OPEN ACCESS

COMMENTARY

INNOVATION PRESS

Journal homepage: www.the-innovation.org/ai-plus

AI Plus

AI Plus 1 (2026):100009

Contents lists available at INNOVATION PRESS

From general problem solver to scalable agentic problem solver

Shengcai Liu[1] **and Ke Tang**[1,*]

[1]Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China

*Correspondence: tangk3@sustech.edu.cn

Received: May 21, 2026; Accepted: August 15, 2026; Published Online: August 18, 2026; https://doi.org/10.59717/ipj.aiplus.2026.100009

Citation: Liu S. and Tang K. (2026). From general problem solver to scalable agentic problem solver. *AI Plus* **1**:100009.

FROM GPS TO APS

In 1959, Herbert A. Simon, John C. Shaw, and Allen Newell described the General Problem Solver (GPS), an early attempt to build a machine that could solve problems by manipulating symbolic representations of goals, states, and operators.[1] GPS was intellectually bold: it suggested that problem solving might be organized as a general computational process. Yet its ambition quickly met the constraints of its time. Problems had to be formalized in advance, operators had to be specified by humans, and demonstrations were largely restricted to well-structured symbolic domains. GPS offered a vision of generality, but not a scalable bridge from messy reality to executable decisions.

Modern optimization research faces a similar limitation. Standard benchmark libraries such as MIPLIB have been essential for comparing solvers under controlled conditions.[2] However, these benchmarks assume that a real-world problem has already been formulated as a well-defined mathematical instance. In practice, much of the hard work lies before and after that formal-

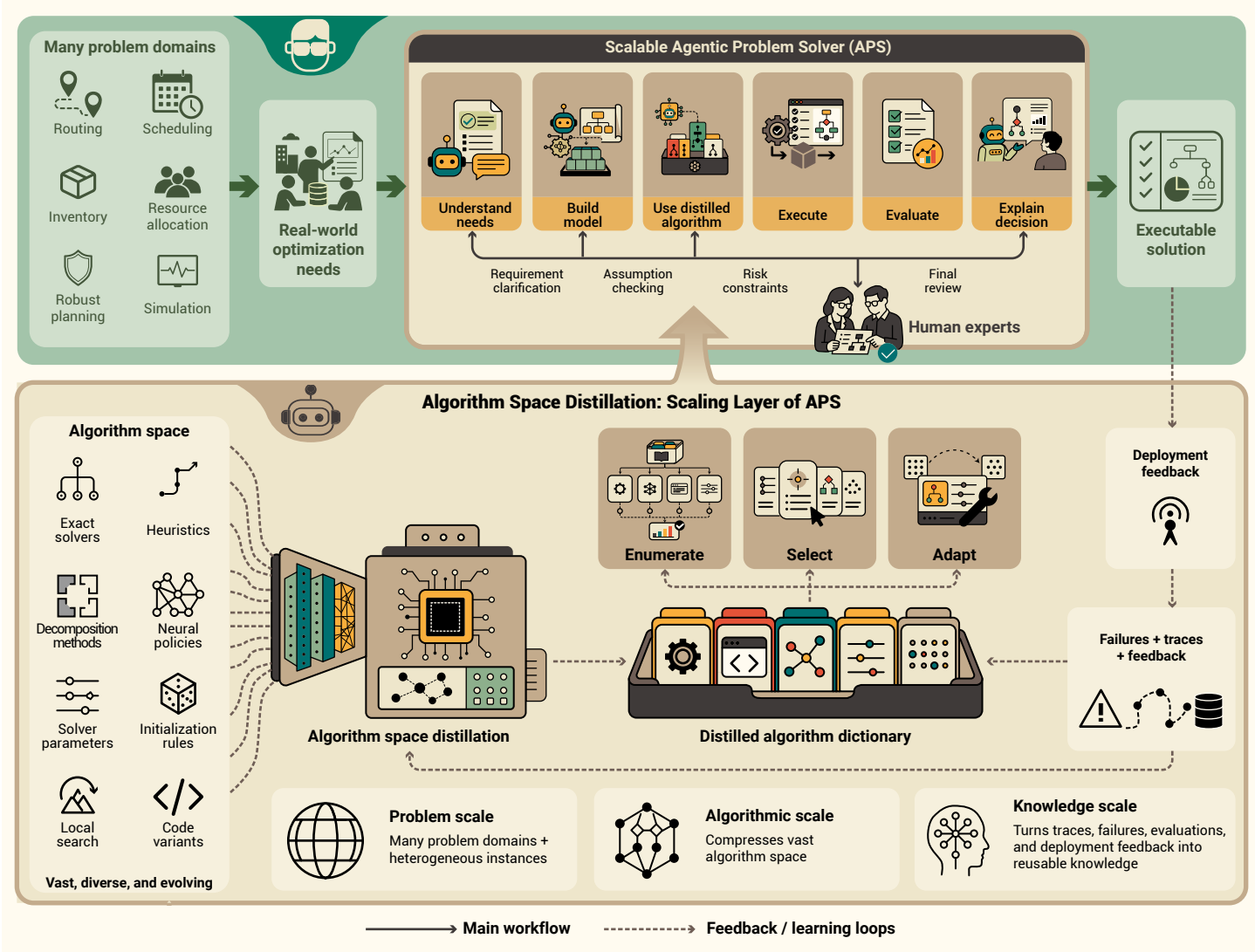

Figure 1. Illustration of APS and algorithm space distillation as its scaling layer.

ization: translating ambiguous needs, imperfect data, and hidden constraints into a validated, explainable decision. The central gap is not merely between a weak solver and a stronger solver. It is between a real pain point and a trustworthy executable plan.

We propose Agentic Problem Solver (APS) as a new paradigm for optimization research and deployment: a scalable, tool-using agentic system that translates real-world decision needs into executable and explainable solutions. As illustrated in Figure 1, APS is neither a larger solver nor merely a large language model (LLM) wrapper around existing solvers. It is a scalable problem-solving architecture that coordinates the full path from requirement understanding to modeling, algorithm use, execution, evaluation, and explanation. More explicitly, an APS maps a real-world request and available data into a decision package: it elicits and formalizes the request into a problem instance, selects and executes algorithmic tools, evaluates feasibility and performance, and returns a model, assumptions, validation evidence, recommended actions, and an explanation that humans can inspect or revise. This path has become viable because of advances in several complementary technologies: LLMs can interpret and generate structured artifacts; agents can use external tools; mature solvers and simulators provide executable feedback; and accumulated traces can be reused across tasks. APS therefore revisits the ambition of general problem solving, but with a practical optimization-centered architecture built for scale.

APS-LIKE SYSTEMS ARE ALREADY EMERGING

The case for APS does not rest only on speculation. Early APS-like systems already demonstrate fragments of the full vision. They can be grouped by the part of the decision pipeline they automate.

First, automatic modeling and solving agents focus on problem formulation and solver execution. OptiMUS showed that LLM agents can translate natural-language optimization tasks into linear and mixed-integer formulations, generate solver code, execute it, and use feedback to repair errors.[3] More recent systems extend this line of work from prompt-based approaches to trainable and deployable modeling agents. LLMOPT introduces a learning-based framework with a general formulation scheme, multi-instruction tuning, model alignment, and self-correction to improve generalization across diverse optimization problem types.[4] ORLM further develops open-source and customizable operations research (OR) modeling LLMs, using synthesized OR instruction data and an industrial benchmark to support domain adaptation and practical deployment.[5] Together, these works show a progression from automatic formulation and code execution to specialized, trainable, and benchmarked APS-like optimization agents.

Second, interactive optimization copilots focus on model use and interpretation. Systems such as OptiGuide, OptiChat, and Gurobi AI Modeling support what-if analysis, sensitivity analysis, infeasibility diagnosis, model explanation, and natural-language interaction with existing optimization assets. This category highlights another requirement for APS: the system must not only produce a decision but also make the model and its outputs inspectable by the domain experts responsible for the application context.

Third, production-oriented decision agents focus on operational workflows. Systems such as ORPilot and NVIDIA cuOpt-based supply-chain agents address ambiguity, external data, solver portability, and domain-specific planning. ORPilot supports conversational requirement elicitation, data collection, solver-agnostic intermediate representations, and traceback-guided repair; NVIDIA's cuOpt demonstrations provide logistics-optimization tools that LLM agents can invoke for supply-chain planning. These efforts suggest that APS-like systems are beginning to support multiple stages of the full lifecycle of decision optimization.

Taken together, these systems demonstrate that LLMs can already participate in optimization workflows, from modeling and code generation to model interaction and operational planning. However, a key unresolved challenge is ***scale***: how to move from isolated pipelines and domain-specific copilots to reusable and generalizable problem-solving systems.

THE SCALING CHALLENGE

This scaling challenge is broader than model size or benchmark size. It concerns whether the entire problem-solving loop can scale across problem domains, algorithmic choices, and accumulated knowledge.

The first dimension is problem scale. Real optimization problems rarely arrive as textbook linear programs. They may combine routing, scheduling, inventory, resource allocation, robust planning, simulation, forecasting, and human preferences, with missing data or objectives negotiated during deployment. A scalable APS must therefore operate over a spectrum of optimization problems rather than a narrow set of benchmark problems.

The second dimension is algorithmic scale. An APS should not merely generate a model and call a default solver. The relevant action space may include many types of algorithmic strategies, from solver configurations and heuristics to learned policies, and even executable code variants, as illustrated by recent progress in LLM-guided program search.[6] Searching this space from scratch for every request is often computationally impractical. The system needs explicit, testable, and reusable algorithmic knowledge.

The third dimension is knowledge scale. If each APS deployment remains an isolated pipeline, the system will repeatedly rediscover the same modeling patterns, solver settings, and failure modes. A scalable APS must accumulate experience from previous instances, solver traces, failed attempts, synthetic cases, and deployment feedback, then convert that experience into reusable problem-solving capability.

ALGORITHM SPACE DISTILLATION AS THE SCALING LAYER

In optimization, an algorithm can be viewed as a mapping from a problem instance to a solution. Algorithm design is therefore a higher-level mapping from a problem instance to an algorithm.[7] However, directly learning such a mapping over a large algorithm space is difficult. Algorithm space distillation addresses this difficulty by converting a large space of possible algorithmic strategies into a compact dictionary of reusable strategies. It can therefore serve as a scaling layer for APS: instead of searching the full algorithmic action space for every new request, an APS can select, enumerate, or adapt strategies from the distilled dictionary.

This idea can be made explicit with a compact formalization. Let P denote a set (space) of problem instances of interest. Let A denote a broad algorithm space, including solvers, parameter settings, decomposition templates, heuristics, learned policies, initialization rules, and executable code variants. Let $L(a,p)$ denote a scalar performance score for applying algorithmic strategy a to problem instance p, which may measure solution quality and runtime. Algorithm space distillation aims to derive a compact dictionary $T \subseteq A$ with size at most K:

$$\max_{\substack{T \subseteq A \\ |T| \le K}} \sum_{p \in P} w(p) \max_{a \in T} L(a,p).$$

Here, higher values of L are better, $w(p)$ represents the frequency or importance of instance p, and K controls dictionary size. The term $\max_{a \in T} L(a,p)$ represents the performance of T on instance p, assuming APS can use the best available dictionary member on that instance. The objective maximizes aggregate weighted performance over P while keeping T compact. Intuitively, T should contain complementary strategies so that important regions of P are covered by at least one well-performing member in T.

In an idealized finite setting, this objective admits an approximation guarantee. Assume that A is finite, that $1 \le K \le |A|$, and that $L(a,p) \ge 0$ for every $a \in A$ and $p \in P$. For nonempty T, define $m_T(p) = \max_{a \in T} L(a,p)$, and use the convention $m_\varnothing(p) = 0$. Define $F(T) = \sum_{p \in P} w(p) m_T(p)$, so that $F(\varnothing) = 0$.

For any $S \subseteq T$ and $a \notin T$, the marginal gain of adding a satisfies

$$\Delta(a|S) = F(S \cup \{a\}) - F(S) = \sum_{p \in P} w(p) \max\{0, L(a,p) - m_S(p)\} \ge \sum_{p \in P} w(p) \max\{0, L(a,p) - m_T(p)\} = \Delta(a|T).$$

The displayed diminishing-returns inequality establishes that F is submodular. Moreover, because $w(p) \ge 0$, every marginal gain is nonnegative, so F is monotone.

Consider a greedy distillation procedure initialized with $T_0 = \varnothing$. At each step $i = 0, \ldots, K-1$, it chooses $a_i \in \operatorname{argmax}_{a \in A \setminus T_i} (a|T_i)$ and sets $T_{i+1} = T_i \cup \{a_i\}$. Let $_i = (a_i|T_i) = F(T_{i+1}) - F(T_i)$ and let T^* be an optimal dictionary satisfying $|T^*| \le K$.

Monotonicity and submodularity give

$$F(T^*) - F(T_i) \le F(T_i \cup T^*) - F(T_i) \le \sum_{a \in T^* \setminus T_i} (a|T_i) \le K_i.$$

Therefore, $_i \ge [F(T^*) - F(T_i)]/K$, and the residual gap satisfies

$$F(T^*) - F(T_{i+1}) \le (1 - 1/K)[F(T^*) - F(T_i)].$$

Iterating this inequality from $T_0 = \varnothing$ gives

$$F(T_K) \ge \left[1 - (1 - 1/K)^K\right] F(T^*) \ge (1 - 1/e) F(T^*).$$

Therefore, in this idealized finite setting, the greedy distillation algorithm achieves a $(1 - 1/e)$-approximation to the optimal dictionary of size at most K. This guarantee assumes that each greedy step can identify the algorithm with the largest exact marginal gain over $A \setminus T_i$ using evaluations on the complete finite set P, which may be computationally expensive. In practice, the algorithm space may be infinite, only a limited sample of instances from P may be available, and evaluations of L may be costly or noisy. Reliable approximate distillation under these conditions remains an open problem.

The distilled dictionary T is a reusable algorithmic memory that APS can use in three modes. In **enumerate** mode, APS evaluates all dictionary members and selects the best observed result. In **select** mode, APS chooses one or a few promising dictionary members before execution. In **adapt** mode, APS uses a dictionary member as a starting point and specializes it for the current instance. These three modes replace the online search over A with a more controlled problem of reusing T. They also connect APS to established approaches in algorithm selection and automated machine learning (AutoML). Specifically, select mode borrows the central idea of per-instance algorithm selection: using features of a problem instance to choose a promising method from a portfolio.[8] Enumerate mode resembles parallel algorithm portfolios, where multiple complementary solvers are run in parallel and the best found solution is returned.[9] Adapt mode is related to algorithm configuration and AutoML-style search over method variants.[10]

A schematic routing example makes this concrete. Suppose an APS is asked to plan daily deliveries with vehicle capacities, time windows, uncertain travel times, and last-minute orders. These requests define a family of routing instances P. The algorithm space A may include exact solvers, large-neighborhood search, repair heuristics, and learned initializers. Distillation would construct a compact dictionary T of complementary strategies, such as an exact solver for small tightly constrained instances, a large-neighborhood-search template for medium-scale routes, and a fast repair heuristic for dynamic orders. For a new request, APS may enumerate several dictionary members, select one using features such as fleet size and time-window tightness, or adapt one by tuning penalties or repair rules.

This mechanism addresses the three scaling dimensions described above. For problem scale, the dictionary T is constructed over an instance space P rather than a single instance; different members of T can specialize to different structural regions of P, as captured by the weighted objective over instances. For algorithmic scale, the size constraint $|T| \le K$ makes the online action space smaller and more explicit, while still allowing T to contain heterogeneous algorithmic strategies from A. For knowledge scale, past evaluations represented by $L(a, p)$, together with solver traces and deployment feedback, become signals for updating the dictionary and its modes of use rather than remaining isolated records.

The distillation process itself can also scale with data and computing resources. More candidate algorithms enlarge A, the raw space from which useful specialists may be selected. More problem instances reveal finer structure in P. As more evaluations, traces, and deployment feedback become available, they can improve both the dictionary and its modes of use.

BEYOND SCALING

Algorithm space distillation addresses a central scaling bottleneck, but it is not the whole APS agenda. A practical APS must also be verifier-grounded: its models, solver code, feasibility claims, objective values, and explanations should be checked against domain rules and executable evidence. It needs human oversight because real objectives are often negotiated, constraints may be implicit, and final decisions require human confirmation.

If successful, APS would reorganize rather than replace optimization expertise. Human experts would define problem domains, curate data, specify unacceptable risks, inspect edge cases, and improve the distilled algorithmic library. APS would handle much of the repetitive work of turning problem statements into validated decision packages. Its long-term promise is accumulated competence: learning from many problems, algorithms, failures, and deployment feedback to produce better decisions for the next real-world task.

This Commentary presents APS as a conceptual architecture and research agenda; it does not provide a concrete APS implementation or controlled experimental evaluation, which we leave as important future work. Future APS evaluation should assess not only solution quality and runtime, but also formulation correctness, feasibility checking, robustness to underspecified requirements, explanation quality, and human intervention. A related empirical example is the automatic construction of parallel algorithm portfolios.[9] This work constructs compact portfolios of complementary algorithm configurations from a parameter space and demonstrates improved generalization performance relative to existing portfolio-construction approaches across three binary optimization problem classes. Although this evidence does not validate the full APS framework, it shows that compact sets of algorithmic strategies can be automatically constructed and evaluated in a restricted setting.

REFERENCES

1. Newell A., Shaw J.C. and Simon H.A. (1959). Report on a general problem-solving program. *Proc. Int. Conf. Inf. Process.*, pp. 256–264.
2. Gleixner A., Hendel G., Gamrath G. et al. (2021). MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Math. Program. Comput.* **13**(3):443–490. DOI:10.1007/s12532-020-00194-3
3. AhmadiTeshnizi A., Gao W. and Udell M. (2024). OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. *Proc. Mach. Learn. Res.*, pp. 577–596.
4. Jiang C., Shu X., Qian H. et al. (2025). LLMOPT: Learning to define and solve general optimization problems from scratch. *Proc. Int. Conf. Learn. Represent.*
5. Huang C., Tang Z., Hu S. et al. (2025). ORLM: A customizable framework in training large models for automated optimization modeling. *Oper. Res.* **73**(6):2986–3009. DOI:10.1287/opre.2024.1233
6. Romera-Paredes B., Barekatain M., Novikov A. et al. (2024). Mathematical discoveries from program search with large language models. *Nature* **625**:468–475. DOI:10.1038/s41586-023-06924-6
7. Tang K. and Yao X. (2024). Learn to optimize—a brief overview. *Natl. Sci. Rev.* **11**:nwae132. DOI:10.1093/nsr/nwae132
8. Moradi B., Muñoz M.A. and Kirley M. (2026). Beyond distribution shift: Investigating generalisation in automated algorithm selection for single-objective black-box optimisation. *IEEE Trans. Evol. Computat.*, pp. **1**–1. DOI:10.1109/TEVC.2026.3697890.
9. Wang Z., Liu S., Yang P. et al. (2025). Evolving generalizable parallel algorithm portfolios for binary optimization problems via domain-agnostic instance generation. *IEEE Trans. Evol. Computat.*, pp. **1**–1. DOI:10.1109/TEVC.2025.3635185.
10. Baratchi M., Wang C., Limmer S. et al. (2024). Automated machine learning: Past, present and future. *Artif. Intell. Rev.* **57**:122. DOI:10.1007/s10462-024-10726-1

DECLARATION OF INTERESTS

The authors declare no competing interests.