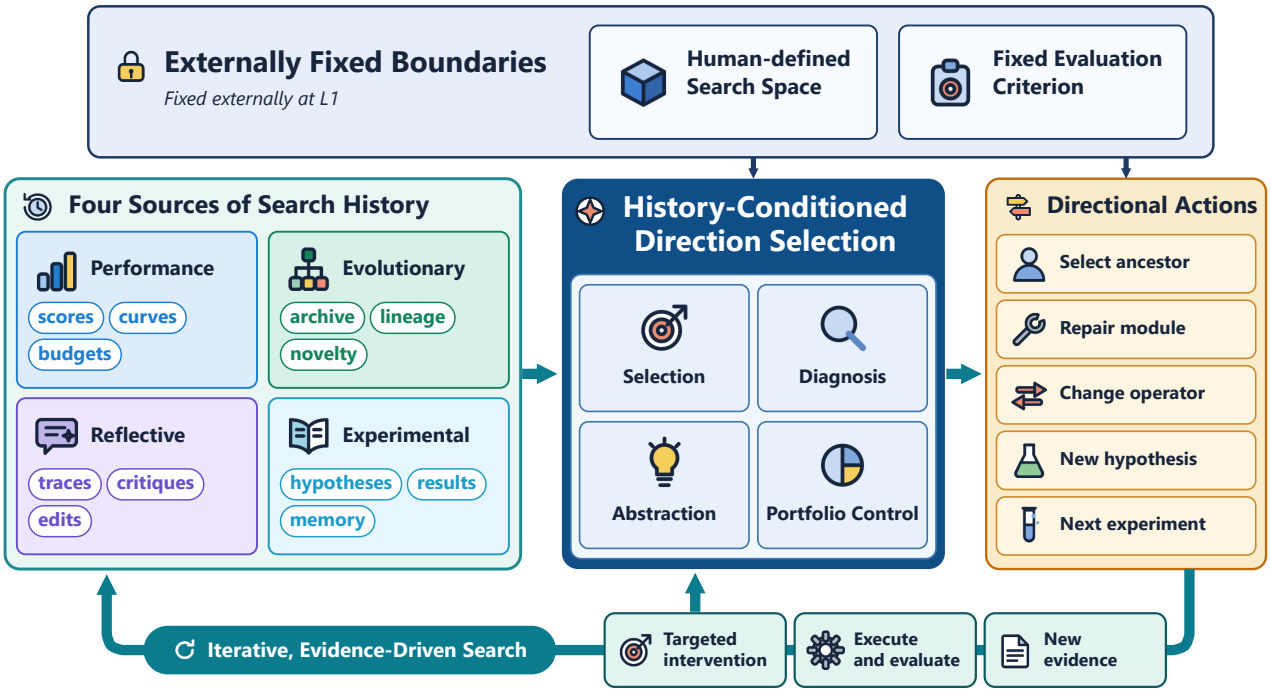


Figure 3: **History-conditioned direction selection at L1.** The search space S and the operative evaluation criterion E are fixed externally, so the profile is $\langle \mathrm{A}, \mathrm{H}, \mathrm{H} \rangle$ and AI controls only the direction of the search. The AI reads four carriers of accumulated history—performance (candidate scores, learning curves, budget), evolutionary (population archive, ancestry, variation), reflective (execution traces, critiques, suggested edits), and experimental (hypotheses, results, persistent memory)—and its direction policy P_D applies selection, diagnosis, abstraction, and portfolio control to that record. The resulting directional act selects an ancestor, repairs a module, changes an operator, proposes a new hypothesis, or allocates the next experiment. After execution and evaluation, the new candidates and the evidence they carry return along the rail at the bottom and close the loop. L1 is thus defined by *what to try next* within a fixed representation and objective; L2 begins when AI instead changes *what can be tried*.

4.1. History-Conditioned Direction Selection

History as search state. Let the research history at cycle t contain attempted interventions, observed evidence, and the relations among them:

$$h_t = \{(c_i, q_i, \delta_i, \rho_i)\}_{i<t}, \qquad D_t = P_D(x_t, h_t, S_t).$$

Here c_i is a previous candidate, q_i its measured outcome, δ_i an optional textual or structured diagnosis, and ρ_i its lineage, neighborhood, or experimental context. The direction policy P_D may select a bottleneck, an ancestor to elaborate, a region to abandon, an operator to emphasize, or a hypothesis that explains the observed pattern. At canonical L1, the search space S_t and the operative evaluation criterion E_t remain externally fixed; AI controls which intervention or resource allocation is pursued *within* that space. A fixed pruning or acceptance rule is part of E_t, not an additional L1 decision.

Performance histories: scalar allocation. This definition connects recent LLM systems to a longer history of adaptive search. Random search provides a history-agnostic baseline, whereas Bayesian optimization,

model-based AutoML, and population methods use accumulated observations to decide which configuration or training regime deserves the next unit of compute [19, 230, 71, 106]. Hyperband, BOHB, Optuna, and Freeze-Thaw Bayesian optimization provide useful allocation baselines: their fixed pruning or promotion criteria remain part of the external evaluation protocol, while the history-conditioned policy determines which configuration or budget receives the next unit of attention [134, 65, 8, 238]. If an AI system learns or revises the pruning, acceptance, or stopping criterion, that component is evaluation control and is assigned L3 by the precedence rule rather than L1.

Performance histories: response surfaces. Neural architecture search similarly learns or evolves proposal policies from previously evaluated architectures [336, 210, 145, 62]. Progressive NAS and Neural Architecture Optimization use evaluated architectures to predict productive expansions or latent-space moves; ENAS and Network Morphism preserve information through shared parameters or function-preserving transformations [140, 160, 201, 267]. Predictor-based methods such as BANANAS and Neural Predictor fit explicit response surfaces over architecture histories, while early performance prediction, zero-cost proxies, and Once-for-All reduce the evidence cost of deciding which direction deserves full training [271, 268, 16, 173, 23]. Reproducible suites including NAS-Bench-101, NAS-Bench-201, NATS-Bench, and surrogate NAS benchmarks make it possible to replay identical histories and compare direction policies rather than incomparable final models [298, 54, 53, 308].

Evolutionary histories: archives and variation. Population-based systems retain more than scores: they preserve ancestry, island membership, mutation operators, novelty, and rediscovery. FunSearch, AlphaEvolve, OpenEvolve, and ShinkaEvolve use archive state to decide which ancestor or island should generate the next program, while Language Model Crossover, LLMatic, SBLLM, PromptBreeder, and CodeEvolve use successful lineages to choose a variation strategy [214, 190, 219, 123, 177, 185, 81, 69, 12]. Here the history is relational: direction depends on where a candidate sits in an evolving population, not only on its scalar value.

Reflective histories: critiques and trajectories. Language makes failed attempts directly interpretable as search evidence. ProTeGi, OPRO, TextGrad, and GEPA translate errors, scores, or execution traces into targeted edits to AI artifacts [204, 293, 306, 6]. These methods treat a critique as a directional signal rather than as a final evaluation alone.

Experimental histories: replay and research memory. The fourth family grounds direction selection in reproducible experimental records. NAS-Bench-101, NAS-Bench-201, NATS-Bench, and surrogate benchmarks permit counterfactual replay of the same evaluated history, making competing direction policies directly comparable [298, 54, 53, 308]. Research agents such as AI Scientist, MLE-STAR, and AutoResearchClaw extend the same principle to literature, hypotheses, code, failed runs, and reviewer feedback, using accumulated evidence to select the next experiment [158, 184, 146]. Experimental history therefore supports both discovery and audit: it records why a direction was chosen and whether the choice survives replication.

From observation to a directional act. Across the four families, history influences direction through *selection*, *diagnosis*, *abstraction*, and *portfolio control*. Performance histories primarily rank or predict; evolutionary histories preserve ancestry and strategy; reflective histories explain what should change; and

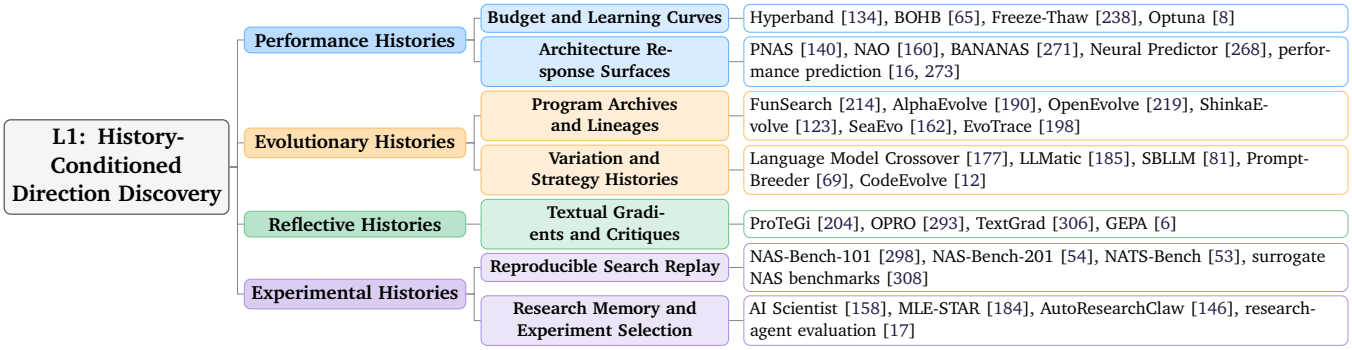


Figure 4: **Taxonomy of L1 search-direction discovery from accumulated history.** The literature separates into four history carriers. Performance histories support budget allocation and response-surface prediction; evolutionary histories preserve archives, lineages, and variation strategies; reflective histories convert critiques and trajectories into targeted revisions; and experimental histories use replayable records or research memory to select the next experiment.

experimental histories determine which uncertainty should be resolved next. Modern systems often combine several carriers. The history is therefore not merely a log, but the state from which the system constructs its next research move.

Figure 4 organizes the field into performance, evolutionary, reflective, and experimental histories according to the evidence that the AI actually reads. This organization is intentionally causal rather than application-based: one system may combine scores, lineages, critiques, and experimental memory, but each carrier supports a different kind of directional act. Table 3 then decomposes representative methods into their direction policy, the history evidence they read, and the next directional act they control.

4.2. Evolutionary Program Improvement

Archive-conditioned program evolution. Evolutionary coding agents are the clearest current realization of history-conditioned direction discovery. FunSearch stores high-performing programs in an island-structured archive, samples exemplars from that archive, and asks an LLM to continue the observed line of improvement under a human-provided program skeleton and executable evaluator [214]. AlphaEvolve scales the same basic pattern to general codebases and multiple evaluators [190]; OpenEvolve exposes an open implementation with program databases, diversity-aware sampling, and iterative LLM mutation [219]. In all three, the decisive L1 operation is the use of accumulated program lineages and scores to choose how the next proposal should depart from its ancestors.

Language models as history-sensitive variation operators. Several adjacent systems make this direction policy more explicit. Algorithm Evolution and Evolution of Heuristics evolve natural-language algorithmic ideas together with executable implementations [143, 142]. LLMs can also serve as evolutionary optimizers, surrogate models, or learned variation operators that infer productive moves from the current population [150, 91]. Evolution through Large Models and Language Model Crossover showed that pretrained models can produce semantic mutations and recombinations from parent examples rather than blind syntactic perturbations [127, 177]. LLMatic couples this capacity to a quality-diversity archive for neural architecture search, and SBLLM retrieves successful optimization patterns before composing the next code edit [185, 81]. These systems turn population history into an implicit local model of "what kind of change works here."

Table 3: Representative L1 methods as history-conditioned direction policies, grouped by the history carrier of Figure 4. Each method reads h_t and emits a directional or resource-allocation decision D_t.

Method	Direction rule	Direction emitted D_t	Evidence read h_t
Performance histories			
Hyperband [134]	$D_t = \text{PromoteNext}(h_t)$	configuration and next budget (continue or resume)	learning curves, validation scores, rung budgets
BOHB [65]	$D_t = \arg\max_{c,b}\ \alpha(c, b \mid h_t)$	configuration and next budget	scores, budgets, model-based density estimates
BANANAS [271]	$D_t = \arg\max_{a \in S}\ \alpha(\widehat{q}_t(a))$	architecture neighborhood to expand	evaluated architectures, encodings, predictor uncertainty
Evolutionary histories			
FunSearch [214]	$D_t = (p, \mu) \sim P_D(\cdot \mid \mathcal{A}_t)$	parent program and semantic variation	island archive, program scores, ancestry
OpenEvolve [219]	$D_t = (p, o, \ell) \sim P_D(\cdot \mid \mathcal{D}_t)$	parent, edit operator, edit location	program database, lineage, diversity, evaluator feedback
PromptBreeder [69]	$D_t = (p, m) \sim P_D(\cdot \mid \mathcal{P}_t)$	parent prompt and mutation strategy	task prompts, mutation prompts, fitness history
Reflective histories			
TextGrad [306]	$D_t = \text{Edit}(-\nabla_{\text{text}} L(\pi_t))$	editable component and targeted revision	execution trace, loss, textual gradient
GEPA [6]	$D_t = \text{SelectEdit}(\mathcal{A}_t, \Delta_t)$	parent and reflection-conditioned edit	reflective feedback, lineage, Pareto archive
Experimental histories			
AI Scientist [158]	$D_t = \arg\max_e\ U(e \mid \mathcal{M}_t)$	next hypothesis and experiment	literature, hypotheses, code, results, reviews
MLE-STAR [184]	$D_t = \text{Revise}(\pi_t, \mathcal{M}_t)$	pipeline component and revision strategy	pipeline trials, ablations, validation, memory
AutoResearchClaw [146]	$D_t = P_D(\mathcal{L}_t, \mathcal{X}_t, \mathcal{R}_t)$	research action and artifact to revise	literature memory, experiment log, reviewer feedback

Evolving the proposal policy. EvoPrompt applies evolutionary selection to prompts [90], while PromptBreeder evolves both task prompts and the mutation prompts that generate them [69]. ShinkaEvolve adds novelty rejection, bandit-based model selection, and parent sampling to improve sample efficiency [123]. This is a stronger L1 pattern than merely sampling more candidates: past outcomes alter which parent, mutation model, and novelty threshold govern the next move. It remains L1 so long as the program representation, admissible edits, and executable objective are supplied externally.

Strategy-level archives. The same loop now appears across algorithm and system design. CodeEvolve combines island-based evolution with inspiration-driven crossover [12]; LLM-FE evolves feature-engineering programs [2]; program-search benchmarks test whether search histories support reusable abstractions across games and ARC-like tasks [57, 226]. Evolution of Kernels, SeaEvo, graph-augmented evolution, and adaptive population handoff further study how the archive should route proposals among local strategies [89, 162, 37, 161]. Work isolating the contribution of evolutionary selection shows that the population mechanism, not only the base model, materially changes performance [315]. EvoTrace goes further by replaying evolutionary histories and separating structural innovations from constant tuning, rediscovery, and

cyclic edits [198].

The L1–L2 classification test. Under our unit-of-analysis rule, FunSearch and OpenEvolve belong to L1 when the mutable program boundary, programming language, evaluator, and admissibility constraints are supplied by humans and AI uses the archive to choose the next improvement direction. The same software can exhibit an L2 component if it invents new representations, primitives, or edit operators that redefine which candidates are expressible. This distinction classifies the demonstrated control decision, not the project name.

4.3. Prompt, Pipeline, and Workflow Optimization

Textual gradients over prompt histories. Prompt optimization provides a second large family of L1 systems because the optimization trace can be represented directly in language. Automatic Prompt Engineer samples and ranks candidate instructions [331]; ProTeGi turns errors into textual gradients and edits prompts by beam search [204]; OPRO places previous solutions and scores in the optimizer model's context so that it proposes a better next instruction [293]. PromptAgent formulates prompt revision as strategic planning [258], and TextGrad propagates language feedback through compound pipelines [306]. These methods differ in their search operator, but each converts a record of prior failures and successes into a directional signal.

Editable units from tokens to compound programs. The field has progressively expanded the editable unit. AutoPrompt, BBT, BBTv2, GrIPS, RLPrompt, and TEMPERA optimize discrete tokens, continuous surrogates, edit sequences, or test-time prompt policies [222, 236, 235, 202, 48, 316]. SAMMO searches structured prompt programs [218]; DSPy compiles declarative language-model modules into optimized pipelines [113]; MIPRO jointly selects instructions and demonstrations for multi-stage programs [192]. PromptWizard, CriSPO, and automatic instruction-exemplar selection use task analyses or multi-aspect critiques to decide which component should change next [4, 92, 251]. Local prompt optimization studies whether local edits can outperform global rewrites under constrained budgets [107].

Reflective evolution. GEPA makes the L1 mechanism unusually visible: it samples execution trajectories, reflects on natural-language feedback, proposes targeted prompt updates, and maintains a Pareto frontier of candidates [6]. The value of such reflection is not guaranteed. Re-evaluations of OPRO show sensitivity to the optimizer model, scorer, handcrafted meta-instruction, and token budget [317]. These results motivate a distinction between *history exposure* and *history use*: placing scores in a context window does not establish that the next edit is causally responsive to them.

Credit assignment across workflows. History-conditioned workflow search is especially important for AI4AI because prompts, evaluators, data filters, and agent roles interact. A weak pipeline may not need a globally better instruction; it may need a new decomposition, a different handoff, or a targeted repair at the module responsible for a recurring failure. DSPy-style compilation, textual gradients, and critique-suggestion loops make such credit assignment partially explicit [113, 306, 92]. They therefore move L1 from selecting among isolated prompt strings toward selecting directions in a structured development process.

Fixed workflow or AI-provided space? The L1 interpretation depends on what remains fixed. If humans define the modules and AI only decides how to revise them using observed performance, the focal control is direction. If AI invents the module graph, the representation language, or an unbounded family of tools, it begins to provide the search space and must also be reported as L2. The same optimization trace should therefore be accompanied by its full responsibility profile rather than by the method label alone.

4.4. Reflection, Memory, and Search over Trajectories

Diagnostic histories. Scalar scores are often too weak to identify a productive direction. In-scope reflective systems use execution traces, evaluator feedback, or artifact-level critiques to decide which AI artifact should be revised next. Textual gradients, structured critiques, and multi-stage code-generation flows provide such evidence when they change a prompt, program, training procedure, or agent scaffold; standalone answer-refinement and reasoning protocols are outside the AI4AI scope.

Branching histories. Explicit search methods retain several possible directions. In AI4AI settings, the retained branches must correspond to alternative artifact edits, experiments, or research hypotheses rather than merely alternative answers to one task. STaR converts successful reasoning traces into training data for subsequent improvement [311]; this parameter update is why it belongs to AI4AI Learning. Choosing an AI4AI direction may therefore require maintaining a topology of artifact alternatives rather than compressing history into a single incumbent.

Persistent histories. Long-term memory extends the horizon over which direction can change. MemoryBank and MemGPT distinguish active context from persistent experience [327, 195]; ExpeL extracts transferable lessons across task trajectories [320]; Voyager accumulates an executable skill library and uses environment feedback to select increasingly ambitious goals [253]. Generative Agents similarly retrieve, reflect on, and synthesize past events into higher-level plans [196]. These systems are not automatically AI4AI, but their memory mechanisms become L1 components when the remembered experience guides the improvement of an AI artifact or its development process.

Self-correction is an empirical claim. Evidence on intrinsic self-correction is mixed. Controlled studies report that language models often fail to repair reasoning without reliable external feedback, while other work finds gains under carefully constructed intrinsic critique protocols [102, 141]. ReST^{EM} shows a different route: successful and unsuccessful traces can be converted into iterative self-training data rather than used only at inference time [225]. For L1, the central question is therefore not whether a model can produce a critique, but whether the critique selects a better intervention than score-only, shuffled-history, or no-history controls.

Counterfactual dependence on history. The central limitation is that reflection is evidence of control only when it changes the search trajectory. A fluent critique that restates the failure message, a memory item that is never retrieved, or a tree whose branches collapse to the same proposal does not establish AI-provided direction. Causal ablations should remove, shuffle, or replace the historical record and measure whether the resulting direction and final discoveries change.

4.5. Experiment-Driven Research Direction

Research history as a heterogeneous record. At the scale of research workflows, history consists of literature, experimental plans, implementation traces, failed runs, reviewer feedback, and intermediate results. ResearchAgent uses literature-grounded retrieval to propose and refine research ideas [14]. MLAgentBench evaluates whether language agents can plan and execute machine-learning experiments [103], while MLE-bench tests end-to-end machine-learning engineering under realistic competition tasks [27]. The AI Scientist and AI Scientist-v2 use prior experimental evidence to revise hypotheses, code, and reports over repeated cycles [158, 292]. Agent Laboratory distributes literature review, experimentation, and writing among collaborating agents [217].

Search and targeted experimental refinement. MLE-STAR first retrieves plausible model families and then uses block-level ablations to decide which part of a machine-learning solution should be refined next [184]. AutoResearchClaw similarly emphasizes persistent research memory and self-reinforcing cycles, while Jr. AI Scientist begins from a baseline paper and conducts autonomous extensions [146, 181]. These systems are closer to L1 than agents that merely execute a static plan because experimental results can redirect attention among hypotheses, implementation components, and follow-up tests.

Multi-agent evidence aggregation. General agent frameworks supply complementary mechanisms. Data Interpreter records data-science plans and execution feedback [96]; MetaGPT, AutoGen, and AgentVerse coordinate specialized roles through shared messages and artifacts [97, 279, 35]. AgentCoder iterates code generation against testing agents [101], while SWE-agent and Agentless expose how repository feedback can be converted into repair strategies with very different amounts of scaffolding [294, 282]. OSWorld broadens the executable environment in which such history-conditioned decisions can be evaluated [288]. Automated Design of Agentic Systems treats the agent organization itself as an editable artifact [100].

AgentBench, AgentBoard, GAIA, and SWE-bench add complementary views of whether multi-turn histories improve action selection in interactive, assistant, and software-engineering settings [151, 167, 178, 109]. More Agents Is All You Need reports gains from sampling and aggregating multiple agents, whereas AgentScope provides engineering support for maintaining their messages and states [131, 78]. For L1, however, parallelism counts only when disagreement or complementary evidence changes the next research direction; a larger ensemble that repeats the same trajectory is additional compute, not additional control.

Degrees of directional autonomy. These systems reveal a spectrum of directional autonomy. At the weakest end, AI executes a human-authored experiment list. Stronger systems diagnose why a run failed, compare competing explanations, and allocate the next experiment to resolve uncertainty. Stronger still, they may revise the research question after evidence invalidates its assumptions. Such revision is one strong example, but L1 more generally requires a target-specific change in *how the investigation proceeds*, rather than simply completing the next queued operation.

Research-agent audits. Recent audits expose why final-paper appearance is an inadequate proxy for L1 competence. Evaluations of the AI Scientist report mixed reproducibility and uneven experimental validity [17]; implementation-focused analyses argue that weak execution and verification remain binding bottlenecks [333]. Other studies find that automation can hide consequential intermediate decisions, that independent AI scientists may disagree, and that headline scientific impact remains far from established

[163, 74, 285]. MLReplicate measures whether autonomous systems can reproduce machine-learning results, while SciIntegrity-Bench tests citation, attribution, and experimental-integrity failures [76, 296]. Package hallucinations provide a concrete example of a seemingly productive direction failing at the implementation layer [231].

Discoveries that update the improver. Program-search systems also become research agents when discoveries feed back into the next cycle. AlphaTensor and AlphaDev used learned search to reveal new algorithmic strategies [67, 170]; Eureka evolves reward programs from environment outcomes and is therefore an L3 evaluator-generation case rather than an L1 row here [168]; STOP and the Darwin Gödel Machine use previous improvements to modify an improver or coding agent within a fixed edit boundary [310, 312]. AutoRISE applies related strategy evolution to adversarial testing [82]. Such systems are particularly informative because their histories contain both object-level results and evidence about the process that produced those results.

4.6. Direction Quality, Failure Modes, and the Boundary with L2

Degrees of history use. History conditioning is not a binary property of an entire system. *History-exposed* systems receive earlier candidates or scores but need not respond to them. *History-responsive* systems produce detectably different proposals when the record changes. *History-explanatory* systems identify which observation supports the chosen move. *History-counterfactual* systems can state what would have been tried under a different outcome. The stronger degrees support more credible claims that AI, rather than a fixed script, supplies the direction. Reproducible NAS studies show why this distinction matters: sophisticated adaptive policies can lose to random search when budgets, spaces, and evaluation protocols are controlled [133, 270, 273].

Evaluating the direction rather than only the destination. L1 should be evaluated at the level of the direction, not only the final artifact. A useful evaluation stack asks whether the chosen direction is novel relative to the supplied history, causally responsive to evidence, feasible under the remaining budget, diverse across independent runs, and more productive than random, greedy, or history-ablated baselines. For population methods, reporting should include archive coverage, lineage depth, repeated-edit rates, and the contribution of crossover or diversity maintenance [198, 315]. For prompt and reflection methods, it should include edit locality, critique-action agreement, retrieval usage, and robustness to misleading history [306, 92, 320].

Characteristic failure modes. Four failure modes recur. *Historical anchoring* keeps the search close to early incumbents; *archive collapse* removes competing hypotheses; *credit misassignment* attributes a gain to the wrong edit or module; and *feedback overfitting* selects directions that exploit a fixed evaluator. Evolutionary archives and multi-agent portfolios mitigate some of these risks but can also multiply compute without producing genuinely distinct research trajectories [214, 12, 279]. Direction discovery therefore needs independent replay, counterfactual history ablations, and held-out evaluators.

Boundary with L2. The L1–L2 boundary is determined by the origin of the candidate domain. L1 may select ancestors, mutation strategies, modules to repair, or hypotheses to test inside a human-fixed representation. L2 begins when AI substantively changes the editable objects, primitives, constraints, or operators that

define what can be proposed. FunSearch and OpenEvolve are canonical L1 examples in this chapter because their central mechanism uses previous programs and evaluator feedback to choose the next improvement direction under an externally supplied coding boundary. If a deployment allows the same agent to invent that boundary, its full profile must record the additional L2 control.

5. L2: AI Provides the Search Space

At L2, the focal object is the search space itself: where improvement can occur and what objects or moves are available there. A system reaches L2 only when AI substantively determines the search space by selecting, constructing, expanding, compressing, or revising its representation, primitive library, legal composition rules, or validity constraints. Direction may come from either source, while the evaluation signal remains human-authored or externally fixed. Note that AI-generated candidates alone do not establish L2. A system may produce highly novel designs within a human-specified space; L2 requires AI to substantively determine or revise the space from which those candidates can be generated. This process is summarized in Figure 5.

5.1. Formalization and evaluation of AI-defined search spaces

This subsection formalizes the L2 search space, its main components, and its evaluation. We first distinguish the AI-defined active space from the broader ambient universe, then decompose it into its representation, available primitives, legal composition and edit operators, and validity constraints, and finally discuss how to evaluate the space by distinguishing its contribution from that of the navigation policy. Figure 6 organizes L2 work by the search-space artifact supplied by AI.

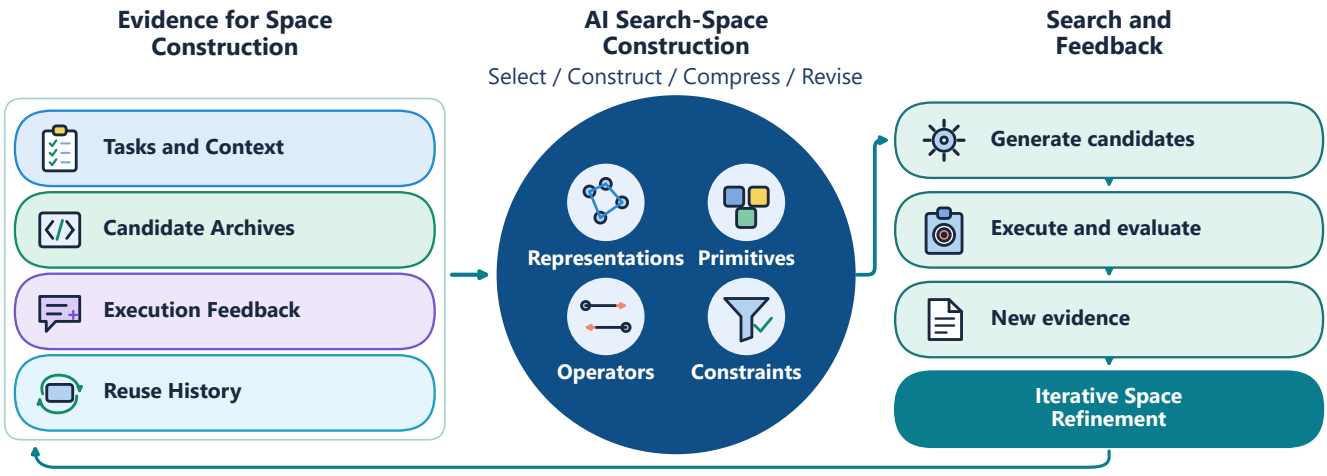


Figure 5: **AI construction of the active search space at L2.** Within a human-provided ambient universe $\mathcal{U}$, AI draws on task context, candidate archives, execution feedback, and reuse history to construct or substantively revise the active search space. Selection, construction, compression, and revision can act on one or more space components: candidate representations $\mathcal{R}$, available primitives $\mathcal{P}$, legal composition and edit operators $\mathcal{O}$, and admissibility constraints $\mathcal{K}$. The resulting space shapes candidate generation, while execution and evaluation produce new evidence that can support iterative space refinement. Search direction may be supplied by humans, predefined rules, or an L1 component; the evaluation criterion E remains externally fixed.

AI-defined active search spaces. At L2, AI defines an active search space within a broader universe. Let $\mathcal{U}$ denote this ambient universe: a programming language, hardware interface, broad architecture grammar, or set of executable APIs. At iteration t, the active space is represented by

$$S_t = (\mathcal{R}_t, \mathcal{P}_t, \mathcal{O}_t, \mathcal{K}_t),$$

where $\mathcal{R}_t$ is the representation, $\mathcal{P}_t$ the available primitives or modules, $\mathcal{O}_t$ the legal composition and edit operators, and $\mathcal{K}_t$ the validity constraints. The candidates induced by this representation belong to the ambient universe, so $\mathcal{C}_t \subseteq \mathcal{U}$. L2 requires AI to construct or substantively revise the search space that shapes candidate generation:

$$S_t \sim P_S(\cdot \mid x_t, h_t), \qquad \mathcal{C}_t = \{c_{t,i}\}_{i=1}^{n_t} \sim G_t(\cdot \mid S_t, D_t; B_t), \qquad q_{t,i} = E_t(c_{t,i}, x_t, S_t, D_t).$$

What matters for L2 is that AI determines S_t, after which G_t is the candidate-generation or navigation policy that samples or constructs candidates inside that space. In the batch abstraction of Section 3, S_t is formed before D_t is selected within it; online systems may instead alternate space construction and direction selection as evidence arrives. The ambient universe $\mathcal{U}$ may remain human-defined, just as a programming language can stay fixed while AI learns a new domain-specific library. This separates search-space construction from navigation: S_t defines the domain over which downstream search operates, whereas G_t determines how that domain is explored under the available budget. For an L2 assignment, E_t in the displayed equation is held externally fixed; if AI constructs or revises its operative criterion, the precedence rule assigns L3.

Search-space evaluation and attribution. Evaluation of an L2 system should focus on the quality and consequences of the AI-defined search space itself. Evidence should identify which components of $(\mathcal{R}, \mathcal{P}, \mathcal{O}, \mathcal{K})$ are constructed or revised by AI and characterize how these changes alter the candidates, compositions, or constraints available to downstream search. Because downstream performance also depends on the navigation policy G_t, evaluation should control for this factor rather than treat it as part of the L2 contribution. Comparisons under a fixed or matched navigation policy can therefore help attribute gains specifically to the AI-defined search space.

We review existing L2 methods by the part of the search space that AI constructs or revises. Figure 6 provides an overview of this component-based organization. while Table 4 decomposes representative methods into their space-construction policies, human-provided ambient universes, and AI-provided active spaces. We organize the discussion around changes to primitives $\mathcal{P}_t$, representations $\mathcal{R}_t$, composition and edit operators $\mathcal{O}_t$, and admissibility constraints $\mathcal{K}_t$. Methods affecting multiple components are discussed under their primary space-defining change, with secondary effects noted where relevant.

Selecting and constructing primitives. Primitive-level construction changes the modules from which downstream candidates are assembled. Neural operations, library functions, and callable tools or skills instantiate $\mathcal{P}_t$ in different settings; the candidate space consists of architectures, programs, or action routines built from these primitives under the remaining components of S_t. Selection alone can define an active primitive set. DDSAS represents a subspace as a binary edge–operation mask over a fixed architectural graph. Upper Confidence Bound (UCB)-guided global top-L selection dynamically constructs the active subspace, and differentiable optimization subsequently searches within and updates this subspace. [295]. The main change is therefore to the available neural operations, not to the graph encoding or the rules for editing architectures.

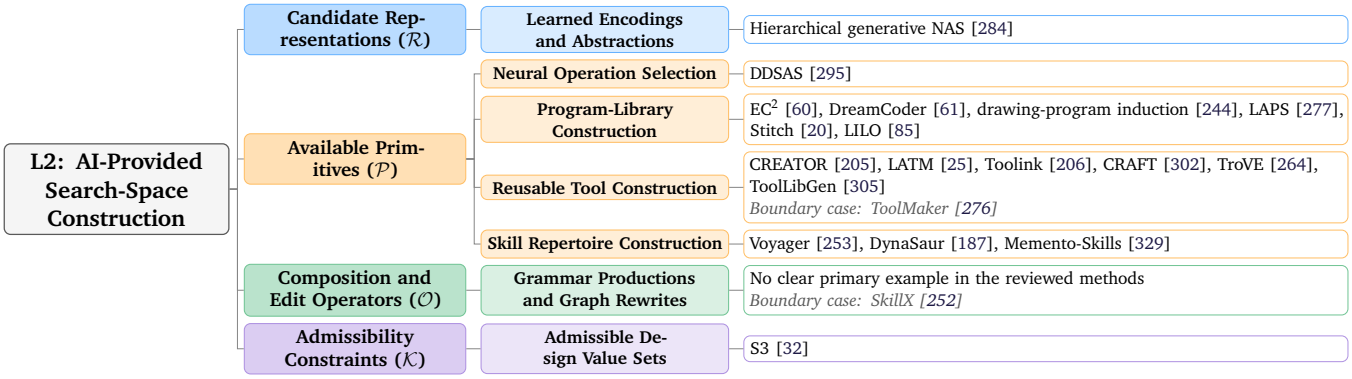


Figure 6: **Component-based taxonomy of L2 AI-provided search spaces.** Methods are classified by the primary search-space component that AI constructs or revises: candidate representations ($\mathcal{R}$), available primitives ($\mathcal{P}$), legal composition and edit operators ($\mathcal{O}$), and admissibility constraints ($\mathcal{K}$). The operator branch is retained as a distinct intervention target without a clear primary example among the reviewed methods. ToolMaker and SkillX are marked as boundary cases rather than unqualified examples of primitive or operator construction. Attribution is component-specific.

Program-library learning constructs new primitives rather than selecting existing ones. EC^2 learns reusable domain-specific subroutines while jointly training a neural recognition model to guide search over programs expressed in the learned domain-specific language(DSL) [60]. DreamCoder interleaves task solving with abstraction learning that extends its DSL with reusable symbolic abstractions, and drawing-program induction applies the same library-learning mechanism to induce reusable, abstract, and compositional drawing subroutines. [61, 244]. LAPS additionally uses natural-language supervision to learn abstractions and search heuristics [277]. Stitch synthesizes abstractions that capture recurring program structure and compress the program corpus, while LILO integrates LLM-guided and enumerative program synthesis, Stitch-based library compression, and an LLM-based auto-documentation (AutoDoc) module that generates human-readable names and docstrings for learned abstractions. [20, 85]. Across these systems, the learned functions become callable constituents of subsequent programs, making the library update the primary $\mathcal{P}_t$ intervention.

Tool-making systems expose the same mechanism through agent interfaces. CREATOR generates a tool before deciding how to apply it, LATM caches generated Python tools for subsequent requests, and Toolink creates a toolkit before planning and tool calling [205, 25, 206]. CRAFT abstracts generated code solutions into reusable tools, forming a specialized toolset for downstream retrieval and use, whereas TROVE incrementally grows and trims a toolbox of reusable high-level functions [302, 264]. ToolLibGen clusters question-specific tools and refactors them into structured libraries with unified interfaces [305]. Here, tools affect the search space because later solutions invoke them as available building blocks, rather than because tool generation itself constitutes a final solution.

Voyager and DynaSaur extend this construction across tasks by accumulating executable skills or actions that later solutions can reuse [253, 187]. Memento-Skills maintains and refines multi-artifact skills comprising declarative specifications, helper scripts, and prompts [329]. Their L2-relevant object is the evolving repertoire of usable routines, not the retrieval or routing policy that selects among them. Such primitives can encapsulate behavior already expressible in the base language; they need not introduce new semantic capabilities to alter the building blocks and lengths of candidate solutions.

ToolMaker illustrates a related but narrower case: it implements reusable tools from scientific code repositories and prepares their execution environments, while the desired task specification, repository, and required

Table 4: Search-space construction in representative L2 methods, grouped by the four space components of Figure 6. Each row separates the AI-constructed active space S_t from the human-specified ambient universe $\mathcal{U}$. h_t is accumulated search evidence, $\mathcal{P}_t$ the reusable primitives ($\mathcal{P}_0$ initial), Π_t solved programs (Π_{train} execution-checked training solutions), $\mathcal{L}_t$ language annotations, and x_t, X_t, τ a task, task examples, and a task family.

Method	Space rule	AI-constructed space S_t	Human universe $\mathcal{U}$
Candidate representations ($\mathcal{R}$) · *Learned encodings and abstractions*			
Hierarchical generative NAS [284]	$\mathcal{R}_t = \text{Cluster}(\text{Enc}_\phi(\text{cells}))$ $S_t = \text{Dec}_\phi(\mathcal{R}_t^L)$	G-VAE cell embeddings; proxy-clustered cell families	architectural template, cell space, zero-cost proxies
Available primitives ($\mathcal{P}$)			
DDSAS [295]	$S_t = \text{Mask}(\text{Top}_L(\text{UCB}(h_t)))$	dynamic binary edge–operation mask	fixed architecture graph, candidate operations, budgets
DreamCoder [61]	$\mathcal{P}_{t+1} = \mathcal{P}_t \cup \text{Abstract}(\Pi_t)$	probabilistic DSL enriched with reusable abstractions	base DSL, primitives, execution semantics
LAPS [277]	$(\mathcal{P}_{t+1}, \theta_{t+1})$ $= \text{Learn}(\mathcal{P}_t, \theta_t, \Pi_t, \mathcal{L}_t)$	language-grounded abstractions	base DSL, program syntax, language annotations
Stitch [20]	$\mathcal{P}_{t+1} = \mathcal{P}_t \cup \text{CTS}(\Pi_t)$	reusable functions that compress common program patterns	base DSL, primitives, valid expressions
LILO [85]	$\mathcal{P}_{t+1} = \mathcal{P}_0 \cup$ $\text{AutoDoc}(\text{Stitch}(\mathcal{P}_t, \Pi_t))$	compressed abstractions with LLM-written names and docstrings	base DSL, composition rules, task descriptions
CREATOR [205]	$\mathcal{P}_t = \text{CreateTools}(x_t; \mathcal{U})$	reusable tools generated before planning their use	Python runtime, APIs, task specifications
LATM [25]	$\mathcal{P}_{t+1} = \mathcal{P}_t \cup \text{Cache}($ $\text{Verify}(\text{MakeTool}(X_t)))$	verified Python tools cached for reuse	Python runtime, tool interface, task examples
Toolink [206]	$\mathcal{P}_t = \text{CreateToolkit}(\tau, X_t; \mathcal{U})$	task-family toolkit for Chain-of-Solving planning	Python runtime, task family, tool interface
CRAFT [302]	$\mathcal{P}_t = \text{Dedup}(\text{Validate}($ $\text{Abstract}(\Pi_{\text{train}})))$	abstracted, validated, deduplicated tool library	task APIs, Python runtime, training tasks, validators
TroVE [264]	$\mathcal{P}_{t+1} = \text{Trim}_t(\mathcal{P}_t \cup$ $\text{Select}(\text{GenByUse}(x_t)))$	execution-consistent reusable functions, pruned by usage	base primitives, programming language, execution interface
Voyager [253]	$\mathcal{V}_t = \{s_t :$ $\text{TaskSuccess}(x_t, s_t; h_t) = 1\}$ $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \mathcal{V}_t$	success-verified, retrievable, composable skills	Minecraft APIs, JavaScript, environment mechanics
Composition and edit operators ($\mathcal{O}$) · *Grammar productions and graph rewrites*			
No clear primary example. Boundary case: SkillX [252] learns skills that guide execution but do not change the operators.			
Admissibility constraints ($\mathcal{K}$) · *Admissible design value sets*			
S3 [32]	$S_t = \prod_{k=1}^{K} \mathcal{D}_{k,t}$ $\mathcal{D}_{k,t+1} = \text{Evolve}(\mathcal{D}_{k,t}; h_t)$	evolved value sets for depth, width, heads, windows, MLP, Q–K–V	ViT blocks, design dimensions, initial choices, budgets

UCB: upper confidence bound; CTS: corpus-guided top-down search; G-VAE: graph variational autoencoder.

input arguments are provided externally [276]. This establishes tool realization, but does not by itself establish autonomous selection of the primitive vocabulary.

Learning candidate representations. Representation-level construction changes the encoding through which candidates are exposed to downstream search, rather than merely changing how often fixed candidates are proposed. In hierarchical generative NAS, a graph variational autoencoder learns embeddings of cell architectures, while clustering based on zero-cost performance proxies identifies families of similar cells. The macro search operates on sequences of cell-family identifiers, channel counts, and strides; conditional generation with CCNF and G-VAE decoding then instantiates these high-level descriptions as concrete architectures. The method first explores this high-level space using evolutionary search (HL-Evo) and subsequently trains a sequence generator on the explored architectures [284]. We therefore attribute its primary learned space abstraction to the graph representation and cell-family abstraction. The human-specified architectural template remains distinct from this learned encoding, as does the sequence generator's role in proposing candidate architectures.

Revising composition and edit operators. Operator-level construction would change the reusable rules that assemble or transform candidates, such as grammar productions or admissible graph rewrites. This differs from adding a function that can be invoked through an unchanged composition language, and from selecting a particular sequence of existing actions. Under this distinction, the reviewed abstraction and tool-learning systems mainly modify $\mathcal{P}_t$, and do not provide a clear primary example of learned $\mathcal{O}_t$. SkillX further illustrates this boundary: although it learns hierarchical reusable skills, these skills are retrieved as execution guidance rather than modifying the underlying composition operators [252]. Such guidance does not by itself demonstrate that AI has changed the legal composition or edit operators. We therefore retain operator construction as a distinct intervention target without assigning these systems to it solely because they describe or perform composition.

Adapting admissibility constraints. Constraint-level construction changes which otherwise expressible candidates are admitted to the active search. S3 represents an architecture space through admissible sets of values for design dimensions, including depth, embedding dimension, attention heads, window size, MLP ratio, and query–key–value dimensions. Their Cartesian product defines the active search space. Using a supernet-based space-quality estimate, S3 evolves these value sets before the final architecture search [32]. With the design variables and architectural building blocks held fixed, we classify this intervention primarily as a change to $\mathcal{K}_t$: it revises the active admissibility conditions rather than simply selecting an individual architecture.

5.2. Evaluation and Minimum Evidence for L2

Having identified the different components of S_t that AI can construct or revise, the next question is whether downstream improvements can actually be attributed to the resulting search space. This requires evaluating the space itself while controlling for other factors that affect search performance.

Evaluating AI-defined search spaces. Evaluation of an L2 system should assess the quality and consequences of the AI-defined search space S_t, rather than only the performance of an individual selected candidate. The quality of a final candidate reflects not only the space construction process, but also the navigation policy G_t, evaluator variance, and search budget. A source-matched L2 evaluation should therefore isolate the contribution of S_t by comparing controlled conditions: the AI-constructed space with a fixed search procedure; a human-designed space under the same search procedure; a randomized or stale AI-derived space with comparable capacity; and the original method with its full procedure. Evaluation should consider

properties of the induced space, including validity, unique coverage, semantic diversity, achievable quality under matched sampling, transfer performance, and search cost.

For induced libraries and toolboxes, space-level evaluation should further consider compression, reuse frequency, interface validity, and held-out task benefit, rather than relying only on performance over the tasks used for construction. Because downstream results depend jointly on S_t and the navigation policy G_t, evaluations should control or match G_t across conditions so that improvements can be attributed to changes in the search space itself.

Such attribution requires evaluation environments where the evaluator and task distribution remain external while the provenance of the search space is systematically varied.

Existing benchmarks. Existing benchmarks provide standardized environments for evaluating search procedures and agent capabilities. In neural architecture search, NAS-Bench-101, NAS-Bench-201, NATS-Bench, and surrogate NAS benchmarks enable reproducible comparisons by providing predefined architecture spaces together with tabulated or learned performance estimates [298, 54, 53, 308]. These resources allow researchers to replay architecture search algorithms under controlled conditions without repeatedly training all candidate architectures.

For tool- and skill-based agents, SWE-bench, OSWorld, AgentBench, AgentBoard, and MLE-bench provide task environments and execution-based evaluation protocols covering software engineering, computer interaction, general agent behavior, and machine learning workflows [109, 288, 151, 167, 27]. Together, these benchmarks establish common grounds for comparing systems that differ in their internal search spaces, tools, or learned capabilities.

Minimum reporting package. An L2 paper should report: (i) the human-provided ambient universe $\mathcal{U}$; (ii) the machine-produced artifact defining S_t; (iii) the changed components of $(\mathcal{R}, \mathcal{P}, \mathcal{O}, \mathcal{K})$; (iv) whether the artifact is persistent and reusable, where applicable; (v) freeze-and-replay or otherwise navigation-matched comparisons under matched compute; and (vi) who supplied direction D_t and evaluator E. Together, these measurements attribute downstream gains specifically to the AI-defined search space.

5.3. Composition with Direction and Evaluation Control

Practical systems often combine L2 with other forms of AI control. It is therefore necessary to distinguish which system components determine direction, search space, and evaluation, rather than assigning a single control level to the system as a whole.

Composition with direction control. Search within an AI-defined space may follow directions specified by humans or predefined rules, and can also be guided by an L1 component that determines which directions to pursue. Voyager exemplifies this combination: its automatic curriculum selects goals, while AI-generated skills expand the available reusable components [253]. Direction control determines what to pursue, while search-space construction defines the representations, primitives, operations, and constraints available for pursuing it. Voyager also uses AI-based self-verification to judge whether a task has been completed, which determines whether the resulting program is committed to its skill library, constituting L3 evaluation control under our taxonomy.

Evaluation remains external. L3 begins when AI constructs or substantively revises what counts as progress. Eureka, Constitutional AI, RLAIF, LLM-as-a-judge, self-rewarding models, and process supervision concern evaluation control rather than search-space provenance [168, 15, 126, 322, 303, 137]. Such mechanisms can coexist with L2, but their contributions should be attributed separately: an AI-generated library remains an L2 artifact when judged by a fixed external evaluator, whereas an AI-generated or substantively revised reward belongs to evaluation control even when the candidate space is entirely human-designed.

L2 should therefore be attributed to the component that constructs or revises S_t, rather than to the system as a whole. A single system may combine direction control, search-space construction, navigation, and evaluation control; when AI additionally begins to construct or revise the criterion E by which progress is judged, the analysis moves from L2 search-space control to L3 evaluation control.

6. L3: AI Provides the Evaluation Signal

The control transferred at L3. L3 transfers control over *the criterion that determines progress*. An AI system qualifies for L3 when it constructs or substantially revises the operative signal used to rank candidates, allocate search effort, accept discoveries, or terminate the loop. Humans may still specify the direction and search space. L3 therefore includes every profile whose evaluation coordinate is AI-provided: $\langle \mathrm{H}, \mathrm{H}, \mathrm{A} \rangle$, $\langle \mathrm{H}, \mathrm{A}, \mathrm{A} \rangle$, $\langle \mathrm{A}, \mathrm{H}, \mathrm{A} \rangle$, and $\langle \mathrm{A}, \mathrm{A}, \mathrm{A} \rangle$. Only the last profile places all three control variables under AI control. The defining property of L3 is the provenance of the evaluation signal, rather than end-to-end autonomy.

The evaluator papers below are counted as AI4AI L3 evidence only when their AI-generated tests, judgments, rewards, or adaptive environments enter a discovery loop and change candidate selection, training, or acceptance. Standalone benchmark scoring, answer ranking, or post hoc review remains an evaluation technique, not an AI4AI intervention.

6.1. The Object of Attribution: the Operative Evaluation Signal

Evaluation as a structured artifact. At iteration t, represent the operative evaluation object as

$$E_t = (\mathcal{T}_t, J_t, A_t, \tau_t),$$

where $\mathcal{T}_t$ contains tests, tasks, or environments; J_t maps evidence to local judgments; A_t aggregates judgments across criteria, samples, or evaluators; and τ_t defines acceptance, rejection, or stopping thresholds. L3 requires an AI-generated component to enter the subsequent discovery loop:

$$\widehat{E}_t \sim P_E(\cdot \mid x_t, D_t, S_t, h_t, \mathcal{M}), \qquad q_{t,i} = \widehat{E}_t(c_{t,i}; x_t, S_t, D_t), \qquad c_{t+1} = \mathrm{Select}(\{c_{t,i}, q_{t,i}\}_{i=1}^{n_t}).$$

Here h_t is the accumulated research history used in Chapter 2, $\mathcal{M}$ denotes the model or agent that constructs the evaluator when such a model is present, and $c_{t,i} \in \mathcal{C}_t$ is an individual candidate. The evaluator may be instantiated before candidate scoring from x_t, D_t, S_t, h_t, or it may update online with additional history absorbed into h_t; in either case, $q_{t,i}$ is the candidate-level evidence used by selection. The selected candidate c_{t+1} is the candidate-level counterpart of the target update x_{t+1} in the full discovery cycle. The sources of direction D_t and search space S_t do not determine the L3 label. The relevant criterion is whether AI generates a component of $\widehat{E}_t$ and whether that component causally governs selection.

Seed specification, learned mapping, and operative authority. Human involvement is compatible with L3. People may provide a constitution, examples, a broad rubric, or an executable interface, while AI generates

tests, instantiates criteria, learns a reward mapping, generates reward code, or determines how evidence is aggregated. The attribution question is not whether humans appear upstream, but whether replacing the AI-generated component changes which candidates are judged better. A frozen, human-authored benchmark scored by deterministic execution remains a human evaluation, even if an AI agent reads its failures. An AI-generated test suite, learned verifier, or generated reward program qualifies as L3 when it determines subsequent selection.

The ranking-intervention test. A strong L3 claim should make $\widehat{E}_t$ inspectable and replay the same candidate set under four conditions: the AI-generated evaluator, a human-authored anchor, a capacity-matched stale or randomized evaluator, and an independent holdout evaluator. The analysis should report rank correlation, pairwise disagreement, calibration, false-accept and false-reject rates, and downstream regret. It should repeat these comparisons after swapping candidate order and model identity, while matching output length and style and checking for contamination. If replacing the evaluator leaves the search trajectory unchanged, the evaluator provides auxiliary commentary rather than operative evaluation control.

Figure 7 summarizes how an AI-provided evaluation signal connects human anchors and generated evaluation components to consequential discovery decisions. The separate validation branch audits candidate gains using independent anchors; it does not grant the optimizing system control over the external evaluator.

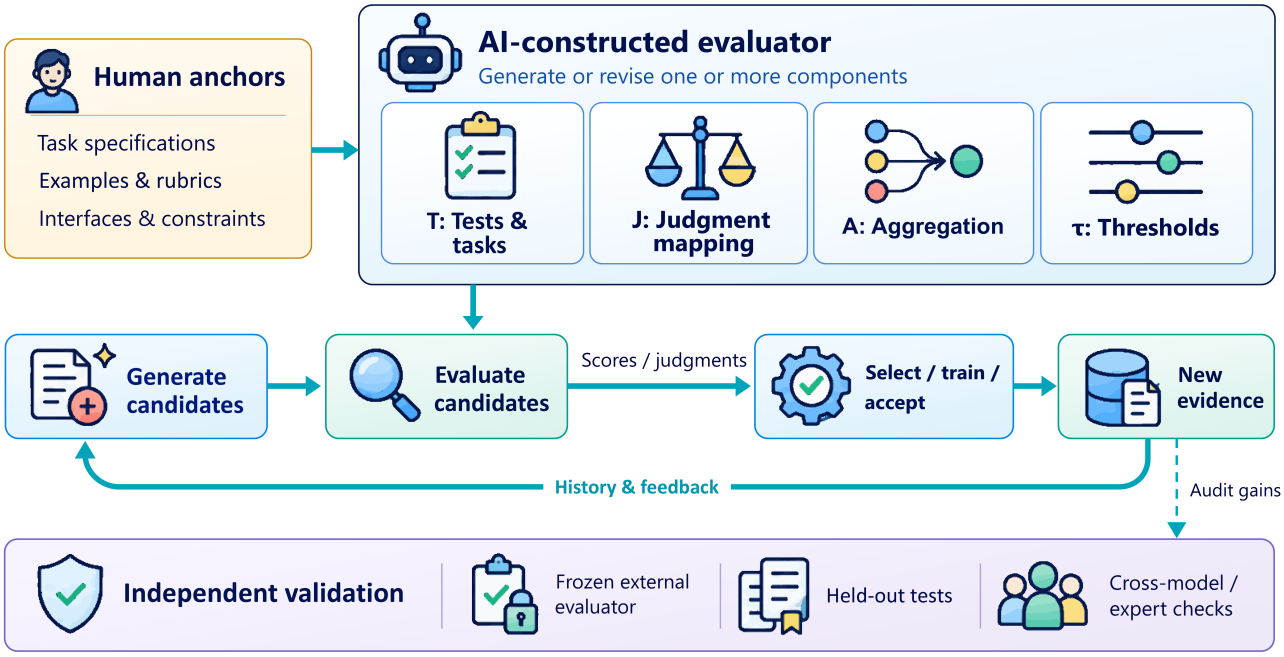


Figure 7: **AI-provided evaluation signals in the discovery loop.** Human specifications, examples, and interfaces anchor an evaluator in which AI generates or substantially revises one or more components: tests and tasks $\mathcal{T}$, the judgment mapping J, aggregation rules A, or decision thresholds τ. The resulting scores or judgments govern candidate selection, training, or acceptance, and new evidence supports subsequent discovery. The independent-validation branch audits gains with a frozen external evaluator and held-out or cross-model checks; it remains distinct from the operative signal used for optimization. The sources of direction and search space are attributed separately.

Figure 8 organizes L3 work by the evaluation artifact supplied by AI. Table 5 uses the decomposition introduced for L1 and L2 and distinguishes the human anchor from the machine-generated signal that enters the loop.

6.2. Executable Evaluation Artifacts

Generated tests can become operative, but generation alone is insufficient. CodeT uses language-model-generated tests and a dual execution-agreement score, combining agreement with the generated tests and with other sampled programs, to select a program at inference time [28]. ChatUniTest constructs unit tests tailored to the focal Java method and validates or repairs those tests. CodeJudge uses an LLM to estimate semantic code correctness without test cases [39, 248]. These methods can act on different parts of E: test synthesis changes $\mathcal{T}$, whereas semantic judging changes J. The technical ability to generate a test or judgment does not by itself establish L3. ChatUniTest and CodeJudge count as L3 only when their generated tests or semantic judgments enter candidate selection, training, resource allocation, or an operative accept–reject rule; test repair, annotation, or post hoc scoring remains evaluation support. Their evidence is strongest when generated tests are checked for validity, coverage, independence from the candidate generator, and agreement with hidden tests.

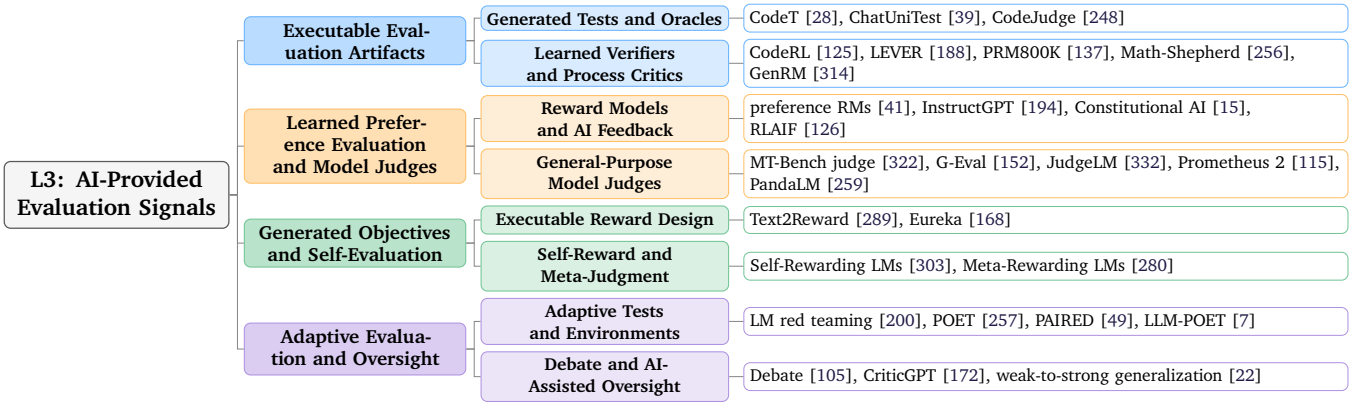


Figure 8: **Taxonomy of L3 AI-provided evaluation signals.** The literature separates into executable evaluation artifacts, learned preference evaluation and model judges, generated objectives and self-evaluation, and adaptive evaluation and oversight. An L3 artifact may change the test distribution $\mathcal{T}$, judgment mapping J, aggregation rule A, or decision threshold τ, and must causally affect the discovery loop. The leaf nodes are candidate artifacts or representative methods, not unconditional L3 assignments; only configurations that satisfy the closed-loop counterfactual in Section 6.1 count as L3. Robustness and recursive-evidence checks are cross-cutting validation protocols applied across these families, not an additional artifact family.

Outcome verifiers learn a pass–fail boundary. CodeRL trains a critic to predict functional correctness and feeds its dense feedback into actor updates and critical sampling. LEVER combines execution results with a learned verifier to rerank programs generated by a language model [125, 188]. Cobbe et al. train an outcome verifier on final-answer correctness and use it to select among sampled GSM8K solutions with Best-of-N [43]. CRITIC instead runs a generate, tool-check, and revise loop for individual outputs. Its feedback can guide revision, although the paper neither trains a reward model nor explicitly ranks a population [84]. CRITIC counts as L3 only when the critique controls the operative revision or accept–reject decision in the analyzed discovery loop; a one-off tool check or post hoc critique remains an evaluation technique.

Process supervision localizes evaluation. Outcome rewards score complete trajectories, whereas process rewards score intermediate steps and can reveal where a derivation first fails. PRM800K provides human step labels. Its solution score is the product of step-positive probabilities; minimum reduction is evaluated as an alternative for Best-of-N selection [137]. Math-Shepherd estimates step quality from rollout success without human step annotations and uses the resulting process reward for verification and PPO [256]. Generative verifiers recast verification as next-token prediction: GenRM scores a candidate by the probability of the "Yes" token, while GenRM-CoT aggregates scores across sampled verification rationales [314]. These methods move different parts of E from fixed specifications and final-answer checks to AI-generated judgments; their labels, rollouts, and aggregation procedures should therefore be reported separately.

6.3. Learned Preference Evaluation and Model Judges

Preference models turn comparisons into a reusable objective. Preference-based reinforcement learning learns a reward function from human comparisons, and InstructGPT uses the inferred reward to guide PPO updates [41, 194]. These are mixed-control L3 examples when the learned mapping is the operative score consumed by optimization: humans provide the comparisons, while the learned mapping produces the signal. Constitutional AI replaces part of direct human feedback with model-generated critiques, revisions, and

Table 5: Evaluation-signal decomposition of representative L3 candidates, grouped by the artifact families of Figure 8. Each row separates the AI-produced component $\widehat{E}_t$ from its human anchor E^{H}.

Method	Evaluation policy	AI-produced signal $\widehat{E}_t$	Human anchor E^{H}
Executable evaluation artifacts			
CodeT [28]	$q(c) = \alpha A_{\text{test}}(c, \widehat{\mathcal{T}}) + (1-\alpha) A_{\text{peer}}(c)$	generated test suite $\widehat{\mathcal{T}}$; dual execution-agreement score	problem statement, language runtime, execution semantics
Math-Shepherd [256]	$q(z) = \prod_k p_\phi(\text{positive} \mid z_{\le k})$	auto-built step labels; learned process reward model	math problems, final-answer correctness checks
GenRM [314]	$q(y) = p_\theta(\text{Yes} \mid x, y, I)$	generative verifier that judges correctness as tokens	verification demonstrations, supervised task corpus
Learned preference evaluation and model judges			
Constitutional AI [15]	$q(y) = R_\phi(y \mid x, \mathcal{C})$	critiques, revisions, preferences, learned reward	written constitution $\mathcal{C}$, seed helpfulness data
RLAIF [126]	$q(y) = R_\phi(x, y; \widehat{p}_{\text{AI}})$	AI preference labels $\widehat{p}_{\text{AI}}$; fitted reward model	task prompts, principles, calibration examples
G-Eval [152]	$q(y) = \sum_i p(s_i \mid x, y, r)\, s_i$	CoT criterion use; probability-weighted score	human criteria r, task inputs, reference context
Prometheus 2 [115]	$q(y) = J_\phi(x, y_{[1:2]}, r, y^*)$	trained rubric-conditioned direct or pairwise judgment	rubric r, reference answer y^*, evaluation examples
Generated objectives and self-evaluation			
Eureka [168]	$q(\pi) = \mathbb{E}_{\tau\sim\pi}[\widehat{R}_\psi(\tau)]$	reward program $\widehat{R}_\psi$ evolved from training feedback	environment code, natural-language objective, simulator dynamics
Self-Rewarding LMs [303]	$q(y) = J_\theta(x, y; r)$	responses, self-judgments, preference-training pairs	seed instruction data, judging prompt, training procedure
Adaptive evaluation and oversight			
LM red teaming† [200]	$q(M) = \mathbb{E}_{x\sim\widehat{\mathcal{T}}_t}[H(M(x))]$	adaptive prompt distribution $\widehat{\mathcal{T}}_t$	harm taxonomy, target model, safety policy
PAIRED† [49]	$q(e) = R(\pi_A, e) - R(\pi_P, e)$	regret-shaped adaptive environment	generator interface, agents, task dynamics
CriticGPT† [172]	$q(y) = A(J_\phi(x, y), J_{\text{H}}(x, y))$	learned critique that guides review	human reviewer, code task, known-error supervision

† Typically used for testing or review; L3 only if the AI signal changes downstream evaluation, selection, or operative acceptance.

preferences conditioned on written principles. RLAIF trains a reward model from AI-generated preferences and uses that model for reinforcement learning [15, 126]. In both systems, the human constitution or task specification serves as an anchor, whereas the labels and reward are machine-generated; the L3 designation still requires that this machine-generated signal change training, selection, or acceptance.

Model judges often function as evaluation technologies rather than closed loops. MT-Bench, G-Eval, JudgeLM, Prometheus, and Prometheus 2 introduce language-model evaluators, rubrics, and tests of alignment with human judgments. Their original papers mainly report benchmark scores and evaluator quality rather than downstream candidate optimization [322, 152, 332, 114, 115]. They therefore become L3 components only when their ratings influence candidate survival, resource allocation, or acceptance. PandaLM is a narrower closed-loop case: its judge feeds or supports instruction-tuning hyperparameter

selection, while its synthetic GPT-3.5 judgments and human holdout serve distinct evidentiary roles [259]. Treating PandaLM as L3 requires evidence that these judgments causally alter which configuration is retained or receives resources, rather than merely informing an offline comparison.

Rubric generation changes the measurement model. A fixed human rubric delegates scoring but not the selection of criteria. A stronger L3 system derives criteria, weights, counterexamples, and disqualifying conditions from the task and then applies them. Reports should state whether AI changed $\mathcal{T}$, J, A, or τ, whether those changes entered optimization, and whether generated criteria survived expert review or predicted delayed real-world outcomes.

6.4. Generated Objectives and Self-Evaluation

Executable reward design exposes the evaluator as code. Text2Reward translates a natural-language objective into executable reward code for reinforcement learning. Its term "data-free" refers to the absence of offline demonstrations, not to the absence of human task specifications or optional feedback. Eureka uses an LLM to synthesize and iteratively improve reward programs from environment code and training feedback [289, 168]. The resulting program can be versioned, executed, ablated, and supplied to a different policy optimizer. This makes generated rewards unusually auditable L3 artifacts when the generated program is the operative training or selection signal. Eureka also illustrates compositional control. Reward-code evolution can additionally function as L1 direction over prior reward candidates if the configuration uses those alternatives to decide which objective to pursue; otherwise it remains part of the L3 evaluator. The underlying simulator and action interface may remain human-provided.

Self-reward closes the generator–evaluator loop. Self-Rewarding Language Models use the model itself as a judge to create preference data for iterative training, placing response generation and part of evaluation in one shared loop [303]. Meta-Rewarding Language Models add a meta-judge that evaluates the quality of the model's own judgments, creating a second-order evaluator over J_t [280]. These systems are structurally important because evaluator improvement can amplify candidate improvement. However, shared-model errors, self-preference, and feedback collapse make independent anchors essential. Reported gains should therefore be separated from external benchmark validation, which does not itself provide the training signal.

6.5. Adaptive Oversight, Robustness, and Recursive Evidence

Adaptive evaluation searches for failures. Language-model red teaming generates adversarial questions that elicit harmful behavior and changes the test distribution in response to the system under evaluation [200]. POET co-evolves environments and agents; PAIRED trains an adversarial environment generator through regret; and LLM-POET uses language models to propose and mutate environments [257, 49, 7]. Their evaluation artifact is not a static score but an adaptive distribution $\mathcal{T}_t$ whose difficulty and coverage evolve with the candidate population. An adaptive $\mathcal{T}_t$ is only operative L3 evidence when its changes alter the judgment mapping, aggregation, threshold, or candidate selection/training in the analyzed loop. If it only generates challenge cases or acts as a curriculum while the evaluator and acceptance rule remain fixed, it is an adaptive test or training artifact rather than L3 evaluation control.

Scalable oversight distributes judgment across agents. AI safety via debate proposes adversarial arguments as a way to help a human judge questions that are difficult to evaluate directly [105]. Scalable-oversight

work studies whether models and assistance protocols can extend human supervision, while weak-to-strong generalization asks whether weak supervision can elicit stronger model capabilities [21, 22]. CriticGPT trains a critic to identify subtle errors in model-written code and improve human review quality [172]. These protocols combine human anchors with AI-produced evidence, but neither debate, CriticGPT, nor weak-to-strong generalization is automatically L3. They count as L3 only when the generated arguments, critiques, or supervision signals change the operative J, A, or τ and thereby alter selection, training, or acceptance in the discovery loop.

Evaluation-guided search and scientific review. An AI score matters for AI4AI when it changes parent selection, mutation, experiment allocation, early stopping, or acceptance. Candidate-level evaluations should therefore be linked to a trajectory-level quantity such as cumulative independent utility, best-so-far quality, or discovery regret. This causal view separates systems that merely summarize experimental outputs from systems whose evaluators control the next experiment.

Scientific review is a compound evaluator. The AI Scientist combines idea generation, experiment execution, manuscript production, and automated review; its successor increases the degree of end-to-end automation [158, 292]. Automated review can apply criteria such as novelty, soundness, clarity, and significance, but it remains a proxy for scientific value. A rigorous decomposition should expose the review rubric, reviewer model, aggregation procedure, acceptance threshold, and any literature or execution checks. Review-model scores should be calibrated against blinded expert judgments and later reproducibility rather than treated as ground truth.

Evaluator cascades can combine heterogeneous evidence. Low-cost unit tests may filter obvious failures before expensive learned judges, formal checkers, or expert audits. Such cascades should report which stage has veto power and how uncertainty propagates. Otherwise, an apparently independent final evaluator can inherit a biased candidate pool created by an earlier AI judge.

Optimization pressure exposes proxy mismatch. Reward-model performance can degrade when policies are optimized beyond the data regime on which the reward was trained [80]. Reward hacking occurs when proxy improvement lowers the intended reward. Goal misgeneralization occurs when an agent pursues the wrong objective out of distribution, and sycophancy occurs when preference optimization favors agreement over truth [227, 124, 220]. These failure modes are related but distinct. The distinction matters at L3 because the evaluator is a mutable component inside the optimization loop.

Model judges have systematic preferences. LLM evaluators can exhibit position bias and inconsistent pairwise preferences [255]. Broader audits find sensitivity to model identity and to demographic or stylistic features; evaluators may also prefer outputs generated by themselves or related models [297, 265]. Evaluator ensembles can reduce some forms of overoptimization and variance in synthetic settings, but correlated training data and architectures preserve shared blind spots [44]. Optimization may therefore follow a proxy reward model while leaving the latent objective unchanged. Ensembles mitigate this risk but do not eliminate it [59].

Minimum reporting package. An L3 paper should report: (i) the human-provided specification and data; (ii) the AI-generated components of $(\mathcal{T}, J, A, \tau)$; (iii) how the signal changes selection; (iv) agreement and calibration against independent anchors; (v) robustness under increased optimization pressure; (vi) sensitivity to evaluator identity, prompts, ordering, and contamination; and (vii) the provenance of direction and search space. Useful anchors include executable ground truth, formal verification, hidden human-authored tests, cross-model judging, delayed real-world outcomes, preregistered protocols, and randomized expert audits.

Self-evaluation is necessary but not sufficient. STOP uses a language model to improve a scaffolding program that recursively improves solutions. The Darwin Gödel Machine iteratively edits an agent and empirically validates proposed modifications [310, 312]. Under the fixed boundaries used in the application table, both systems are neighboring L1 systems. They become L3 only if an AI-produced evaluator or acceptance rule governs the next modification. Self-rewarding and meta-rewarding systems, by contrast, place learned evaluators inside iterative improvement loops. These systems demonstrate increasingly tight feedback between candidate and evaluator, but a single generation can benefit from evaluator drift, benchmark overfitting, or lucky search.

Claims must survive a frozen external evaluator. A recursive-improvement claim should show that generation $g+1$ improves the mechanism that produces generation $g+2$. It should also show that gains persist across multiple generations and that each generation remains better under an evaluator frozen before the run. The evaluator itself must be versioned so that criterion changes are not mistaken for candidate improvement. AI4AI-Bench begins to isolate agents' ability to modify training algorithms rather than merely tune hyperparameters or collect data [40]. Stronger evidence would combine such mechanism-level tasks with held-out domains, compute-matched baselines, lineage tracking, rollback tests, and external evaluation that the improving system cannot inspect or rewrite. L3 therefore marks control over evaluation; it becomes evidence for recursive self-improvement only when evaluator improvement and candidate improvement remain distinguishable across generations.

7. Applications and Benchmarks

Complementing the analysis of control in Sections 4–6, this section offers a distinct perspective through applications, examining what AI4AI improves and how those improvements are evaluated across domains. We organize the discussion into four families: *AI4AI Learning*, *AI4AI Inference*, *AI4AI Data*, and *AI4AI Infrastructure*. These cover improvements to learning procedures, frozen-model behavior, data, and the software and hardware stack. Figure 10 groups representative methods by these targets. The tables select representative methods that illustrate distinct application artifacts, retaining the earlier level labels and summarizing AI-produced artifacts alongside the human-provided setup. Methods with several applications may contribute to more than one family. Research workflows and shared evaluation components connect these families, while Table 10 connects representative benchmarks to AI4AI development tasks and their validation.

7.1. AI4AI Learning

AI4AI Learning improves procedures that update a target model's parameters. Its artifacts include configurations, pipelines, architectures, optimizers, learning rules, rewards, adapters, and compressed or distilled

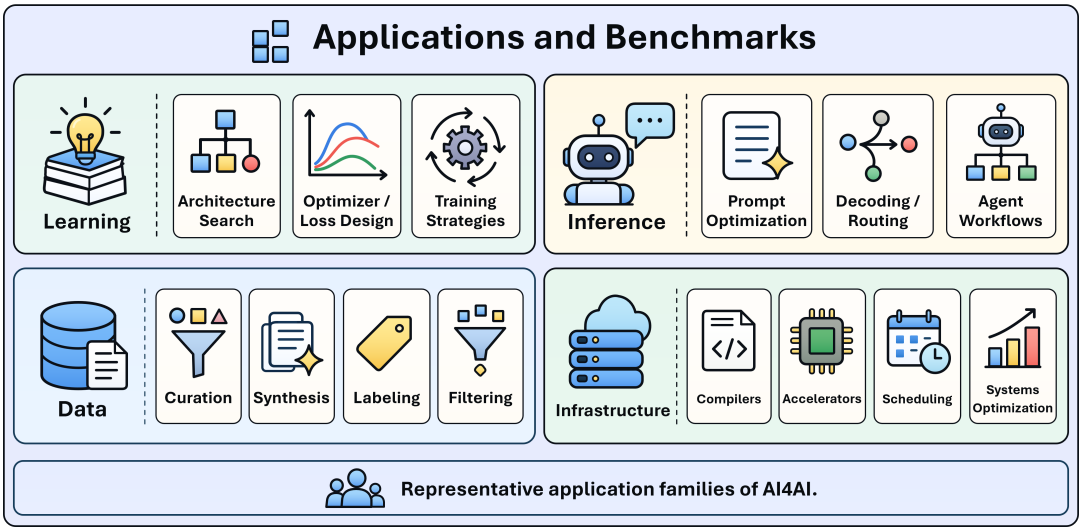


Figure 9: Overview of AI4AI applications organized by the target of improvement: Learning, Inference, Data, and Infrastructure.

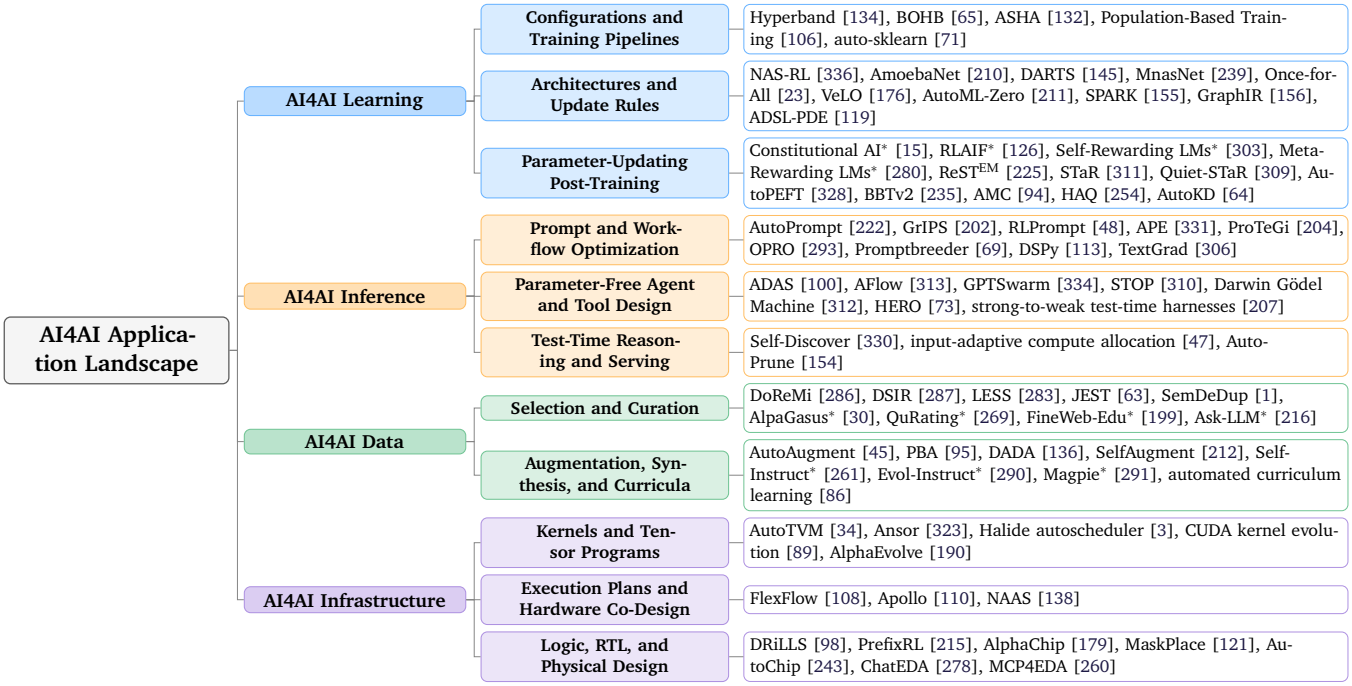


Figure 10: **Taxonomy of AI4AI applications.** Representative methods are organized by the artifact being improved: learning procedures, inference-time behavior and computation, data construction, and infrastructure artifacts or tool flows. Benchmarks are listed separately in Table 10. This application axis is orthogonal to the L0–L3 control hierarchy: methods in any branch may differ in whether AI controls the search direction, search space, or evaluation signal. A superscript * marks the same configuration-dependent condition used in the application tables; the figure is representative rather than exhaustive.

models. This includes continuous prompt vectors and recovery training. The resulting model is evaluated after the discovered procedure has been applied.

Table 6: Representative AI4AI Learning methods, selected to illustrate distinct training artifacts. Level labels follow Sections 4–6.

Method	Level	AI-designed	Human-fixed
Configurations and training pipelines			
Hyperband [134]	L1	configurations promoted at each rung	search space, budget schedule, fidelity metric
auto-sklearn [71]	L1	pipeline, hyperparameters, ensemble	pipeline grammar, meta-tasks, validation metric
PBT [106]	L1	per-member hyperparameter schedule	config space, mutation operators, train budget
Architectures and update rules			
DARTS [145]	L1	mixed-op weights, then discrete cell	operator set, search space, validation loss
S3 [32]	L2	admissible architectural value sets	ViT blocks, initial choices, budgets
VeLO [176]	L1	learned update-rule weights	optimizer family, meta-training distribution
AutoML-Zero [211]	L1	setup, predict and learn programs	primitive set, function set, meta-distribution
Parameter-updating post-training			
Constitutional AI [15]	L3	critiques, revisions, preference labels	constitution, harm rules, base model
Eureka [168]	L3	executable training reward programs	task description, simulator, policy optimizer
STaR [311]	L1	rationales and rationalizations	seed rationales, answer key, fine-tuning recipe
Adaptation and compression			
AutoPEFT [328]	L1	adapter types, sizes, insertion layers	adapter space, performance and parameter cost
AMC [94]	L1	per-layer compression ratios	pruning space, latency constraint, recovery

7.1.1. Configurations and Training Pipelines

Hyperparameter optimization uses trial outcomes to choose configurations and allocate training resources. Bayesian optimization learns a response surface over validation outcomes [230]. Freeze–Thaw Bayesian optimization, Hyperband, BOHB, ASHA, and DEHB use partial learning curves or adaptive allocation to concentrate resources on promising trials [238, 134, 65, 132, 13]; SMAC3 and Optuna support conditional and mixed domains [139, 8]. Random search provides a reference for the benefit of adaptation [19].

AutoML systems expand the target from a configuration to a pipeline composing preprocessing, model choice, and training [71, 70, 335]. Population-Based Training adapts schedules during training by copying model states and mutating hyperparameters [106]. Comparisons should include the full trial budget and the cost of interrupted, resumed, and unsuccessful runs.

7.1.2. Architectures and Update Rules

Neural architecture search optimizes cells and macro-structures through reinforcement learning, evolution, progressive expansion, learned predictors, differentiable relaxation, and shared-weight training [336, 210, 140, 160, 201, 145]. Network Morphism reuses trained networks through function-preserving transformations, while BANANAS and Neural Predictor guide proposals using evaluated architectures [267, 271, 268, 270]. Performance prediction and zero-cost proxies reduce screening cost [16, 173, 273]. MnasNet and ProxylessNAS incorporate device latency, and Once-for-All and BigNAS extract device-specific subnetworks from a trained supernet [239, 24, 23, 299]. EfficientNet combines a searched base network with compound scaling [240].

The search domain can itself be adapted to the workload. DDSAS dynamically selects active edge–operation

combinations; hierarchical generative NAS learns cell embeddings and cell-family abstractions; and S3 evolves admissible sets of architectural dimensions before the final search [295, 284, 32]. These are the architecture-design applications of the L2 methods in Section 5. LLMatic uses language-model variation and a quality-diversity archive to explore architectures [185]. SPARK, GraphIR, and ADSL-PDE expose structured edit surfaces through factors, intermediate representations, and a domain-specific language for neural solvers [155, 156, 119].

The update rule itself can be discovered. Learned optimizers and VeLO learn neural update rules; symbolic optimizer search and evolved reinforcement-learning losses discover compact learning programs; and AutoML-Zero evolves complete algorithms from mathematical primitives [10, 274, 176, 36, 42, 211]. MAML learns initial parameters for rapid adaptation across tasks [72]. Transfer to held-out workloads, longer horizons, larger models, and altered data regimes tests the usefulness of the discovered procedure. Evaluation should place discovery or meta-training cost alongside downstream learning curves.

7.1.3. Parameter-Updating Post-Training

Learned feedback supports alignment and task-specific training. Preference-based reinforcement learning and InstructGPT fit rewards from comparisons; Constitutional AI and RLAIF use model-generated critiques or preferences to guide training [41, 194, 15, 126]. Self-Rewarding Language Models produce their own preference judgments, Meta-Rewarding adds a judge of those judgments, and PandaLM applies model judgments to instruction-tuning hyperparameter selection [303, 280, 259]. These applications reuse the evaluation mechanisms reviewed in Section 6.3.

Executable reward design provides feedback for reinforcement learning. Text2Reward translates task descriptions into reward code, and Eureka revises reward programs using environment code and training results [289, 168]. CodeRL trains a correctness critic for code-policy updates, PRM800K supplies human step supervision, and Math-Shepherd builds step-quality labels from rollout outcomes and uses process rewards for verification and PPO [125, 137, 256]. Self-training provides a complementary route: ReST^{EM} retrains on correctness-filtered solutions, STaR fine-tunes on successful rationales and rationalizations, and Quiet-STaR learns internal rationales through token-prediction feedback [225, 311, 309].

AutoPEFT and NOAH search trainable adaptation modules, while BBT and BBTv2 optimize continuous prompt vectors [328, 319, 236, 235]. AMC, MetaPruning, AutoCompress, HAQ, AutoQNN, and AutoKD optimize pruning, quantization, or distillation choices with subsequent training [94, 153, 148, 254, 83, 64]. Reports should separate discovery cost from adaptation or recovery cost and include trainable parameters, optimizer-state memory, and final model accuracy.

Benchmarks for AI4AI Learning. NAS-Bench-101, NAS-Bench-201, NATS-Bench, and NAS-Bench-301 provide exact or surrogate outcomes over architecture spaces; TransNAS-Bench-101 adds task diversity and HW-NAS-Bench adds device costs [298, 54, 53, 224, 308, 56, 128]. HPO-B and HPOBench provide reproducible black-box and multi-fidelity optimization tasks [11, 58]. For learned architecture spaces, comparisons should run the same search procedure over the constructed space and a human-designed counterpart at matched cost, following Section 5.2.

AI4AI-Bench asks agents to modify training algorithms in frozen repositories and reruns submissions from scratch under a hidden evaluator [40]. RE-Bench compares agents with human experts at matched time budgets on training, scaling-law, and kernel-optimization tasks [275]. Automated LLM Speedrunning measures reproduction of known training-speed improvements [321]. MLAgentBench, MLE-bench, MLGym,

PaperBench, and MLReplicate assess experiment implementation and result replication [103, 27, 186, 233, 76].

PostTrainBench adds autonomous LLM post-training under a bounded compute budget, allowing agents to develop training strategies, curate data, and run experiments [209]. MLS-Bench requires improvements to targeted method components to generalize across controlled settings and scales, with edit scopes, capacity constraints, and repeated evaluation limiting alternative sources of gain [166]. Its catalog also includes ML systems tasks, so individual tasks may belong to other application families. These protocols strengthen evidence for method effectiveness and transfer; scientific novelty requires additional assessment.

7.2. AI4AI Inference

AI4AI Inference improves behavior while target-model and adapter parameters remain frozen. Its artifacts include discrete prompts and demonstrations, workflows, agent scaffolds, reusable tools and libraries, routing and context policies, and test-time computation policies. A search controller or recognition model may be trained to produce these artifacts; the target model is used without parameter updates.

7.2.1. Prompt and Workflow Optimization

AutoPrompt, GrIPS, and RLPrompt search discrete prompts using token substitutions, phrase edits, or a learned prompt policy [222, 202, 48]. APE generates and ranks instructions, ProTeGi uses error critiques to guide edits, OPRO conditions proposals on previous prompts and scores, and EvoPrompt and Promptbreeder apply evolutionary search [331, 204, 293, 90, 69]. PromptAgent plans prompt revisions, while TEMPERA learns test-time prompt-editing policies [258, 316].

Compound applications optimize several prompts or model calls together. SAMMO searches structured prompt programs, DSPy compiles language-model pipelines, MIPRO jointly selects instructions and demonstrations, and TextGrad propagates textual feedback through a computation graph [218, 113, 192, 306]. PromptWizard, CriSPO, and instruction-exemplar selection use task analysis or multi-aspect critiques, while local prompt optimization focuses on constrained edits [4, 92, 251, 107]. GEPA combines execution-trace reflection with a Pareto archive to select targeted changes [6]. Comparisons should report optimization cost and transfer across tasks and models; OPRO re-evaluations also motivate disclosing the optimizer model, scorer, meta-instruction, and token budget [317].

7.2.2. Agent, Tool, and Library Design

ADAS searches agent programs containing roles, tool calls, memory, and control flow; AFlow searches code-represented workflows; and GPTSwarm optimizes node prompts and graph connectivity [100, 313, 334]. STOP and the Darwin Gödel Machine revise improver or agent code across validated iterations [310, 312]. HERO extracts and recombines program edits for applications including agent design [73]. AI4AI at Test-Time studies a stronger builder that constructs routing, deterministic code, and answer-format controls for a weaker frozen target [207].

Reusable tools change the building blocks available to an agent. CREATOR constructs tools before planning their use, LATM caches generated Python tools, and Toolink builds task-family toolkits [205, 25, 206]. CRAFT abstracts solutions into a specialized toolset, TroVE grows and trims reusable functions, and ToolLibGen aggregates tools into libraries with unified interfaces [302, 264, 305]. Voyager accumulates executable skills together with a curriculum and self-verification, DynaSaur expands available actions, and Memento-Skills

Table 7: Representative AI4AI Inference methods, selected to illustrate distinct prompts, agents, libraries, and computation policies. Target-model and adapter parameters stay frozen.

Method	Level	AI-designed	Human-fixed
Prompt and workflow optimization			
APE [331]	L1	candidate instructions; final pick	proposal prompt, output format, task metric
GEPA [6]	L1	reflection-guided edits and candidate selection	editable prompts, task feedback, budget
DSPy [113]	L1	module instructions, few-shot demos	program signature, module set, program metric
Agent, tool, and library design			
ADAS [100]	L1	agent code: roles, tools, control flow	tool pool, role template, task and holdout sets
DreamCoder [61]	L2	reusable program abstractions	base DSL, execution semantics, task examples
CRAFT [302]	L2	specialized reusable tool library	runtime, task APIs, training tasks, validators
Voyager [253]	L3	skills, curriculum, success judgments	Minecraft APIs, environment mechanics
Test-time reasoning and serving			
Self-Discover [330]	L1	chosen modules; reasoning structure	module library, selection rules, task metric
Input-adaptive allocation [47]	L1	learned per-input compute budget	budget grid, cost model, task mix
AutoPrune [154]	L1	token-pruning policy in the DSL	pruning DSL, token budget, task metric

refines skills comprising specifications, scripts, and prompts [253, 187, 329]. ToolMaker realizes tools from supplied repositories, and SkillX constructs skill knowledge used as execution guidance [276, 252].

Program-library learning provides a complementary route to reusable inference-time components. EC^2, DreamCoder, and drawing-program induction learn reusable subroutines; LAPS adds language supervision; Stitch compresses recurring program structure; and LILO combines synthesis, compression, and documentation [60, 61, 244, 277, 20, 85]. These applications connect the library-construction work in Section 5 to downstream program synthesis and agent development. Their value depends on interface validity, reuse, and held-out task benefit, together with library construction and maintenance cost.

7.2.3. Test-Time Reasoning and Serving

Self-Discover selects and adapts reasoning modules into a task-specific structure [330]. Input-adaptive allocation learns how to distribute computation across inputs, and AutoPrune searches training-free visual-token pruning policies for frozen multimodal models [47, 154]. Best-of-N, majority voting, and fixed verifier-guided search provide reference quality–cost curves for these policy-design applications [229]. Useful measurements include generated tokens, verifier calls, tool calls, latency, memory, and energy.

Verification components support selection and refinement of generated programs and reasoning traces. CodeT selects programs using generated tests and execution agreement, LEVER combines execution with a learned verifier, and outcome verifiers select among sampled mathematical solutions [28, 188, 43]. GenRM produces generative correctness judgments, and CRITIC uses tool feedback to guide revision [314, 84]. These mechanisms supply evaluation components for the AI4AI loops reviewed in Section 6. Their contribution to a developed prompt, agent, or computation policy can be measured by replacing the verifier while matching candidates and serving budgets.

Benchmarks for AI4AI Inference. AIE-Bench evaluates agents that build or modify agents, and VeRO-Bench evaluates versioned target–optimizer pairs under controlled budgets [180, 250]. AgentBench, AgentBoard, and GAIA assess interactive task performance; SWE-bench and OSWorld provide software and computer-interaction environments for evaluating developed agents and tools [151, 167, 178, 109, 288]. Code-to-Play and ConceptSearch add program-search tasks over games and ARC-like problems [57, 226]. Prompt and workflow studies should report optimization cost alongside task scores, with target models, tool access, and serving budgets held comparable.

AutoHarnessBench separates a proposer from a fixed-weight actor: the proposer revises the actor's harness using bounded public feedback, and an independent evaluator compares the frozen result with the initial harness on held-out tasks [228]. τ^{τ}-Bench extends agent construction to client requirements, business records, inherited code, production APIs, and serving constraints, evaluating the delivered agent against held-out simulated users [221]. These protocols assess the development intervention as well as the resulting agent's behavior.

7.3. AI4AI Data

AI4AI Data improves the examples, mixtures, transformations, and curricula used to develop AI systems. The discovered artifact is the data or data policy; downstream training tests its usefulness. Comparisons use a shared training recipe and report data construction and processing costs.

7.3.1. Selection and Curation

DoReMi optimizes domain weights with a proxy model; DSIR resamples toward a target distribution; LESS selects instruction data through gradient alignment; and JEST jointly selects batches using learner and reference-model losses [286, 287, 283, 63]. SemDeDup supplies a semantic deduplication baseline [1]. Influence functions, TracIn, dataset cartography, and forgetting events offer complementary signals for diagnosing and selecting examples [117, 203, 237, 247]. Rankings should be validated on disjoint data and across architectures.

Model-judged filtering makes quality evaluation part of corpus construction. AlpaGasus rates instruction–response pairs, QuRating learns a corpus rater from pairwise judgments, FineWeb-Edu distills educational-value annotations into a classifier, and Ask-LLM judges informativeness for pretraining [30, 269, 199, 216]. These applications reuse the model judges reviewed in Section 6.3. Downstream value should be evaluated independently of the rater, including corpus coverage and sensitivity to rating prompts and thresholds.

7.3.2. Augmentation, Synthesis, and Curricula

AutoAugment, Population Based Augmentation, DADA, and SelfAugment search transformation policies or schedules [45, 95, 136, 212]. SelfAugment uses a self-supervised proxy in place of labeled validation accuracy. Policies should be tested across child architectures and datasets, with controller cost separated from final training cost.

Self-Instruct, Evol-Instruct, Orca, UltraChat, and Magpie generate instructions, responses, or dialogues. Cosmopedia generates textbook-style pretraining text, and Nemotron-CC combines quality filtering with synthetic rephrasing [261, 290, 183, 51, 291, 18, 234]. The generator, quality filter, deduplication rule, and downstream recipe can be ablated separately at matched token budgets. LLM-FE adds a feature-engineering application by evolving programs for tabular learning [2].

Table 8: Representative AI4AI Data methods, selected to illustrate distinct data artifacts and policies.

Method	Level	AI-designed	Human-fixed
Selection and mixing			
DoReMi [286]	L1	domain mixture weights	domain partition, reference model, excess-loss objective
LESS [283]	L1	gradient-similarity scores; subset	target examples, warm-up recipe
Model-judged quality filtering			
QuRating [269]	L3	pairwise judgments; trained rater	named quality criteria, sampling rule
Augmentation and feature construction			
AutoAugment [45]	L1	op pairs, probabilities, magnitudes	operation set, magnitude bins, child model
LLM-FE [2]	L1	evolved feature-engineering programs	tabular tasks, input schema, validation metric
Synthetic corpora and curricula			
Self-Instruct [261]	L1	new instructions, inputs and outputs	seed tasks, output format, heuristic filters
Automated curriculum [86]	L1	which task to train on next	task set, progress signals

Curriculum methods determine which tasks or environments supply subsequent training data. Automated curriculum learning treats task selection as a nonstationary bandit rewarded by learning progress [86]. POET, PAIRED, and LLM-POET generate or evolve environments together with agents [257, 49, 7]. Their generated environments serve as training artifacts, while their evaluation roles follow the component analysis in Section 6.5. Evaluation should test transfer beyond the construction curriculum.

Benchmarks for AI4AI Data. DataPerf, DataComp, and DataComp-LM compare data artifacts under standardized downstream tasks and training protocols [171, 77, 130]. DataPerf covers several data-centric tasks, DataComp targets image–text pretraining, and DataComp-LM uses a standardized Common Crawl pool for language-model pretraining. Data-Juicer supplies processing infrastructure for such experiments [29]. Reports should include coverage, duplication, contamination, privacy, licensing, and fairness, and confirm proxy-scale rankings at larger model and data scales.

Synthetic-data evaluation adds complementary evidence. AgoraBench compares language models as data generators through standardized student-training outcomes, while DataEnvGym provides teacher environments in which generation policies adapt to student errors and skill feedback across training iterations [116, 112]. The former isolates generation utility; the latter tests a feedback-driven data-construction process.

Curation-Bench holds the model, training recipe, and evaluator fixed while agents inspect data and iteratively implement selection policies; its vision–language instruction-tuning experiments distinguish successful execution from effective research exploration [111]. RSIBench-Data evaluates data-centric post-training research through capability-gap diagnosis, data strategies, and checkpoint feedback using shared training and evaluation services [174]. Because it also permits changes to specified training settings, reports should identify which gains arise from data and which involve learning-procedure changes. Training a student to assess data value does not by itself change the primary domain from Data to Learning.

7.4. AI4AI Infrastructure

AI4AI Infrastructure improves the software and hardware stack used to train and serve AI: algorithms, kernels, tensor programs, compiler schedules, parallel execution plans, accelerators, RTL, and physical design. Artifacts are evaluated through execution, compilation, simulation, or hardware measurement. General-purpose logic and EDA methods provide complementary techniques for this stack; Table 9 identifies their application scope.

7.4.1. Kernels and Tensor Programs

AlphaTensor searches matrix-multiplication algorithms, and AlphaDev searches assembly implementations of sorting routines [67, 170]. AutoTVM, Ansor, and the Halide autoscheduler optimize tensor-program schedules with learned cost models [34, 323, 3]. LLM-driven kernel evolution extends the target to CUDA source and dispatch logic [89]. AlphaEvolve applies program evolution to a matrix-multiplication tiling heuristic used in Gemini training and an arithmetic-circuit simplification for a TPU [190].

Evolutionary program systems provide reusable optimization machinery for these applications. FunSearch, OpenEvolve, ShinkaEvolve, and CodeEvolve maintain archives and generate edits from successful lineages [214, 219, 123, 12]. Algorithm Evolution and Evolution of Heuristics combine algorithmic ideas with implementations; Evolution through Large Models, Language Model Crossover, and LLM evolutionary optimizers or surrogates supply proposal and variation mechanisms [143, 142, 127, 177, 150, 91]. SBLLM retrieves code-optimization patterns, while SeaEvo, graph-augmented evolution, and adaptive population handoff organize strategy archives [81, 162, 37, 161]. Their infrastructure applications are assessed on the executable algorithm or implementation produced. EvoTrace and evolutionary-search ablations distinguish structural changes from constant tuning, rediscovery, and additional sampling [198, 315].

Kernel comparisons should cover tiling, warp mapping, memory layout, fusion, and dispatch under a common correctness and timing harness. Shapes, dtypes, strides, masks, and numerical tolerances belong in correctness tests. Search and compiler costs should accompany steady-state speedups.

7.4.2. Execution Plans and Hardware Co-Design

FlexFlow searches distributed-training parallelization strategies with an execution simulator [108]. Apollo optimizes accelerator parameters under area constraints, and NAAS jointly searches accelerator architecture, compiler mapping, and neural architecture [110, 138]. ArchGym supplies common interfaces between searchers and architecture simulators [120]. Co-design studies should report fixed-compiler and retuned-compiler comparisons, multiple devices, search cost, and quality–latency–energy–area trade-offs.

7.4.3. Logic, RTL, and Physical Design

DRiLLS and PrefixRL search synthesis sequences and parallel-prefix circuits; AlphaChip, AutoDMP, and MaskPlace optimize macro placement or placement parameters [98, 215, 179, 5, 121]. AlphaChip and PrefixRL have been applied to production TPU and GPU blocks, respectively. AutoChip, ChatEDA, and MCP4EDA connect language-model generation to compilation, simulation, or RTL-to-GDSII feedback [243, 278, 260]. VeriGen and RTLCoder supply learned HDL generation, GPT4AIGChip targets AI accelerator generation, and DeepGate supplies circuit representations for the surrounding design process [242, 149, 75, 135]. Evaluation should separate generation, repair, and tool orchestration and validate final power, performance, and area (PPA) with independent verification.

Table 9: Representative AI4AI Infrastructure methods, selected to illustrate distinct software and hardware artifacts.

Method	Level	AI-designed	Human-fixed
Algorithms, kernels, and tensor programs			
AlphaTensor [67]	L1	matrix-multiplication algorithms	tensor formulation, action space, objective
Ansor [323]	L1	sketch annotations: tiles, unrolling	sketch rules, schedule space, tests
AlphaEvolve [190]	L1	matmul tiling heuristic; TPU circuit	candidate pool, diff encoder, evaluation harness
Execution plans and hardware co-design			
FlexFlow [108]	L1	per-operator parallelization strategy	parallelization space, simulator, hardware model
NAAS [138]	L1	accelerator, mapping, network	architecture grammar, mapping space, simulator
Logic, RTL, and physical design			
DRiLLS§ [98]	L1	sequence of synthesis transformations	transformation set, cell library, area and delay
AlphaChip [179]	L1	macro positions on the canvas	placement space, netlist, power and congestion
AutoChip§ [243]	L1	Verilog modules and their repairs	RTL template, testbench, PPA objective

§ General-purpose EDA application.

Benchmarks for AI4AI Infrastructure. KernelBench evaluates GPU kernels through correctness and speedup over PyTorch reference workloads [193]. CompilerGym and ArchGym provide compiler and architecture search environments with runtime, code-size, or simulator costs [46, 120]. VerilogEval and RTLLM evaluate HDL correctness and design quality, ChipBench covers hierarchical Verilog generation, and ChiPBench evaluates placement through a full physical-design flow [147, 159, 301, 263]. RE-Bench includes a kernel-optimization research task [275]. Reports should include compile or synthesis success, latency distributions, memory traffic, energy, verification coverage, PPA, tool runtime, and transfer across shapes, devices, designs, and technology libraries.

FastKernels checks whether kernel-level gains persist inside production-model execution paths, while Φ-Bench extends evaluation to repository-level implementation and end-to-end optimization of training and inference infrastructure [191, 50]. KernelGenBench evaluates generated Triton kernels across operator sources and hardware platforms, adding controlled evidence about deployment transfer and its cost [307]. MetaKernelBench instead transfers a natural-language skill distilled from one optimization attempt to a fresh attempt at the same problem in another domain-specific language [38]. This diagnostic remains within Infrastructure: it measures same-problem, cross-language knowledge reuse, which should be distinguished from transfer to new tasks or hardware.

7.5. Research Workflows and Shared Evaluation

Research workflows connect the application families. ResearchAgent proposes literature-grounded ideas; The AI Scientist, The AI Scientist-v2, and Agent Laboratory connect ideation or a supplied goal to experiments and reporting [14, 158, 292, 217]. MLE-STAR refines machine-learning pipelines through search and ablations, AutoResearchClaw maintains persistent research evidence, and Jr. AI Scientist extends a baseline paper [184, 146, 181]. Their application is the development workflow: an investigation can modify a training recipe, data policy, inference scaffold, or implementation. The direction-selection components in Section 4.5 support these changes, and automated reviewers supply complementary evaluation components.

Data Interpreter, MetaGPT, AutoGen, AgentVerse, and AgentScope provide planning and coordination;

AgentCoder, SWE-agent, and Agentless provide code-development and repair mechanisms [96, 97, 279, 35, 78, 101, 294, 282]. MemoryBank, MemGPT, ExpeL, and Generative Agents offer memory and experience-management techniques that can support artifact improvement [327, 195, 320, 196]. Multi-agent aggregation also provides a compute baseline for coordinated workflows [131]. Workflow comparisons should track experimental validity, reproducibility, implementation failures, and discovery cost alongside artifact quality [17, 333, 163, 74, 285].

AIRS-Bench evaluates idea generation, implementation, experiment analysis, and iterative refinement on paper-derived research tasks without supplying baseline code [165]. RSIGym supplies shared research services with Data, Harness, and Joint tracks, allowing component-specific studies and joint optimization of data, training settings, and execution harnesses [175]. These are complementary research protocols across the four application families. Their domain coverage follows the actual task and permitted intervention; an extensible environment does not establish coverage of every family, and each track's fixed components should be reported separately.

Evaluation components serve several applications. ChatUniTest and CodeJudge supply generated tests and semantic code judgments; MT-Bench judging, G-Eval, JudgeLM, Prometheus, and Prometheus 2 provide reusable model judges [39, 248, 322, 152, 332, 114, 115]. Their use across training, data filtering, and agent development follows the integration analysis in Section 6. Red teaming and AutoRISE supply adaptive adversarial tests; debate, scalable oversight, CriticGPT, and weak-to-strong supervision combine AI-produced evidence with human oversight [200, 82, 105, 21, 172, 22]. Independent checks should test evaluator agreement, calibration, model and style biases, and resistance to optimization pressure [322, 255, 297, 265, 80, 44, 59].

Benchmarks for research diagnostics. WhatWorkedBench measures experimental understanding by asking agents to acquire budgeted measurements and reconstruct configuration responses, conditional component effects, and interactions [189]. It complements the direction-selection analysis in Section 4.5 by assessing what agents learn from experiments beyond their best observed score. BenchBench evaluates the validity, discrimination, and interaction biases of automatically generated test suites [325]. EvalResearchBench evaluates budgeted task selection or synthesis, grader implementation, and pilot-driven revision, followed by independent assessment of the frozen evaluator's measurement quality and execution cost [318]. These evaluation-design benchmarks provide evidence relevant to E. Under Section 6, an L3 assignment additionally requires that the AI-produced signal becomes operative in a specified improvement loop; independent outer scoring can remain human-defined.

Comparable application evidence. Across the four families, evaluation should report artifact quality, total discovery cost, transfer, and variation across runs. Table 10 links each benchmark to an AI4AI development task, the protocol used to assess it, and the evidence it provides. Some protocols run the optimizer or research agent, while others validate the resulting architecture, dataset, agent, or implementation through shared downstream tests. Search and deployment costs should be separated, including failed trials. MLReplicate adds reproducibility checks, while SciIntegrity-Bench tests research integrity through dilemmas that place honest acknowledgment of failure in conflict with task completion [76, 296]. Together with the application-specific measures above, these protocols connect the direction, space, and evaluation mechanisms reviewed earlier to observable benefits in AI development.

System domains and assessment roles should remain separate from method-level responsibility profiles.

A fixed task permits history-conditioned direction selection, while evidence for AI-provided search space requires tracing changes to active representations or constraints rather than candidate code alone. Repeated research runs should test whether benefits survive variation in experimental trajectories. Recursive progress requires an additional comparison: accepted changes must improve how subsequent improvements are produced. RSIGym's current experiments do not establish this subsequent-cycle benefit [175]. Thus, improved artifacts, reusable research knowledge, and improved future discovery require distinct evidence.

Table 10: Representative benchmarks for AI4AI development. Domain groups indicate primary targets; the cross-family group covers research workflows, experimental understanding, and evaluator design.

Benchmark	AI4AI task	Evaluation protocol	Evidence provided
AI4AI Learning			
NAS-Bench-201 [54]	Select neural architectures for training	Query tabulated outcomes under a shared cell space and training recipe	Accuracy or regret versus search cost
HPOBench [58]	Tune hyperparameters and allocate trial budgets	Compare optimizers on common datasets, spaces, and fidelities	Anytime performance and cost to reach target quality
AI4AI-Bench [40]	Redesign training algorithms	Rerun edited code from scratch against the original algorithm; hidden evaluator	Capability gain from the revised learning procedure
AI4AI Inference			
AutoHarnessBench [228]	Improve a fixed actor's execution harness	Separate proposer and actor; freeze the selected harness after public trials; assess held-out lift	Generalization of harness improvements and research cost
τ^{τ}-Bench [221]	Build a deployable task agent	Construct from client requirements, business records, APIs, and inherited code; test with held-out simulated users	Delivery success under serving constraints
AI4AI Data			
DataComp [77]	Select or filter multimodal training data	Train CLIP with a shared recipe and compute scale	Downstream gains from the submitted dataset
DataEnvGym [112]	Construct training data from student feedback	Iterate generation, student training, and evaluation in shared teacher environments	Student gains across rounds and use of feedback
Curation-Bench [111]	Discover data-selection policies	Fix the model, training recipe, and evaluation; iterate executable data policies	Data efficiency and research behavior beyond execution
AI4AI Infrastructure			
KernelBench [193]	Generate and refine ML GPU kernels	Execute against PyTorch references on specified hardware	Fraction correct and faster than a speedup threshold
Φ-Bench [50]	Engineer training and inference infrastructure	Implement and optimize real-repository tasks; gate rewards on correctness and anti-hacking checks	Repository-level functionality and performance
ChiPBench [263]	Optimize chip-placement methods	Run placements through a shared downstream physical-design flow	Final routed PPA beyond intermediate placement proxies
Cross-family AI research and evaluation			
RE-Bench [275]	Automate ML research engineering	Compare agents and human experts under matched time budgets	Task improvement versus time and monetary cost
PaperBench [233]	Reproduce AI research methods and experiments	Execute submitted code in a fresh environment; grade with paper-specific rubrics	Weighted replication score over implementation and result checks
WhatWorkedBench [189]	Understand effects of experimental changes	Buy limited configuration measurements; predict effects and interactions against exhaustive references	Effect recovery, configuration choice, and experimental understanding
EvalResearchBench [318]	Construct reliable executable evaluators	Select or synthesize tasks, implement graders, and pilot within budgets; freeze for independent assessment	Score/rank agreement and execution cost

8. Open Questions and Future Works

The preceding sections organized AI4AI by the source of control over direction, search space, and evaluation. This section addresses two complementary questions: what remains unresolved about AI-controlled discovery, and what evidence and research methods would be needed to address these gaps.

8.1. Open Questions

When is research control worth delegating? AI-provided direction, search space, or evaluation does not necessarily improve research quality [249, 163]. The value of delegation may depend on how quickly errors become observable, whether their consequences are reversible, how costly independent verification is, and how much human effort the decision would otherwise require [21]. These properties need not favor the same responsibility profile across domains or stages of a project. Control variables may also interact: a constructed space may be productive only under a compatible direction policy, while that policy may depend on feedback that an existing evaluator cannot provide. Which task properties predict these dependencies, and when does joint delegation outperform selective delegation?

Answering this question requires separating decision provenance, causal influence, and beneficial contribution. An AI-generated decision may change the search trajectory without improving it; observed gains may also reflect mechanisms other than the one a system claims to introduce [133, 198]. A small intervention may instead unlock discoveries that would otherwise remain inaccessible. Can these effects be identified through controlled interventions rather than inferred from system descriptions? More broadly, should research autonomy be a fixed configuration or a revisable allocation of authority that changes with evidence, uncertainty, and failure risk?

Can AI expand the boundaries of practical discoverability? A sufficiently expressive language may already represent a desirable algorithm without making it discoverable within a realistic budget. New abstractions, primitives, and composition rules can shorten useful descriptions, alter search neighborhoods, or eliminate unproductive combinations without necessarily enlarging the set of theoretically expressible solutions [61, 85, 270]. The central question for search-space construction is therefore not simply whether AI can propose more candidates, but whether it can make previously inaccessible, high-quality solutions practically reachable. When does a learned representation expand this frontier, and when does it merely encode a narrow preference for familiar solutions?

The same question extends to the organization of discovery. Recent systems illustrate both progress and remaining constraints. AIDE^2 recursively improves research-agent mechanisms while retaining a fixed benchmark-based acceptance rule [232]. The Agentic Scientist supports AI-generated task interfaces and evaluators, but its main empirical studies rely on manually prepared tasks, and reliable task construction remains challenging in complex settings [122]. At a given unit of analysis, AI may provide D, S, and E while higher-level rules for maintaining hypotheses, decomposing experiments, accumulating evidence, and coordinating research programs remain human-designed. Can AI identify limitations in this organization and improve the research workflow itself? Such changes can be analyzed by shifting the unit of analysis rather than introducing another autonomy level. Whether improved workflows transfer beyond the settings in which they were discovered remains a separate question from whether they improve the current run.

What establishes progress when evaluation itself evolves? When a candidate system and its evaluator change together, increasing internal scores does not isolate capability improvement from changes in the measurement procedure. Even a fixed proxy can become unreliable under stronger optimization [80, 59],

while evolving model-based judgments introduce further risks of changing standards and shared biases [280, 265]. A revised evaluator may detect previously overlooked failures, but it may also reward easier tasks, favor the generator's characteristic outputs, or relax acceptance conditions. These effects can influence not only the current ranking but also which evidence becomes available to later discovery cycles. How can a system distinguish better measurement of an intended objective from an unintended change in what it is optimizing?

Reliable assessment requires evidence that is not reducible to the improving loop's own judgments. Such grounding need not consist exclusively of human ratings: it may include executable checks, independent experiments, hidden tests, or delayed observations [188, 137, 21]. However, each anchor has a limited scope. Formal correctness does not establish scientific importance, and agreement with a simulator does not establish validity in the target environment. What combination of independent evidence is sufficient for a particular discovery claim? How does the required verification budget scale with optimization pressure, discovery horizon, and shared errors between generator and evaluator? Any assurance must remain conditional on the validity and coverage of these anchors.

When do local improvements become cumulative research knowledge? An artifact can be useful without revealing why it works, when its advantages persist, or how it should inform subsequent designs. The stronger question is whether AI4AI can convert experimental outcomes into knowledge that supports new predictions and discoveries. Prior work on learned program libraries and experiential learning provides starting points for reusable abstractions and accumulated knowledge [61, 85, 320]. Such knowledge might take the form of executable abstractions, experimentally supported design rules, or validated negative results with explicit applicability conditions. It need not be immediately reducible to a concise human-readable explanation, nor must every constituent be unprecedented. Its research value depends on whether others can test, reuse, and revise it.

What representations allow successful discoveries to remain reusable without turning contingent observations into rigid assumptions? Approaches based on persistent skill libraries show one way to carry experience forward [253, 329], but the ability to preserve conflicting evidence and recognize misleading abstractions remains an open question. Most importantly, how can the contribution of accumulated knowledge be distinguished from access to more compute, longer histories, or stored solutions? A convincing account of cumulative discovery should explain how inherited knowledge changes the productivity of research on genuinely new problems, not merely how it reproduces previous successes.

When does recursive self-improvement become sustained and reinforcing? Successive improvements to an artifact do not establish that the mechanism producing those improvements has become more capable. Likewise, repeated self-modification does not, by itself, establish sustained gains in research productivity [310, 312]. A claim of *recursive benefit* requires evidence that an earlier improvement strengthens the production of later improvements. Recent tests of self-improving research agents illustrate both the relevance and difficulty of measuring this effect independently [232]. Under what conditions does this feedback remain beneficial across generations, and when does it saturate because verification, experimentation, or accessible novelty becomes the limiting factor? Sustained improvement and accelerating improvement are also distinct claims; neither should be inferred from generation count alone.

The relevant feedback may extend beyond a single self-editing agent. AI-assisted methods have already produced advances in algorithm discovery, data optimization, and computational infrastructure [190, 286, 34]; these advances do not, by themselves, establish recursive benefits across these areas. Can improvements in inference, learning, data, and infrastructure reinforce one another after accounting for integration costs, verification overhead, and resource constraints? Conversely, which dependencies prevent local gains

from propagating through the wider AI development process? These questions concern the dynamics of improvement rather than its control label: an externally evaluated loop may exhibit recursive gains, while an AI-evaluated loop may fail to do so.

8.2. Future Work

Causal evaluation and adaptive allocation of research control. Future evaluation environments should expose D, S, and E as inspectable, versioned components that can be replaced independently or jointly where their interfaces permit meaningful comparisons. Reproducible search benchmarks and trajectory-level analyses offer useful starting points, but need to be extended to comparisons of control-source changes [298, 275, 198]. Runs should share initial targets and resource constraints while recording human effort, candidate histories, rejected directions, space revisions, evaluator changes, and rescue interventions. Comparisons should distinguish whether a component changes the discovery trajectory from whether that change improves independently assessed outcomes. Joint interventions are needed to reveal interactions that single-component ablations may miss.

These environments could support adaptive delegation policies that allocate authority according to uncertainty, decision consequences, and verification cost, extending concerns already raised in work on automated learning and scalable oversight [249, 21]. Such policies should also specify when to seek external approval or restore an earlier configuration. Their objective would be reliable research productivity under resource and risk constraints, rather than maximizing the number of AI-controlled variables. This would make evolving responsibility profiles experimentally testable choices about how research should be organized.

Open-ended search-space construction and meta-discovery. Research systems should be able to construct and revise candidate representations, reusable primitives, composition operators, and task-specific admissibility constraints within explicit external boundaries. Program abstraction learning and dynamic search-space methods provide partial precedents [61, 85, 295]. Suitable tasks would require the system to diagnose why an existing representation obstructs progress and then propose a space-defining artifact that supports subsequent discovery. Both expansion and compression should be permitted: a smaller space may be more productive if it removes irrelevant combinations while preserving useful alternatives.

Evaluation should include the cost of constructing the space and test its utility on unseen tasks with multiple compatible search procedures. This can help separate reusable space quality from adaptation between one representation and one search policy. A complementary direction is to make experimental decomposition, hypothesis management, and evidence-update protocols editable research objects, building on broader work on automated agent and workflow design [100, 313]. Improvements to these mechanisms should be tested through independent replay and transfer, while execution permissions, resource limits, and non-negotiable validity requirements remain outside the system's unilateral control.

Independently grounded evaluator co-evolution. A promising experimental design separates an adaptive internal evaluator from an independent validation path. Research on AI-generated rewards and meta-judges illustrates mechanisms for constructing or refining internal evaluation signals [289, 280]. The internal evaluator may generate tests, revise judgment mappings, or improve feedback as new failures emerge. The validation path should assess whether those revisions remain faithful to the intended objective and should not be rewritable by the improving system during the same run. Independence should be investigated through error analysis and interventions rather than assumed from different model names, given documented biases in model-based judgments [255, 265].

Experiments should vary search intensity, discovery horizon, and evaluator flexibility while measuring

false acceptance, false rejection, and divergence between internal scores and external outcomes wherever independently assessed outcomes are available. Evaluating optimized high-scoring candidates is essential because static evaluator quality may not survive optimization pressure [80, 59]; rejected candidates should also be sampled to detect discoveries that a biased evaluator excludes. Longer studies could use pre-specified audit updates and cross-version calibration, with fresh audit data unavailable to the optimizing loop. The resulting evidence could characterize how verification requirements scale, rather than treating evaluator accuracy on a fixed candidate distribution as sufficient.

Knowledge-producing research memory. Research memory should preserve more than successful candidates and their scores. Existing systems have demonstrated the utility of retaining experiences, skills, and reusable program abstractions [320, 253, 85]. Structured records can connect hypotheses, implementation versions, experimental conditions, failures, applicability boundaries, and replication results. Proposed abstractions should retain links to supporting and conflicting evidence so that later systems can inspect their provenance and revise them when conditions change.

Knowledge extraction should be evaluated through its downstream consequences. One system could infer a design rule from historical experiments, while another independently implements it and tests predictions on unseen tasks. Ablations should distinguish access to original artifacts, raw histories, textual summaries, and reusable abstractions under comparable budgets. Shared repositories should also support negative and retracted findings, allowing accumulation to reduce redundant exploration rather than merely enlarge the archive. The central outcome is whether inherited knowledge enables new discoveries more effectively, not whether the system produces persuasive retrospective explanations.

Longitudinal and cross-family recursive-improvement experiments. Multi-generation studies should evaluate both the current artifact and the system's ability to produce subsequent improvements. Initial work on self-improving agents and benchmarks for AI algorithm design motivates this distinction [310, 232, 40]. Each generation should be tested on common starting targets and held-out research tasks under matched budgets. Relevant controls include a frozen initial improver, a system that accumulates history without modifying its research mechanism, and alternatives given equivalent additional computation. Selective rollback and interrupted inheritance can test whether later gains causally depend on earlier discoveries.

Integrated environments could further allow improvements to propagate across Learning, Inference, Data, and Infrastructure. Existing work demonstrates optimization within several of these areas, but not necessarily a closed, mutually reinforcing cross-family loop [190, 286, 34]. Experiments should separately report research capability at fixed resources, gains from reinvesting saved resources, and the total cost of validation and integration. Blocking individual feedback paths would reveal whether cross-family coupling adds value beyond isolated improvements. Measuring progress against both resource use and elapsed time would help distinguish persistent productivity gains from apparent acceleration caused by changing budgets or shorter discovery cycles.

AI4AI4Science: discovering AI methods for scientific discovery. AI4AI4Science extends this agenda to the AI-driven discovery of methods that enable scientific progress. Work on algorithmic discovery and automated scientific research provides relevant foundations [214, 190, 158], but AI4AI4Science specifically targets improvements to the AI methods used in scientific inquiry. These methods may include representations, architectures, learning rules, data strategies, inference procedures, and computational implementations. Their ultimate value is assessed through the scientific investigations they make possible. The defining intervention is a substantive AI-originated change to a scientific AI method, rather than simply executing a fixed AI-assisted research workflow.

Future environments could couple scientific bottleneck diagnosis with AI-method construction. In scientific computing, a system might respond to numerical or long-horizon prediction failures by proposing new solver representations, training objectives, or computational implementations; automated design of neural PDE solvers offers a relevant starting point [119]. In molecular or materials research, it might revise generative methods, uncertainty estimation, or experimental acquisition strategies when evidence exposes limitations in existing approaches. These examples are proposed research settings, not evidence that reliable end-to-end AI4AI4Science loops have already been established.

Evaluation should establish both methodological and scientific value. First, the discovered AI method should be compared with appropriate baselines under matched data, compute, and experimental conditions. Second, its gains should translate into independently supported scientific outcomes, more reliable predictions, or lower costs of obtaining useful evidence. Improvements against a surrogate model or simulator must not be equated with scientific validation. Studies of benchmark and metric failures in automated research, alongside reward overoptimization, illustrate why internal or proxy improvements require careful interpretation [163, 80]. Stronger evidence may require higher-fidelity computation, independent measurements, or physical experiments. The long-term research objective is a reciprocal process in which scientific problems motivate new AI methods, and independently validated method improvements expand the range of scientific questions that can be investigated.

9. Conclusion and Limitations

AI4AI research can be analyzed through changing sources of control over the discovery loop: who specifies how to explore, where to explore, and what counts as progress. This survey formalizes these differences through a Direction–Space–Evaluation ladder. Under the proposed taxonomy, the highest-ranked AI-provided component determines a system's level. L0 serves as the reference level, with direction, space, and evaluation specified by humans. L1 denotes AI-provided direction while space and evaluation remain human-authored. L2 denotes AI-provided search space with human-authored evaluation, regardless of who supplies direction. L3 denotes AI-provided evaluation regardless of who supplies direction or space.

Higher levels are not guaranteed to produce better systems. At L1, promising directions may not translate into successful experiments [333], and learned exploration strategies may fail to transfer across tasks, scales, or optimization horizons [274, 176]. Learning from execution can also improve average outcomes without improving the best discoveries. At L2, richer representations do not necessarily yield more usable search spaces. Learned abstractions may be difficult to compose or reuse [85], while observed gains may come from additional sampling rather than the constructed space itself [133]. At L3, evaluation errors can become selection pressures. Signals that perform well on fixed candidates may become unreliable under sustained optimization [80, 59], and more faithful evaluation can consume resources otherwise available for discovery [273, 134].

Across levels, end-to-end improvements do not by themselves identify which control mechanism contributed to progress [198, 315]. Comparisons must separate changes in research control from differences in underlying models, experimental infrastructure, and computational investment [133, 275]. Jointly adapting control mechanisms introduces a further challenge: stronger task performance does not establish a stronger capacity for subsequent self-improvement [310, 40]. Overall, the proposed taxonomy provides a systematic basis for comparing AI4AI systems, while distinguishing the allocation of research control from evidence of its effectiveness.

References

[1] Amro Abbas, Kushal Tirumala, Daniel Simig, Surya Ganguli, and Ari S. Morcos. SemDeDup: Data-efficient learning at web-scale through semantic deduplication. *arXiv preprint arXiv:2303.09540*, 2023.

[2] Nikhil Abhyankar, Parshin Shojaee, and Chandan K. Reddy. LLM-FE: Automated feature engineering for tabular data with llms as evolutionary optimizers. *arXiv preprint arXiv:2503.14434*, 2025.

[3] Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michael Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Fredo Durand, and Jonathan Ragan-Kelley. Learning to optimize halide with tree search and random programs. *ACM Transactions on Graphics*, 38(4):121:1–121:12, 2019.

[4] Eshaan Agarwal, Joykirat Singh, Vivek Dani, Raghav Magazine, Tanuja Ganu, and Akshay Nambi. PromptWizard: Task-aware prompt optimization framework. *arXiv preprint arXiv:2405.18369*, 2024.

[5] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. AutoDMP: Automated DREAMPlace-based macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*, pages 149–157, 2023.

[6] Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

[7] Fuma Aki, Riku Ikeda, Takumi Saito, Ciaran Regan, and Mizuki Oka. LLM-POET: Evolving complex environments using large language models. *arXiv preprint arXiv:2406.04663*, 2024.

[8] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019.

[9] Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. A survey on data selection for language models. *Transactions on Machine Learning Research*, 2024.

[10] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[11] Sebastian Pineda Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML, 2021.

[12] Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: An open source evolutionary coding agent for algorithmic discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025.

[13] Noor Awad, Neeratyoy Mallik, and Frank Hutter. DEHB: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 2147–2153, 2021.

[14] Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. ResearchAgent: Iterative research idea generation over scientific literature with large language models. *arXiv preprint arXiv:2404.07738*, 2024.

[15] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[16] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[17] Joeran Beel, Min-Yen Kan, and Moritz Baumgart. Evaluating sakana's AI scientist: Bold claims, mixed results, and a promising future? *arXiv preprint arXiv:2502.14297*, 2025.

[18] Loubna Ben Allal, Anton Lozhkov, and Daniel van Strien. Cosmopedia: How to create large-scale synthetic data for pre-training. Hugging Face Blog, March 2024.

[19] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[20] Matthew Bowers, Theo X. Olausson, Lionel Wong, Gabriel Grand, Joshua B. Tenenbaum, Kevin Ellis, and Armando Solar-Lezama. Top-down synthesis for library learning. *arXiv preprint arXiv:2211.16605*, 2022.

[21] Samuel R. Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Christopher Olah, Daniela Amodei, Dario Amodei, Dawn Drain, Dustin Li, Eli Tran-Johnson, Jackson Kernion, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Liane Lovitt, Nelson Elhage, Nicholas Schiefer, Nicholas Joseph, Noemí Mercado, Nova DasSarma, Robin Larson, Sam McCandlish, Sandipan Kundu, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Timothy Telleen-Lawton, Tom Brown, Tom Henighan, Tristan Hume, Yuntao Bai, Zac Hatfield-Dodds, Ben Mann, and Jared Kaplan. Measuring progress on scalable oversight for large language models. *arXiv preprint arXiv:2211.03540*, 2022.

[22] Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeff Wu. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

[23] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[24] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[25] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

[26] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, Tony Wang, Samuel Marks, Charbel-Raphaël Segerie, Micah Carroll, Andi Peng, Phillip Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J. Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Bıyık, Anca Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023.

[27] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

[28] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *International Conference on Learning Representations*, 2023.

[29] Daoyuan Chen, Yilun Huang, Zhijian Ma, Hesen Chen, Xuchen Pan, Ce Ge, Dawei Gao, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, Yaliang Li, Bolin Ding, and Jingren Zhou. Data-juicer: A one-stop data processing system for large language models. *arXiv preprint arXiv:2309.02033*, 2023.

[30] Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. AlpaGasus: Training a better alpaca with fewer data. In *International Conference on Learning Representations (ICLR)*, 2024.

[31] Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops. *arXiv preprint arXiv:2607.07663*, 2026.

[32] Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. Searching the search space of vision transformer. In *Advances in Neural Information Processing Systems*, volume 34, pages 8714–8726, 2021.

[33] Qiguang Chen, Mingda Yang, Libo Qin, Jinhao Liu, Zheng Yan, Jiannan Guan, Dengyun Peng, Yiyan Ji, Hanjing Li, Mengkang Hu, Yimeng Zhang, Yihao Liang, Yuhang Zhou, Jiaqi Wang, Zhi Chen, and Wanxiang Che. Ai4research: A survey of artificial intelligence for scientific research. *Science China Information Sciences*, 69(10):201101, 2026.

[34] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 3393–3404, Red Hook, NY, USA, 2018. Curran Associates Inc.

[35] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023.

[36] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*, 2023.

[37] Xuanzhou Chen and Taoli Cheng. GAE: Graph-augmented evolution for scientific discovery via reinforcement optimization. *arXiv preprint arXiv:2607.10127*, 2026.

[38] Xueyi Chen, Shiyu Liu, Xin Jin, Yuhua Zheng, Xin Li, Haolei Bai, Junhan Zhu, and Huan Wang. Metakernelbench: Measuring gpu kernel knowledge transfer beyond code, 2026.

[39] Yinghao Chen, Zehao Hu, Chen Zhi, Junxiao Han, Shuiguang Deng, and Jianwei Yin. ChatUniTest: A framework for LLM-based test generation. *arXiv preprint arXiv:2305.04764*, 2024.

[40] Yizhe Chi, Wenyi Li, Deyao Hong, Xiaoqiu Wang, Mingju Gao, Kaisen Yang, Bingxiang He, Youjie Zheng, Calvin Xiao, and Qinhuai Na. AI4AI-Bench: Benchmarking LLM agents in algorithmic design for recursive self-improvement. *arXiv preprint arXiv:2608.20318*, 2026.

[41] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[42] John D. Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine, Quoc V. Le, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021.

[43] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[44] Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. Reward model ensembles help mitigate overoptimization. *arXiv preprint arXiv:2310.02743*, 2023.

[45] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. AutoAugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[46] Chris Cummins, Bram Wasti, Jiadong Guo, Brandon Cui, Jason Ansel, Sahir Gomez, Somya Jain, Jia Liu, Olivier Teytaud, Benoit Steiner, Yuandong Tian, and Hugh Leather. CompilerGym: Robust, performant compiler optimization environments for AI research, 2021.

[47] Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In *International Conference on Learning Representations (ICLR)*, 2025.

[48] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

[49] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart J. Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[50] Leilei Ding, Shumin Wang, Yuting Huang, Fanqi Wan, Yinmin Zhang, Qi Han, Yiming Xu, Feiyuan Zhang, Xiaomeng Chu, Guoliang You, Wuyang Zhang, Daxin Jiang, and Yanyong Zhang. ϕ-bench: Can large language models engineer the infrastructure that powers them?, 2026.

[51] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*, 2023.

[52] Tianyu Ding, Aditya Nannapaneni, Bingfan Liu, and Ling Zhang. Autonomous research agents: A survey of AI scientists and the verification gap, 2026.

[53] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-Bench: Benchmarking NAS algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[54] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

[55] Shangheng Du, Jiabao Zhao, Jinxin Shi, Zhentao Xie, Xin Jiang, Yanhong Bai, and Liang He. A survey on the optimization of large language model-based agents. *ACM Computing Surveys*, 58(9):1–37, 2026.

[56] Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. TransNAS-Bench-101: Improving transferability and generalizability of cross-task neural architecture search. *arXiv preprint arXiv:2105.11871*, 2021.

[57] Manuel Eberhardinger, James Goodman, Alexander Dockhorn, Diego Perez-Liebana, Raluca D. Gaina, Duygu Çakmak, Setareh Maghsudi, and Simon Lucas. From code to play: Benchmarking program search for games using large language models. *arXiv preprint arXiv:2412.04057*, 2024.

[58] Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO, 2021.

[59] Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, Peter Shaw, and Jonathan Berant. Helping or herding? Reward model ensembles mitigate but do not eliminate reward hacking. *arXiv preprint arXiv:2312.09244*, 2023.

[60] Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Joshua B. Tenenbaum. Learning libraries of subroutines for neurally-guided bayesian program induction. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[61] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *arXiv preprint arXiv:2006.08381*, 2020.

[62] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[63] Talfan Evans, Nikhil Parthasarathy, Hamza Merzic, and Olivier J. Henaff. Data curation via joint example selection further accelerates multimodal learning. *arXiv preprint arXiv:2406.17711*, 2024.

[64] Roy Henha Eyono, Fabio Maria Carlucci, Pedro M. Esperança, Binxin Ru, and Philip H. S. Torr. AutoKD: Automatic knowledge distillation into a student architecture family. *arXiv preprint arXiv:2111.03555*, 2021.

[65] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446, 2018.

[66] Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

[67] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 2022.

[68] K. J. Kevin Feng, David W. McDonald, and Amy X. Zhang. Levels of autonomy for AI agents. *arXiv preprint arXiv:2506.12469*, 2025.

[69] Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 13481–13544, 2024.

[70] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free AutoML via meta-learning. *Journal of Machine Learning Research*, 23(261):1–61, 2022.

[71] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, volume 28, pages 2962–2970, 2015.

[72] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135, 2017.

[73] Jingwen Fu, Zhen Liu, Yuhan Liu, He Zhang, and Nanning Zheng. Overcoming the weakest-link effect in LLM-driven program optimization via heterogeneous edit recombination. *arXiv preprint arXiv:2607.28947*, 2026.

[74] Xinghong Fu, Ziming Liu, and Max Tegmark. Do two AI scientists agree? *arXiv preprint arXiv:2504.02822*, 2025.

[75] Yonggan Fu, Yongan Zhang, Zhongzhi Yu, Sixu Li, Zhifan Ye, Chaojian Li, Cheng Wan, and Yingyan Celine Lin. GPT4AIGChip: Towards next-generation AI accelerator design automation via large language models. *arXiv preprint arXiv:2309.10730*, 2023.

[76] Sasi Kiran Gaddipati, Diyana Muhammed, Farhana Keya, Gollam Rabby, and Soren Auer. MLReplicate: Benchmarking autonomous research systems for machine learning reproducibility. *arXiv preprint arXiv:2605.16616*, 2026.

[77] Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, Eyal Orgad, Rahim Entezari, Giannis Daras, Sarah Pratt, Vivek Ramanujan, Yonatan Bitton, Kalyani Marathe, Stephen Mussmann, Richard Vencu, Mehdi Cherti, Ranjay Krishna, Pang Wei Koh, Olga Saukh, Alexander Ratner, Shuran Song, Hannaneh Hajishirzi, Ali Farhadi, Romain Beaumont, Sewoong Oh, Alex Dimakis, Jenia Jitsev, Yair Carmon, Vaishaal Shankar, and Ludwig Schmidt. DataComp: In search of the next generation of multimodal datasets. In *Advances in Neural Information Processing Systems*, volume 36, pages 27092–27112, 2023.

[78] Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, Liuyi Yao, Hongyi Peng, Zeyu Zhang, Lin Zhu, Chen Cheng, Hongzhu Shi, Yaliang Li, Bolin Ding, and Jingren Zhou. AgentScope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024.

[79] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026.

[80] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10835–10866, 2023.

[81] Shuzheng Gao, Cuiyun Gao, Wenchao Gu, and Michael Lyu. Search-based LLMs for code optimization. *arXiv preprint arXiv:2408.12159*, 2024.

[82] Tanmay Gautam, Alireza Bahramali, and Sandeep Atluri. AutoRISE: Agent-driven strategy evolution for red-teaming large language models. *arXiv preprint arXiv:2604.22871*, 2026.

[83] Cheng Gong, Ye Lu, Su-Rong Dai, Qian Deng, Cheng-Kun Du, and Tao Li. AutoQNN: An end-to-end framework for automatically quantizing neural networks. *Journal of Computer Science and Technology*, 39(2):401–420, 2024.

[84] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[85] Gabriel Grand, Lionel Wong, Maddy Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. LILO: Learning interpretable libraries by compressing and documenting code. *arXiv preprint arXiv:2310.19791*, 2023.

[86] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1311–1320, 2017.

[87] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on LLM-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

[88] Yang Gu, Hengyu You, Jian Cao, Muran Yu, Haoran Fan, and Shiyou Qian. Large language models for constructing and optimizing machine learning workflows: A survey. *ACM Transactions on Software Engineering and Methodology*, 35(9):1–46, 2026.

[89] Ping Guo, Chao Li, Yinglan Feng, and Chaoning Zhang. Code evolution for control: Synthesizing policies via LLM-driven evolutionary search, 2026.

[90] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. EvoPrompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

[91] Hao Hao, Xiaoqun Zhang, and Aimin Zhou. Large language models as surrogate models in evolutionary algorithms: A preliminary study. *arXiv preprint arXiv:2406.10675*, 2024.

[92] Han He, Qianchu Liu, Lei Xu, Chaitanya Shivade, Yi Zhang, Sundararajan Srinivasan, and Katrin Kirchhoff. CriSPO: Multi-aspect critique-suggestion-guided automatic prompt optimization for text generation. *arXiv preprint arXiv:2410.02748*, 2024.

[93] Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[94] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[95] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2731–2741, 2019.

[96] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Zhibin Gou, Zongze Xu, and Chenglin Wu. Data interpreter: An LLM agent for data science. *arXiv preprint arXiv:2402.18679*, 2024.

[97] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[98] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. DRiLLS: Deep reinforcement learning for logic synthesis. In *Asia and South Pacific Design Automation Conference*, pages 581–586, 2020.

[99] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5149–5169, 2022.

[100] Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *The Thirteenth International Conference on Learning Representations*, 2025.

[101] Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agent-Coder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.

[102] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2024.

[103] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

[104] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[105] Geoffrey Irving, Paul Christiano, and Dario Amodei. AI safety via debate. *arXiv preprint arXiv:1805.00899*, 2018.

[106] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[107] Yash Jain and Vishal Chowdhary. Local prompt optimization. *arXiv preprint arXiv:2504.20355*, 2025.

[108] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *Proceedings of Machine Learning and Systems*, volume 1, 2019.

[109] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024.

[110] Albin Jones, Amir Yazdanbakhsh, Berkin Akin, Christof Angermueller, James Pierce Laudon, Kevin Swersky, Milad Hashemi, Ravi Narayanaswami, Sat Chatterjee, and Yanqi Zhou. Apollo: Transferable architecture exploration. In *ML for Systems Workshop at NeurIPS 2020*, 2020.

[111] Feiyang Kang, Hanze Li, Adam Nguyen, Mahavir Dabas, Jiaqi W. Ma, Frederic Sala, Dawn Song, and Ruoxi Jia. Can generalist agents automate data curation?, 2026.

[112] Zaid Khan, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. Dataenvgym: Data generation agents in teacher environments with student feedback. In *The Thirteenth International Conference on Learning Representations*, 2025.

[113] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

[114] Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models. *arXiv preprint arXiv:2310.08491*, 2023.

[115] Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. *arXiv preprint arXiv:2405.01535*, 2024.

[116] Seungone Kim, Juyoung Suk, Xiang Yue, Vijay Viswanathan, Seongyun Lee, Yizhong Wang, Kiril Gashteovski, Carolin Lawrence, Sean Welleck, and Graham Neubig. Evaluating language models as synthetic data generators. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6385–6403, Vienna, Austria, July 2025. Association for Computational Linguistics.

[117] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894, 2017.

[118] Lingdong Kong, Xian Sun, Wei Chow, Linfeng Li, Kevin Qinghong Lin, Xuan Billy Zhang, Song Wang, Rong Li, Qing Wu, Wei Gao, Yingshuo Wang, Shaoyuan Xie, Jiachen Liu, Leigang Qu, Shijie Li, Lai Xing Ng, Benoit R. Cottereau, Ziwei Liu, Tat-Seng Chua, and Wei Tsang Ooi. AI for auto-research: Roadmap & user guide. *arXiv preprint arXiv:2605.18661*, 2026.

[119] Shengxin Kong, Liwen Xu, and Jingwen Fu. Improving auto-design of neural PDE solvers with a domain-specific language. *arXiv preprint arXiv:2608.04384*, 2026.

[120] Srivatsan Krishnan, Amir Yazdanbaksh, Shvetank Prakash, Jason Jabbour, Ikechukwu Uchendu, Susobhan Ghosh, Behzad Boroujerdian, Daniel Richins, Devashree Tripathy, Aleksandra Faust, and Vijay Janapa Reddi. ArchGym: An open-source gymnasium for machine learning assisted architecture design. *arXiv preprint arXiv:2306.08888*, 2023.

[121] Yao Lai, Yao Mu, and Ping Luo. MaskPlace: Fast chip placement via reinforced visual representation learning. *arXiv preprint arXiv:2211.13382*, 2022.

[122] Robert Tjarko Lange, Luca Grillotti, Yutaro Yamada, Rujikorn Charakorn, and David Ha. The agentic scientist: Orchestrating coding agents for automated scientific (co-)discovery. *arXiv preprint arXiv:2610.18933*, 2026.

[123] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

[124] Lauro Langosco, Jack Koch, Lee Sharkey, Jacob Pfau, and David Krueger. Goal misgeneralization in deep reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12004–12019, 2022.

[125] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[126] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. *arXiv preprint arXiv:2309.00267*, 2023.

[127] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022.

[128] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan Lin. HW-NAS-Bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584*, 2021.

[129] Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. From generation to judgment: Opportunities and challenges of LLM-as-a-judge. *arXiv preprint arXiv:2411.16594*, 2024.

[130] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models. *arXiv preprint arXiv:2406.11794*, 2024.

[131] Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *arXiv preprint arXiv:2402.05120*, 2024.

[132] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems*, volume 2, 2020.

[133] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of the 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 367–377. PMLR, July 2020.

[134] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[135] Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Yu Huang, and Qiang Xu. DeepGate: Learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 667–672, 2022.

[136] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M. Robertson, and Yongxin Yang. DADA: Differentiable automatic data augmentation. In *Proceedings of the European Conference on Computer Vision*, pages 580–595, 2020.

[137] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[138] Yujun Lin, Mengtian Yang, and Song Han. NAAS: Neural accelerator architecture search. In *Proceedings of the 58th Annual ACM/IEEE Design Automation Conference*, Dac '21, pages 1051–1056, San Francisco, California, United States, 2021. IEEE Press.

[139] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, Andre Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, Rene Sass, and Frank Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[140] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Fei-Fei Li, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Computer Vision – ECCV 2018*, volume 11205 of *Lecture Notes in Computer Science*, pages 19–35, 2018.

[141] Dancheng Liu, Amir Nassereldine, Ziming Yang, Chenhui Xu, Yuting Hu, Jiajie Li, Utkarsh Kumar, Changjae Lee, Ruiyang Qin, Yiyu Shi, and Jinjun Xiong. Large language models have intrinsic self-correction ability. *arXiv preprint arXiv:2406.15673*, 2024.

[142] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

[143] Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249*, 2023.

[144] Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Zhe Zhao, Xi Lin, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, Mingxuan Yuan, and Qingfu Zhang. A systematic survey on large language models for algorithm design. *arXiv preprint arXiv:2410.14716*, 2024.

[145] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[146] Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Meng Chen, Congyu Zhang, Letian Zhang, Guiming Chen, Haoqin Tu, Xinyu Yang, Lu Feng, Xujiang Zhao, Haifeng Chen, Jiawei Zhou, Xiao Wang, Weitong Zhang, Hongtu Zhu, Yun Li, Jieru Mei, Hongliang Fei, Jiaheng Zhang, Linjie Li, Linjun Zhang, Yuyin Zhou, Sheng Wang, Caiming Xiong, James Zou, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. AutoResearchClaw: Self-reinforcing autonomous research with human–AI collaboration. *arXiv preprint arXiv:2605.20025*, 2026.

[147] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. VerilogEval: Evaluating large language models for verilog code generation. In *Proceedings of the 2023 IEEE/ACM International Conference on Computer-Aided Design*, 2023.

[148] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(4):4876–4883, 2020.

[149] Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. RTLCoder: Fully open-source and efficient LLM-assisted RTL code generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(4):1448–1461, 2025.

[150] Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. *arXiv preprint arXiv:2310.19046*, 2023.

[151] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[152] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, 2023.

[153] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. MetaPruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019.

[154] Zhen Liu, Wenli Huang, Wei Song, Yuhan Liu, Zhiqin Yang, and Jingwen Fu. An AI4AI framework for visual token pruning. *arXiv preprint arXiv:2608.07193*, 2026.

[155] Zhen Liu, Yuhan Liu, Jinjun Wang, Wei Song, Jianyi Liu, and Jingwen Fu. Structured progressive knowledge activation for LLM-driven neural architecture search. *arXiv preprint arXiv:2605.04057*, 2026.

[156] Zhen Liu, Wanqi Zhou, Shuanghao Bai, Yuhan Liu, Jinjun Wang, and Jingwen Fu. GraphIR: Architecture-level search states for LLM-guided neural architecture evolution. *arXiv preprint arXiv:2608.01633*, 2026.

[157] Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. On LLMs-driven synthetic data generation, curation, and evaluation: A survey. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11065–11082, 2024.

[158] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

[159] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. RTLLM: An open-source benchmark for design RTL generation with large language model. *arXiv preprint arXiv:2308.05345*, 2023.

[160] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *arXiv preprint arXiv:1808.07233*, 2018.

[161] Sichun Luo, Yi Huang, Guanzhi Deng, Haibo Wang, Haochen Luo, Lei Li, Zefa Hu, Junlan Feng, and Qi Liu. Relay, don't route: Adaptive population handoff for cost-efficient LLM-driven evolution. *arXiv preprint arXiv:2608.05651*, 2026.

[162] Sichun Luo, Yi Huang, Haochen Luo, Fengyuan Liu, Guanzhi Deng, Lei Li, Qinghua Yao, Zefa Hu, Junlan Feng, and Qi Liu. SeaEvo: Advancing algorithm discovery with strategy space evolution. *arXiv preprint arXiv:2604.24372*, 2026.

[163] Ziming Luo, Atoosa Kasirzadeh, and Nihar B. Shah. The more you automate, the less you see: Hidden pitfalls of AI scientist systems. *arXiv preprint arXiv:2509.08713*, 2025.

[164] Ziming Luo, Zonglin Yang, Zexin Xu, Wei Yang, and Xinya Du. LLM4SR: A survey on large language models for scientific research. *arXiv preprint arXiv:2501.04306*, 2025.

[165] Alisia Lupidi, Bhavul Gauri, Thomas Simon Foster, Bassel Al Omari, Despoina Magka, Alberto Pepe, Alexis Audran-Reiss, Muna Aghamelu, Nicolas Baldwin, Lucia Cipolina-Kun, Jean-Christophe Gagnon-Audet, Chee Hau Leow, Sandra Lefdal, Hossam Mossalam, Abhinav Moudgil, Saba Nazir, Emanuel Tewolde, Isabel Urrego, Jordi Armengol Estape, Amar Budhiraja, Gaurav Chaurasia, Abhishek Charnalia, Derek Dunfield, Karen Hambardzumyan, Daniel Izcovich, Martin Josifoski, Ishita Mediratta, Kelvin Niu, Parth Pathak, Michael Shvartsman, Edan Toledo, Anton Protopopov, Roberta Raileanu, Alexander Miller, Tatiana Shavrina, Jakob Foerster, and Yoram Bachrach. Airs-bench: a suite of tasks for frontier ai research science agents, 2026.

[166] Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, Kaicheng Yang, Zitao Chen, Wentao Guo, Junlin Yang, Xinyue Ai, Wenhao Chai, Yadi Cao, Ziran Yang, Kun Wang, Dapeng Jiang, Huan ang Gao, Shange Tang, Chengshuai Shi, Simon S. Du, Max Simchowitz, Jiantao Jiao, Dawn Song, and Chi Jin. Mls-bench: A holistic and rigorous assessment of ai systems on building better ai, 2026.

[167] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. *arXiv preprint arXiv:2401.13178*, 2024.

[168] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[169] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[170] Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

[171] Mark Mazumder, Colby Banbury, Xiaozhe Yao, Bojan Karlaš, William Gaviria Rojas, Sudnya Diamos, Greg Diamos, Lynn He, Alicia Parrish, Hannah Rose Kirk, Jessica Quaye, Charvi Rastogi, Douwe Kiela, David Jurado, David Kanter, Rafael Mosquera, Will Cukierski, Juan Ciro, Lora Aroyo, Bilge Acun, Lingjiao Chen, Mehul Raje, Max Bartolo, Evan Sabri Eyuboglu, Amirata Ghorbani, Emmett Goodman, Addison Howard, Oana Inel, Tariq Kane, Christine R. Kirkpatrick, D. Sculley, Tzu-Sheng Kuo, Jonas W Mueller, Tristan Thrush, Joaquin Vanschoren, Margaret Warren, Adina Williams, Serena Yeung, Newsha Ardalani, Praveen Paritosh, Ce Zhang, James Zou, Carole-Jean Wu, Cody Coleman, Andrew Ng, Peter Mattson, and Vijay Janapa Reddi. DataPerf: Benchmarks for data-centric AI development. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 36, pages 5320–5347, 2023.

[172] Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. LLM critics help catch LLM bugs. *arXiv preprint arXiv:2407.00215*, 2024.

[173] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training. *arXiv preprint arXiv:2006.04647*, 2020.

[174] Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. Rsibench-data: Benchmarking data-centric research for recursive self-improvement, 2026.

[175] Fanqing Meng, Lingxiao Du, Haocheng Lu, Qiguang Chen, Ziqi Zhao, Zijian Wu, Jiayuan Zhuo, Mengkang Hu, and Michael Qizhe Shieh. Rsigym: A flexible environment for recursive self-improvement, 2026.

[176] Luke Metz, James Harrison, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, and Jascha Sohl-Dickstein. VeLO: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

[177] Elliot Meyerson, Mark J. Nelson, Herbie Bradley, Adam Gaier, Arash Moradi, Amy K. Hoover, and Joel Lehman. Language model crossover: Variation through few-shot prompting. *arXiv preprint arXiv:2302.12170*, 2023.

[178] Gregoire Mialon, Clementine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.

[179] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594:207–212, 2021.

[180] Abhishek Mishra, Selvam Palanimalai, Yogendra Manawat, Samuel Verboomen, Prannay Hebbar, Damir Vrabac, Deepak Nathani, Sumeet Ramesh Motwani, Kunal Bhatia, and Vignesh Baskaran. AIE-Bench: Benchmarking agents that build agents. 1st Workshop on Combining Theory and Benchmarks (CTB), ICML 2026, 2026.

[181] Atsuyuki Miyai, Mashiro Toyooka, Takashi Otonari, Zaiying Zhao, and Kiyoharu Aizawa. Jr. AI scientist and its risk report: Autonomous scientific exploration from a baseline paper. *arXiv preprint arXiv:2511.04583*, 2025.

[182] Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksandra Faust, Clement Farabet, and Shane Legg. Position: Levels of AGI for operationalizing progress on the path to AGI. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 36308–36321, 2024.

[183] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of GPT-4. *arXiv preprint arXiv:2306.02707*, 2023.

[184] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan O. Arik, and Tomas Pfister. MLE-STAR: Machine learning engineering agent via search and targeted refinement. *arXiv preprint arXiv:2506.15692*, 2025.

[185] Muhammad U. Nasir, Sam Earle, Christopher Cleghorn, Steven James, and Julian Togelius. LLMatic: Neural architecture search via large language models and quality diversity optimization. *arXiv preprint arXiv:2306.01102*, 2023.

[186] Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Nicolaus Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. MLGym: A new framework and benchmark for advancing AI research agents. In *Second Conference on Language Modeling*, 2025.

[187] Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A. Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, Franck Dernoncourt, and Tianyi Zhou. DynaSaur: Large language agents beyond predefined actions. *arXiv preprint arXiv:2411.01747*, 2024.

[188] Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I. Wang, and Xi Victoria Lin. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 26106–26128, 2023.

[189] Jingjie Ning, Xueqi Li, Yibo Kong, and Dongting Li. Whatworkedbench: Benchmarking experimental understanding in ai agents, 2026.

[190] Alexander Novikov, Ngan Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[191] Gabriele Oliaro, Jaeseong Lee, Yichao Fu, Zhaoyuan Su, Owen Lu, Sreeram Vennam, May Jiang, Junli Wang, Hao Zhang, Zhihao Jia, and Samyam Rajbhandari. Fastkernels: Benchmarking gpu kernel generation in production, 2026.

[192] Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*, 2024.

[193] Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can llms write efficient GPU kernels?, 2025.

[194] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[195] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[196] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

[197] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated reinforcement learning (AutoRL): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, 2022.

[198] Nico Pelleriti, Sree Harsha Nelaturu, Zhanke Zhou, Zongze Li, Max Zimmer, Bo Han, and Sebastian Pokutta. What do evolutionary coding agents evolve? *arXiv preprint arXiv:2605.20086*, 2026.

[199] Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[200] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3419–3448, 2022.

[201] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[202] Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. GrIPS: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*, 2022.

[203] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[204] Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, 2023.

[205] Cheng Qian, Chi Han, Yi R. Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. CREATOR: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6922–6939, 2023.

[206] Cheng Qian, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Toolink: Linking toolkit creation and using through chain-of-solving on open-source model. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 831–854, 2024.

[207] Cheng Qian, Wenting Zhao, Liangwei Yang, Heng Wang, Jielin Qiu, Heng Ji, Silvio Savarese, Huan Wang, and Shelby Heinecke. AI4AI at test-time: Strong-to-weak capability transfer via harnesses. *arXiv preprint arXiv:2608.12307*, 2026.

[208] Kiran Ramnath, Kang Zhou, Sheng Guan, Soumya Smruti Mishra, Xuan Qi, Zhengyuan Shen, Shuai Wang, Sangmin Woo, Sullam Jeoung, Yawei Wang, Haozhu Wang, Han Ding, Yuzhe Lu, Zhichao Xu, Yun Zhou, Balasubramaniam Srinivasan, Qiaojing Yan, Yueyan Chen, Haibo Ding, Panpan Xu, and Lin Lee Cheong. A systematic survey of automatic prompt optimization techniques. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33078–33110, 2025.

[209] Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can LLM agents automate LLM post-training? In *Forty-third International Conference on Machine Learning*, 2026.

[210] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[211] Esteban Real, Chen Liang, David So, and Quoc V. Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8007–8019, 2020.

[212] Colorado J. Reed, Sean Metzger, Aravind Srinivas, Trevor Darrell, and Kurt Keutzer. SelfAugment: Automatic augmentation policies for self-supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2674–2683, 2021.

[213] Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, R. B. Xiong, Qingfeng Lan, Wenyi Wang, Nanbo Li, Yibo Yang, Mingchen Zhuge, and Jürgen Schmidhuber. Self-improvements in modern agentic systems: A survey. *arXiv preprint arXiv:2607.13104*, 2026.

[214] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[215] Rajarshi Roy, Jonathan Raiman, Neel Kant, Ilyas Elkin, Robert Kirby, Michael Siu, Stuart Oberman, Saad Godil, and Bryan Catanzaro. PrefixRL: Optimization of parallel prefix circuits using deep reinforcement learning. *arXiv preprint arXiv:2205.07000*, 2022.

[216] Noveen Sachdeva, Benjamin Coleman, Wang-Cheng Kang, Jianmo Ni, Lichan Hong, Ed H. Chi, James Caverlee, Julian McAuley, and Derek Zhiyuan Cheng. How to train data-efficient LLMs. *arXiv preprint arXiv:2402.09668*, 2024.

[217] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using LLM agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

[218] Tobias Schnabel and Jennifer Neville. Symbolic prompt program search: A structure-aware approach to efficient compile-time prompt optimization. *arXiv preprint arXiv:2404.02319*, 2024.

[219] Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent, 2025.

[220] Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, Shauna Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models. *arXiv preprint arXiv:2310.13548*, 2023.

[221] Quan Shi, Keshav Dhandhania, Karthik Narasimhan, and Victor Barres. τ^{τ}-bench: An environment for end-to-end, realistic agent construction, 2026.

[222] Taylor Shin, Yasaman Razeghi, Robert L. Logan, IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*, 2020.

[223] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[224] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. NAS-bench-301 and the case for surrogate benchmarks for neural architecture search, 2020.

[225] Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

[226] Kartik Singhal and Gautam Shroff. ConceptSearch: Towards efficient program search using LLMs for abstraction and reasoning corpus (ARC). *arXiv preprint arXiv:2412.07322*, 2024.

[227] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. *arXiv preprint arXiv:2209.13085*, 2022.

[228] Essam Sleiman, Mersad Abbasi, Ayush Chakravarthy, and Karina Nguyen. Autoharnessbench: Ai r&d at the harness layer. https://github.com/essamsleiman/autoharnessbench, July 2026.

[229] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[230] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[231] Joseph Spracklen, Raveen Wijewickrama, A. H. M. Nazmus Sakib, Anindya Maiti, Bimal Viswanath, and Murtuza Jadliwala. We have a package for you! A comprehensive analysis of package hallucinations by code generating LLMs. *arXiv preprint arXiv:2406.10279*, 2024.

[232] Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive self-improvement of AI research agents. *arXiv preprint arXiv:2609.26457*, 2026.

[233] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 56843–56873, 2025.

[234] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming common crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*, 2024.

[235] Tianxiang Sun, Zhengfu He, Hong Qian, Yunhua Zhou, Xuanjing Huang, and Xipeng Qiu. BBTv2: Towards a gradient-free future with large language models. *arXiv preprint arXiv:2205.11200*, 2022.

[236] Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. Black-box tuning for language-model-as-a-service. *arXiv preprint arXiv:2201.03514*, 2022.

[237] Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A. Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9275–9293, 2020.

[238] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

[239] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2820–2828, 2019.

[240] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, 2019.

[241] Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.

[242] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. VeriGen: A large language model for verilog code generation. *arXiv preprint arXiv:2308.00708*, 2023.

[243] Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. AutoChip: Automating HDL generation using LLM feedback. *arXiv preprint arXiv:2311.04887*, 2023.

[244] Lucas Y. Tian, Kevin Ellis, Marta Kryven, and Joshua B. Tenenbaum. Learning abstract structure for drawing by efficient motor program induction. In *Advances in Neural Information Processing Systems*, volume 33, pages 2686–2697, 2020.

[245] Guiyao Tie, Jiawen Shi, Dingjie Song, Yixiao Huang, Ziji Sheng, Xueyang Zhou, Daizong Liu, Pan Zhou, Yongchao Chen, Ran Xu, Lifang He, Qingsong Wen, Manling Li, Cong Lu, Shuai Li, Pengtao Xie, Yixuan Yuan, Rui Meng, Lei Xing, Lichao Sun, Caiming Xiong, Philip S. Yu, and Jianfeng Gao. AutoResearch AI: Towards AI-powered research automation for scientific discovery. *arXiv preprint arXiv:2605.23204*, 2026.

[246] Guiyao Tie, Pan Zhou, and Lichao Sun. A survey of AI scientists: Surveying the automatic scientists and research. *arXiv preprint arXiv:2510.23045*, 2025.

[247] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019.

[248] Weixi Tong and Tianyi Zhang. CodeJudge: Evaluating code generation with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20032–20051, 2024.

[249] Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and Marius Lindauer. AutoML in the age of large language models: Current challenges, future opportunities and risks. *Transactions on Machine Learning Research*, 2024.

[250] Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. VeRO: A harness for agents to optimize agents, 2026.

[251] Xingchen Wan, Ruoxi Sun, Hootan Nakhost, and Sercan O. Arik. Teach better or show smarter? On instructions and exemplars in automatic prompt optimization. *arXiv preprint arXiv:2406.15708*, 2024.

[252] Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

[253] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[254] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8612–8620, 2019.

[255] Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9440–9450, 2024.

[256] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024.

[257] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

[258] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*, 2023.

[259] Yidong Wang, Zhuohao Yu, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, Xing Xie, Wei Ye, Shikun Zhang, and Yue Zhang. PandaLM: An automatic evaluation benchmark for LLM instruction tuning optimization. *arXiv preprint arXiv:2306.05087*, 2023.

[260] Yiting Wang, Wanghao Ye, Yexiao He, Yiran Chen, Gang Qu, and Ang Li. MCP4EDA: LLM-powered model context protocol RTL-to-GDSII automation with backend-aware synthesis optimization. *arXiv preprint arXiv:2507.19570*, 2025.

[261] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, 2023.

[262] Zheng Wang and Michael O'Boyle. Machine learning in compiler optimization. *Proceedings of the IEEE*, 106(11):1879–1901, 2018.

[263] Zhihai Wang, Zijie Geng, Zhaojie Tu, Jie Wang, Yuxi Qian, Zhexuan Xu, Ziyan Liu, Siyuan Xu, Zhentao Tang, Shixiong Kai, Mingxuan Yuan, Jianye Hao, Bin Li, Yongdong Zhang, and Feng Wu. Benchmarking end-to-end performance of AI-based chip placement algorithms. *arXiv preprint arXiv:2407.15026*, 2024.

[264] Zhiruo Wang, Daniel Fried, and Graham Neubig. TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024.

[265] Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in LLM-as-a-judge. *arXiv preprint arXiv:2410.21819*, 2024.

[266] Jiaqi Wei, Yuejin Yang, Xiang Zhang, Yuhan Chen, Xiang Zhuang, Zhangyang Gao, Dongzhan Zhou, Guangshuai Wang, Zhiqiang Gao, Juntai Cao, Zijie Qiu, Ming Hu, Chenglong Ma, Shixiang Tang, Junjun He, Chunfeng Song, Xuming He, Qiang Zhang, Chenyu You, Shuangjia Zheng, Ning Ding, Wanli Ouyang, Nanqing Dong, Yu Cheng, Siqi Sun, Lei Bai, and Bowen Zhou. From AI for science to agentic science: A survey on autonomous scientific discovery. *arXiv preprint arXiv:2508.14111*, 2025.

[267] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. *arXiv preprint arXiv:1603.01670*, 2016.

[268] Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *arXiv preprint arXiv:1912.00848*, 2019.

[269] Alexander Wettig, Aatmik Gupta, Saumya Malik, and Danqi Chen. QuRating: Selecting high-quality data for training language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 52915–52971, 2024.

[270] Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. *arXiv preprint arXiv:2007.04965*, 2020.

[271] Colin White, Willie Neiswanger, and Yash Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2019.

[272] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

[273] Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*, 2021.

[274] Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3751–3760, 2017.

[275] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

[276] Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelović, and Jakob Nikolas Kather. LLM agents making agent tools. *arXiv preprint arXiv:2502.11705*, 2025.

[277] Catherine Wong, Kevin Ellis, Joshua B. Tenenbaum, and Jacob Andreas. Leveraging language to learn program abstractions and search heuristics. *arXiv preprint arXiv:2106.11053*, 2021.

[278] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. ChatEDA: A large language model powered autonomous agent for EDA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3184–3197, 2024.

[279] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[280] Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. Meta-rewarding language models: Self-improving alignment with LLM-as-a-meta-judge. *arXiv preprint arXiv:2407.19594*, 2024.

[281] Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation in the era of large language model: Survey and roadmap. *IEEE Transactions on Evolutionary Computation*, 29(2):534–554, 2025.

[282] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

[283] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.

[284] Lichuan Xiang, Łukasz Dudziak, Mohamed S. Abdelfattah, Abhinav Mehrotra, Nicholas Donald Lane, and Hongkai Wen. Towards neural architecture search through hierarchical generative modeling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 54236–54254, 2024.

[285] Qiujie Xie, Yixuan Weng, Minjun Zhu, Fuchen Shen, Shulin Huang, Zhen Lin, Jiahui Zhou, Zilan Mao, Zijie Yang, Linyi Yang, Jian Wu, and Yue Zhang. How far are AI scientists from changing the world? *arXiv preprint arXiv:2507.23276*, 2025.

[286] Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing data mixtures speeds up language model pretraining. In *Advances in Neural Information Processing Systems*, volume 36, pages 69798–69818, 2023.

[287] Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. Data selection for language models via importance resampling. *arXiv preprint arXiv:2302.03169*, 2023.

[288] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

[289] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, 2024.

[290] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024.

[291] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned LLMs with nothing. *arXiv preprint arXiv:2406.08464*, 2024.

[292] Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

[293] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

[294] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

[295] Longxing Yang, Yu Hu, Shun Lu, Zihao Sun, Jilin Mei, Yiming Zeng, Zhiping Shi, Yinhe Han, and Xiaowei Li. DDSAS: Dynamic and differentiable space-architecture search. In *Proceedings of the 13th Asian Conference on Machine Learning*, volume 157 of *Proceedings of Machine Learning Research*, pages 284–299, 2021.

[296] Zonglin Yang, Xingtong Liu, and Xinyan Xu. SciIntegrity-Bench: A benchmark for evaluating academic integrity in AI scientist systems. *arXiv preprint arXiv:2605.10246*, 2026.

[297] Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. Justice or prejudice? Quantifying biases in LLM-as-a-judge. *arXiv preprint arXiv:2410.02736*, 2024.

[298] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.

[299] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc V. Le. BigNAS: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717, 2020.

[300] Yang Yu, Peiyu Zang, Chi Hsu Tsai, Haiming Wu, Yixin Shen, Jialing Zhang, Haoyu Wang, Zhiyou Xiao, Jingze Shi, Yuyu Luo, Wentao Zhang, Chunlei Men, Guang Liu, and Yonghua Lin. Towards automated kernel generation in the era of llms. In Diego Calvanese, editor, *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-26*, pages 8069–8077. International Joint Conferences on Artificial Intelligence Organization, August 2026.

[301] Zhongkai Yu, Chenyang Zhou, Yichen Lin, Hejia Zhang, Haotian Ye, Junxia Cui, Zaifeng Pan, Jishen Zhao, and Yufei Ding. ChipBench: A next-step benchmark for evaluating LLM performance in AI-aided chip design. *arXiv preprint arXiv:2601.21448*, 2026.

[302] Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. Craft: Customizing llms by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*, volume 2024, pages 40097–40125, 2024.

[303] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 57905–57923, 2024.

[304] Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for LLM agents. *arXiv preprint arXiv:2603.22386*, 2026.

[305] Murong Yue, Zhiwei Liu, Liangwei Yang, Jianguo Zhang, Zuxin Liu, Haolin Chen, Ziyu Yao, Silvio Savarese, Caiming Xiong, Shelby Heinecke, and Huan Wang. ToolLibGen: Scalable automatic tool creation and aggregation for LLM reasoning. *arXiv preprint arXiv:2510.07768*, 2025.

[306] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[307] Peiyu Zang, Jian Tao, Jialing Zhang, Yichen Yuan, Wentao Zhang, Guang Liu, and Yonghua Lin. Kernelgenbench: Can llms and agents write efficient kernels across operator sources and hardware platforms?, 2026.

[308] Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks. In *International Conference on Learning Representations*, 2022.

[309] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-STaR: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

[310] Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

[311] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

[312] Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

[313] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025.

[314] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Seyed Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations*, 2025.

[315] Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the importance of evolutionary search in automated heuristic design with large language models. *arXiv preprint arXiv:2407.10873*, 2024.

[316] Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. TEMPERA: Test-time prompting via reinforcement learning. *arXiv preprint arXiv:2211.11890*, 2022.

[317] Tuo Zhang, Jinyue Yuan, and Salman Avestimehr. Revisiting OPRO: The limitations of small-scale LLMs as optimizers. *arXiv preprint arXiv:2405.10276*, 2024.

[318] Yaolun Zhang, Tianyi Xu, Yujie Zhao, Jishen Zhao, Qingyun Wu, and Huazheng Wang. Evalresearch-bench: Can ai agents build their own feedback for recursive self-improvement?, 2026.

[319] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. Neural prompt search. *arXiv preprint arXiv:2206.04673*, 2022.

[320] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023.

[321] Bingchen Zhao, Despoina Magka, Minqi Jiang, Xian Li, Roberta Raileanu, Tatiana Shavrina, Jean-Christophe Gagnon-Audet, Kelvin Niu, Shagun Sodhani, Michael Shvartsman, Andrei Lupu, Alisia Lupidi, Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Thomas Foster, Lucia Cipolina-Kun, Abhishek Charnalia, Derek Dunfield, Alexander H. Miller, Oisin Mac Aodha, Jakob Foerster, and Yoram Bachrach. The automated LLM speedrunning benchmark: Reproducing NanoGPT improvements. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

[322] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

[323] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 863–879, 2020.

[324] Tianshi Zheng, Zheye Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17733–17750, 2025.

[325] Yandan Zheng, Haoran Luo, Zhenghong Lin, Wenjin Liu, and Luu Anh Tuan. Benchbench: Benchmarking automated benchmark generation, 2026.

[326] Ruizhe Zhong, Xingbo Du, Shixiong Kai, Zhentao Tang, Siyuan Xu, Hui-Ling Zhen, Jianye Hao, Qiang Xu, Mingxuan Yuan, and Junchi Yan. LLM4EDA: Emerging progress in large language models for electronic design automation. *arXiv preprint arXiv:2401.12224*, 2023.

[327] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023.

[328] Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. AutoPEFT: Automatic configuration search for parameter-efficient fine-tuning. *Transactions of the Association for Computational Linguistics*, 12:525–542, 2024.

[329] Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

[330] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-discover: Large language models self-compose reasoning structures. *arXiv preprint arXiv:2402.03620*, 2024.

[331] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023.

[332] Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. *arXiv preprint arXiv:2310.17631*, 2023.

[333] Minjun Zhu, Qiujie Xie, Yixuan Weng, Jian Wu, Zhen Lin, Linyi Yang, and Yue Zhang. AI scientists fail without strong implementation capability. *arXiv preprint arXiv:2506.01372*, 2025.

[334] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62743–62767, 2024.

[335] Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-PyTorch: Multi-fidelity MetaLearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079–3090, 2021.

[336] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.