大模型驱动的优化

以 MILP 问题为例

Huigen Ye

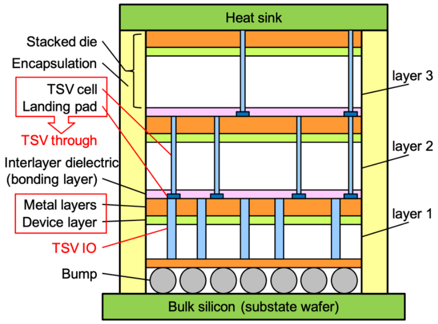
Department of Computer Science&Technology, Tsinghua University

Workshop at LEAD2025, China, September, 2025

Background

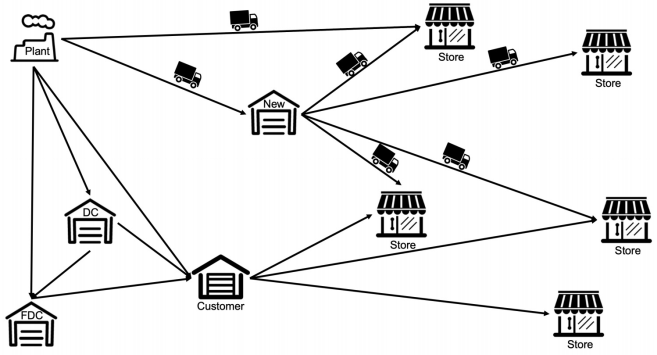
- **Several Optimization Cases**
  - 3D IC Partitioning Problem(Meitei et al., 2020)
  - Pickup and Delivery Problem with Time Windows(Dumas et al., 1991)
  - Supply Chain Management Problem(Villa, 2001)
- **Problem Definition**
  - Definition:

$$\min_x c^T x, \qquad\qquad (1)$$
$$\text{subject to } Ax \leq b, l \leq x \leq u, x \in \mathbb{Z}^n.$$



(a) IC Partitioning Problem

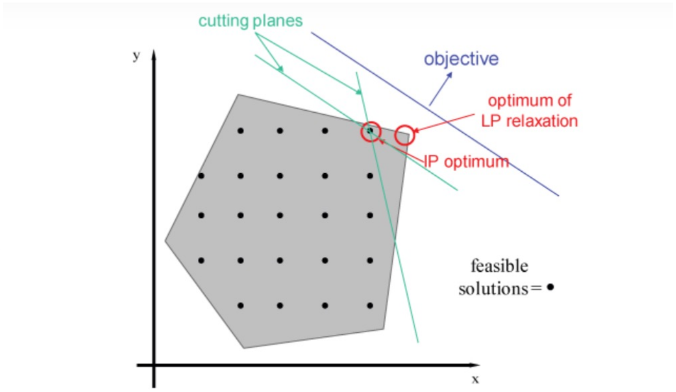
(b) Pickup and Delivery Problem



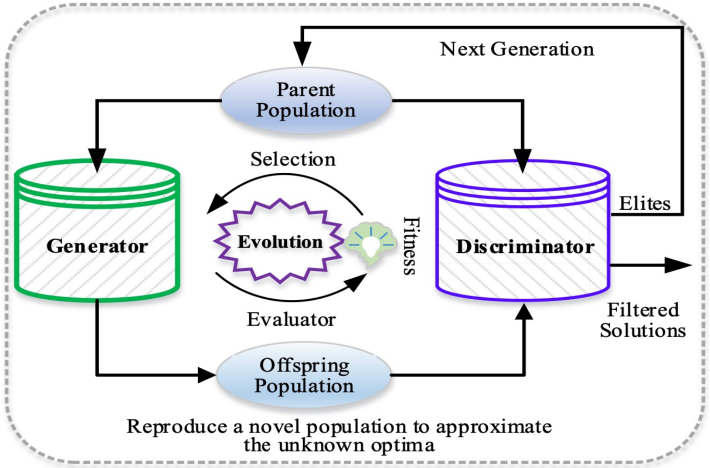
(c) Supply Chain Management

Background

- **Traditional Mathematical and Exact Methods**
  - **Exact Methods:**
    - **Solver:** SCIP(Achterberg, 2009), IPOPT(Biegler et al. 2009), Gurobi(Pedroso 2011), CPLEX(Bliek1ú et al., 2014)
    - **Academic Progress:** Advanced Branch-and-Bound Techniques(Morrison et al., 2016), Cutting Plane Methods(Dey et al., 2018)
  - **Heuristics Method**
    - Large Neighborhood Search(LNS)(Song et al., 2020)
    - Adaptive Constraint Partition(Ye et al., 2023)
  - **Traditional ML-based Method**(Liu et al., 2023)
- **Challenges**
  - **Exact Methods**: Scalability Issues & Exponential Complexity
  - **Heuristics Method:** Careful Parameter Tuning & Cold Start Issues
  - **Traditional ML-based Method:** High sample collection or training costs



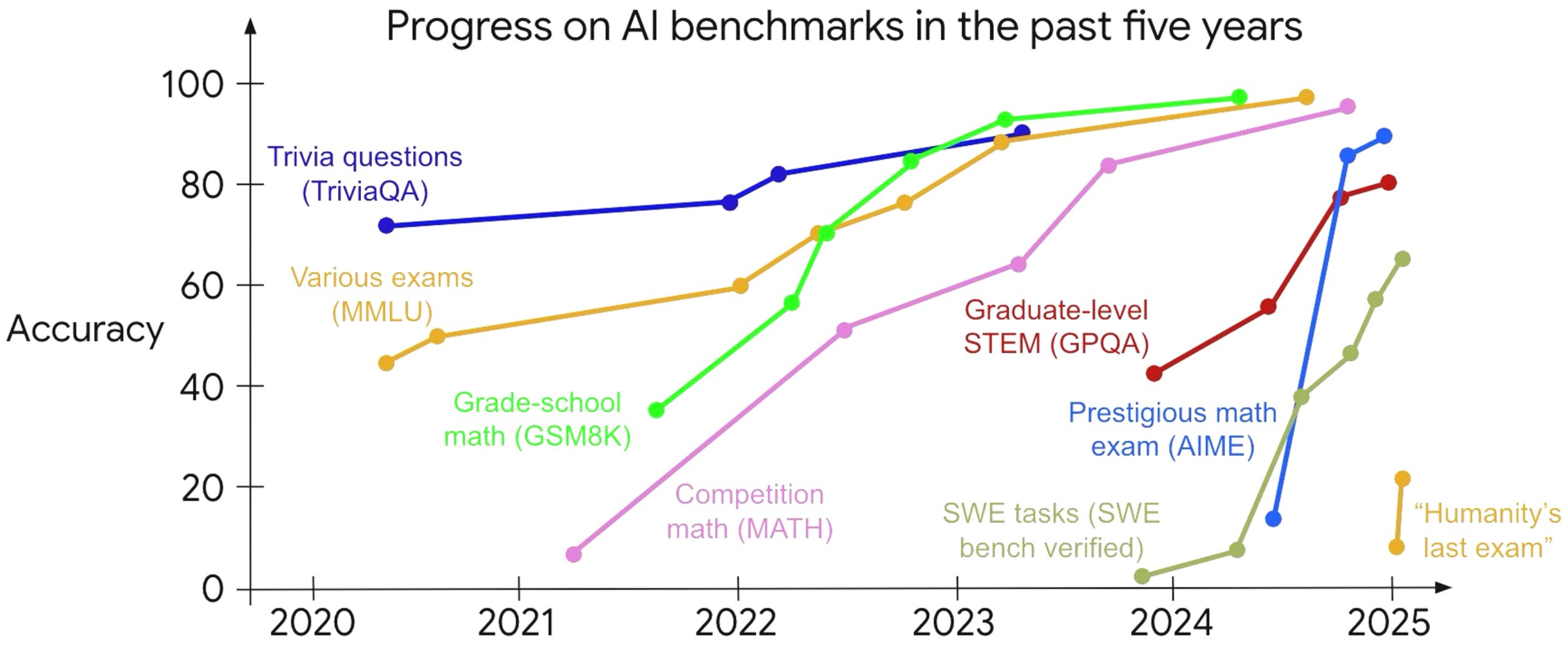
(a) Branch-and-Bound Techniques



(b) Evolution Optimization Method

Background

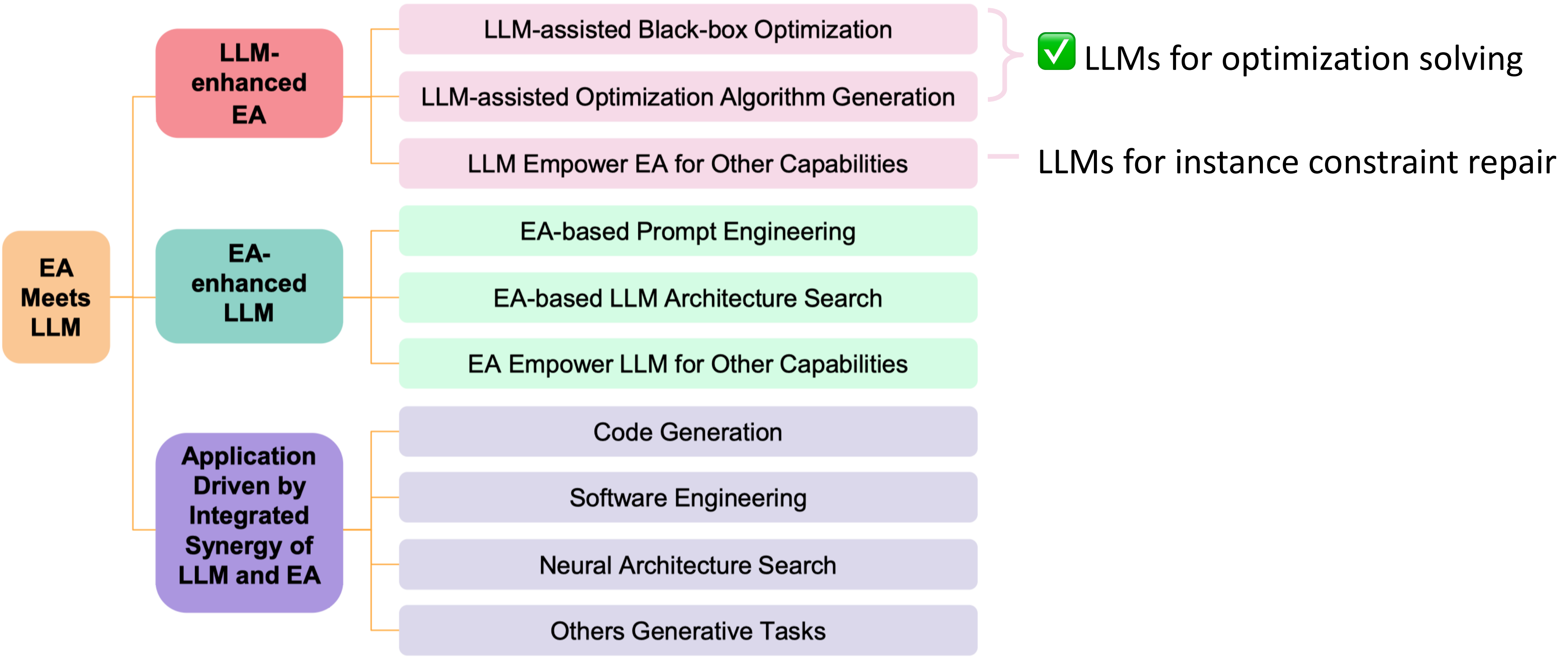
- **Rapid Progress of Large Models**
  - LLM capabilities have accelerated rapidly
  - Complex tasks are now within reach
  - “Impossible” problems are becoming solvable



(a) Breakthroughs on AI benchmarks from 2020 to 2025(Yao et al., 2025)

Background

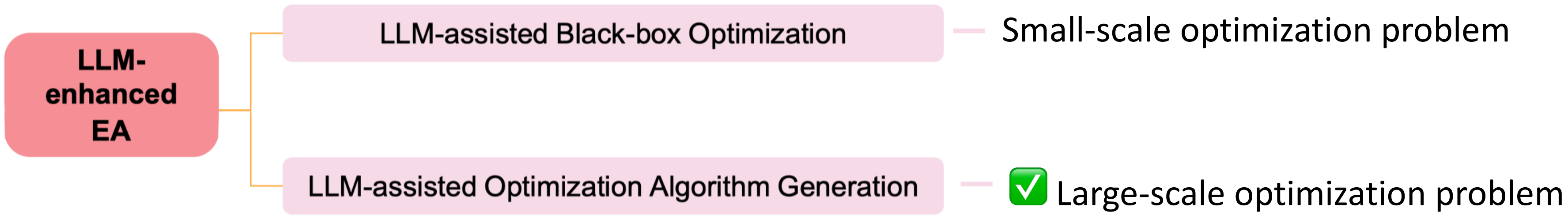
- **How LLMs Enhance Evolutionary Algorithms**
  - LLMs for optimization solving
    - Black-box optimization
    - Algorithm generation
  - LLMs for repairing infeasible optimization models



(a) Categorization of research works on the integration of LLMs and Evolutionary Algorithms (EAs)(Wu et al., 2024)

Background

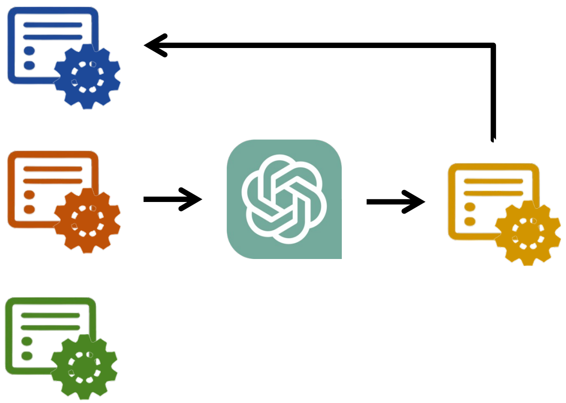
- **How LLMs Enhance Evolutionary Algorithms**
  - **Black-box Optimization**
    - LLM acts as a solver via dialogue
    - Or serves as a search operator
    - Easy to use, but limited by context & reasoning
    - ⚠️ Only suitable for small-scale problems
  - **Algorithm Generation**
    - LLM generates optimization algorithms
    - Algorithms are shorter than full problem inputs
    - Leverages LLM's code generation strength
    - ✅ Scalable to large-scale problems



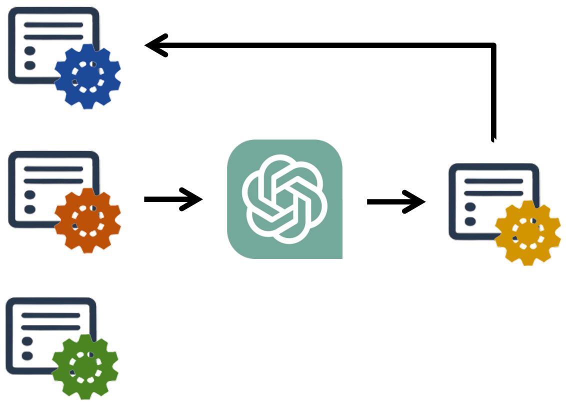
(a) Two main LLM-enhanced EA approaches for optimization solving (Wu et al., 2024)

Basic Method

- **How EA Combines with LLM to Generate Optimization Code**
  - **Whole-code Evolution**
    - Directly evolve entire heuristic codes
    - Different paradigms may mix (e.g. Genetic Algorithm + Feasibility Pump)
    - Easy to use, but limited by context & reasoning
    - ⚠️ Often unstable due to incompatible structures
  - **Operator-level Evolution**
    - Fix a high-quality heuristic framework (e.g. Large Neighborhood Search)
    - Use EA + LLM to evolve key operators (e.g. neighborhood selection)
    - ✅ More stable and effective



(a) Evolve the entire heuristic algorithm

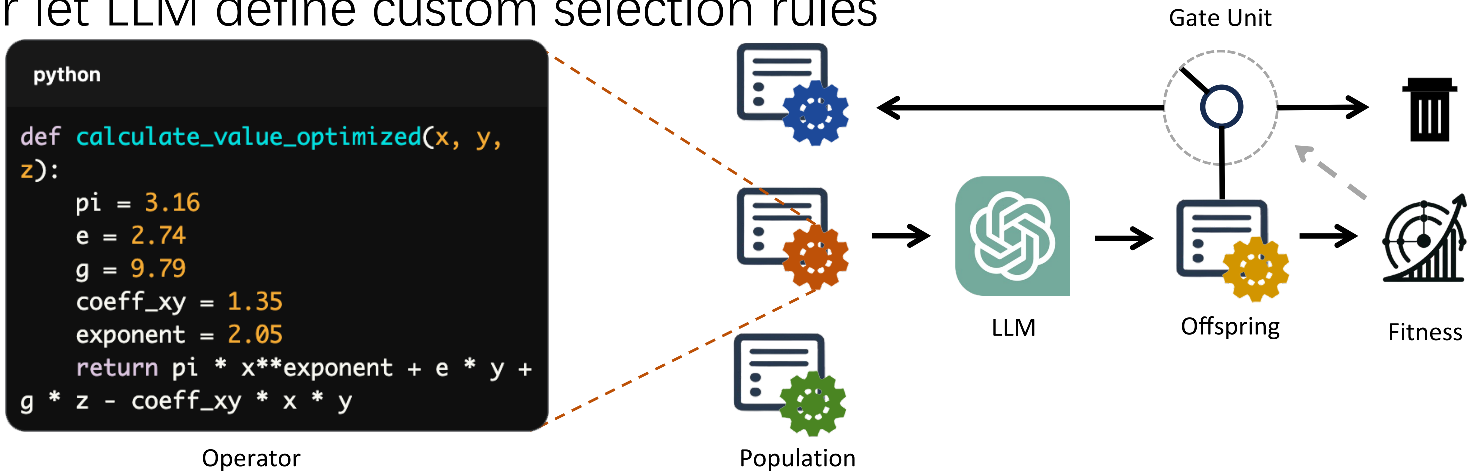


(b) Evolve a component (e.g. operator)

LEAD 2025

Basic Method

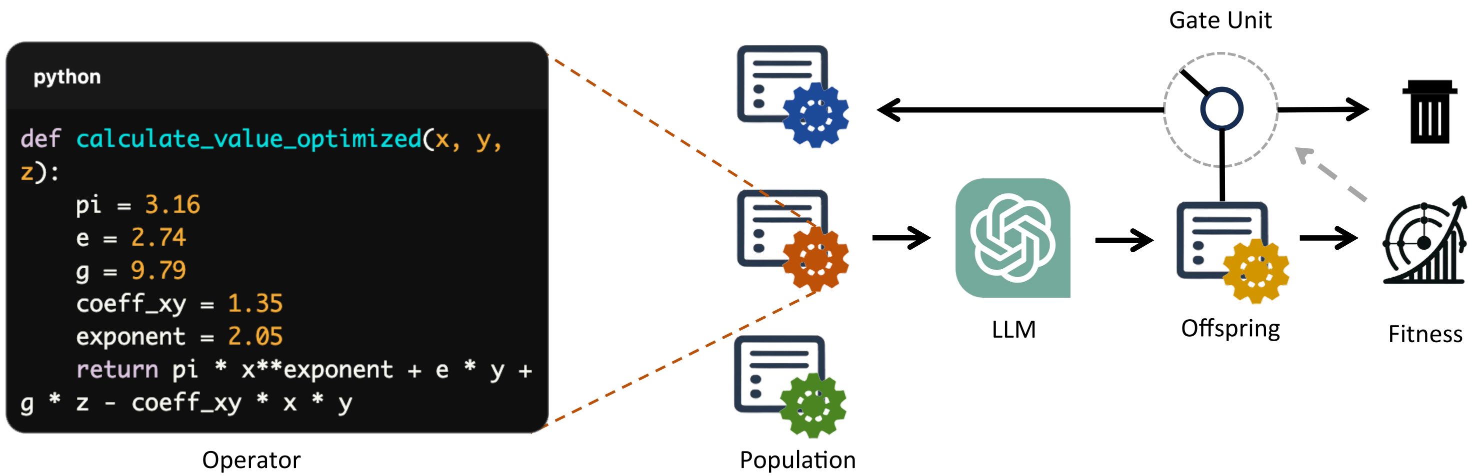
- Practical Workflow of EA + LLM for Code Evolution
  - Select Target
    - Decide which operator to evolve (e.g., neighborhood selection)
    - Define its input and output interfaces
  - Initialize Population
    - Collect existing implementation strategies
    - Or generate initial candidates via LLM using interface descriptions
  - Parent Selection
    - Use standard EA techniques (e.g., tournament, roulette wheel)
    - Or let LLM define custom selection rules



(a) EA + LLM workflow for evolving heuristic operators

Basic Method

- **Practical Workflow of EA + LLM for Code Evolution**
  - **Generate Offspring**
    - Use LLM as a crossover/mutation operator
    - Input parent code, output new candidate strategy
  - **Evaluate and Filter**
    - Run offspring on tasks to compute fitness
    - Gate unit decides whether to keep, replace, or discard based on performance
  - **Eventually, you obtain an efficient, evolved operator** 🥳 🥳



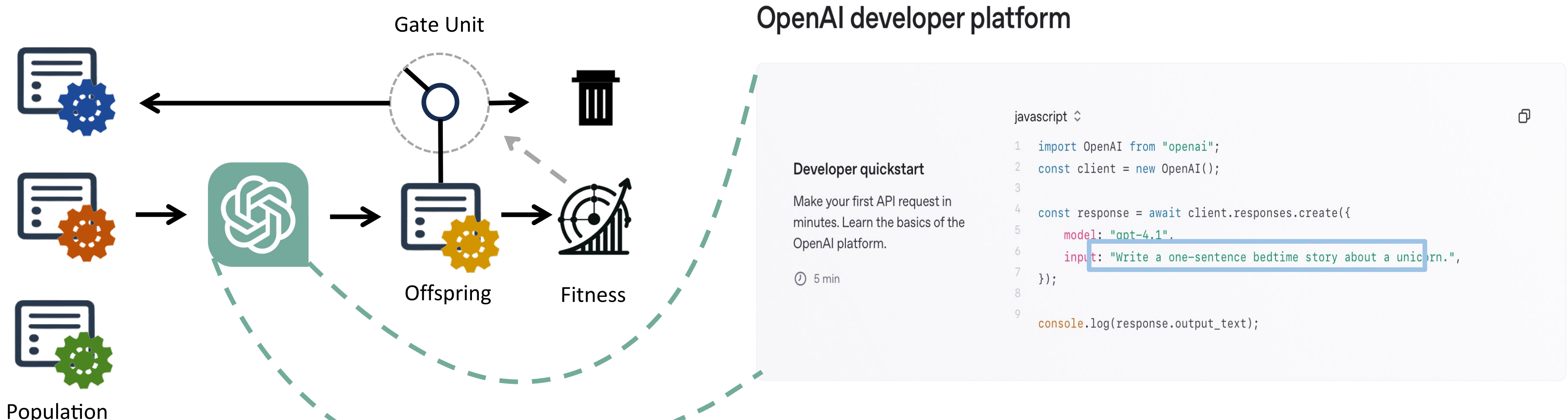
(a) EA + LLM workflow for evolving heuristic operators

LEAD 2025

Basic Method

- How to Use LLMs as Black-box Optimizers via API
  - **Key Idea**
    - We use the LLM as a black-box optimizer
    - Just send the parent strategy + prompt to the API
    - and extract new offspring code from the response
  - **Example**
    - Using the OpenAI API (https://platform.openai.com/docs/overview)
    -

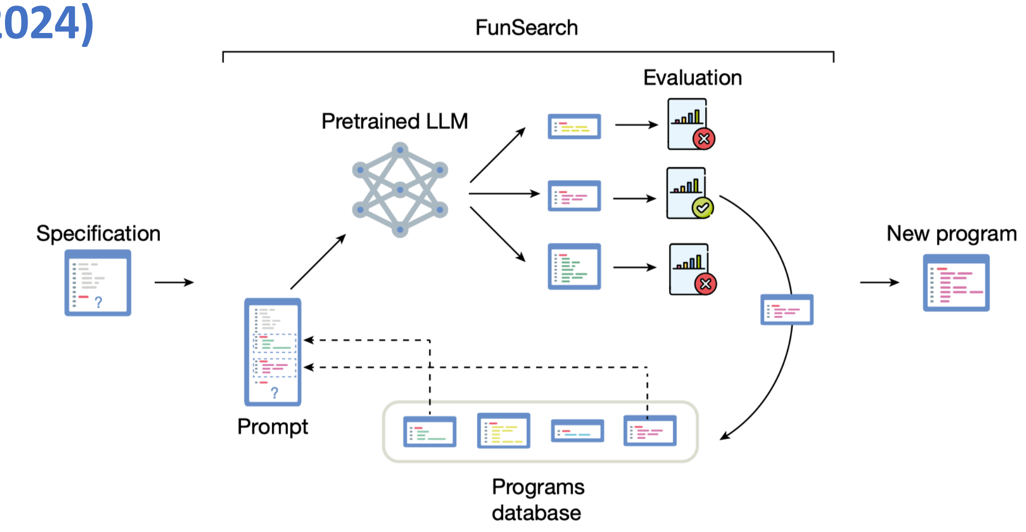
```js
input: "Improve this strategy: [parent_code_here]"
→ output: [new evolved code]
```



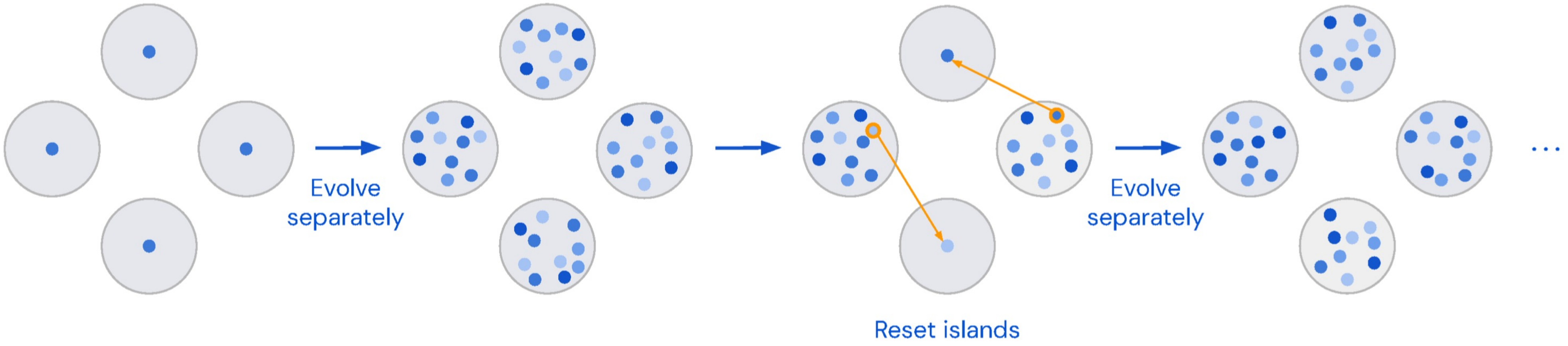
(a) Parent code + prompt → GPT → New offspring code

Advanced Method

- **[Nature] Mathematical discoveries from program search with large language models** (Romera-Paredes et al., 2024)
  - **Ranked Prompting for Evolution**
    - Show 3 programs: A > B > C
    - → LLM learns what "better" looks like
  - **Island Model**
    - Independent evolution of subgroups
    - → Encourages diversity, avoids local optima



(a) FunSearch framework[3]

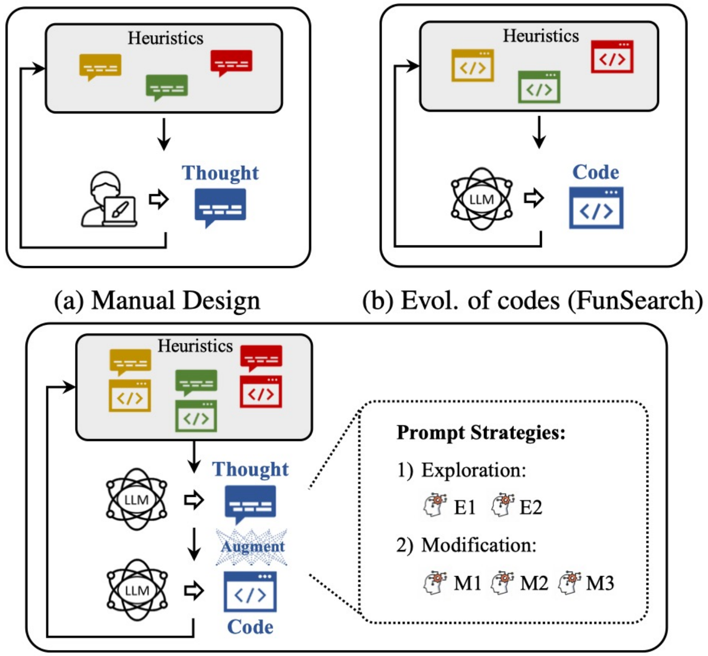


(b) Evolutionary Method with Island Model(Romera-Paredes et al., 2024)

Advanced Method

- **[ICML2024 Oral] Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model**
(Liu et al., 2024)

  - 🚀 **Key Innovation: Co-evolution of Code and Thought**
  - **Before-**Most prior work (e.g., FunSearch) evolves code only
  - **EOH's Idea**-Evolve both:
    - 🧠 Natural language description ("thought"): summarizes the high-level idea
    - 💻 Code: implements the details
  - **Benefit**
    - → Thought helps understanding and generalization
    - → Code offers executable precision



(a) Manual Design

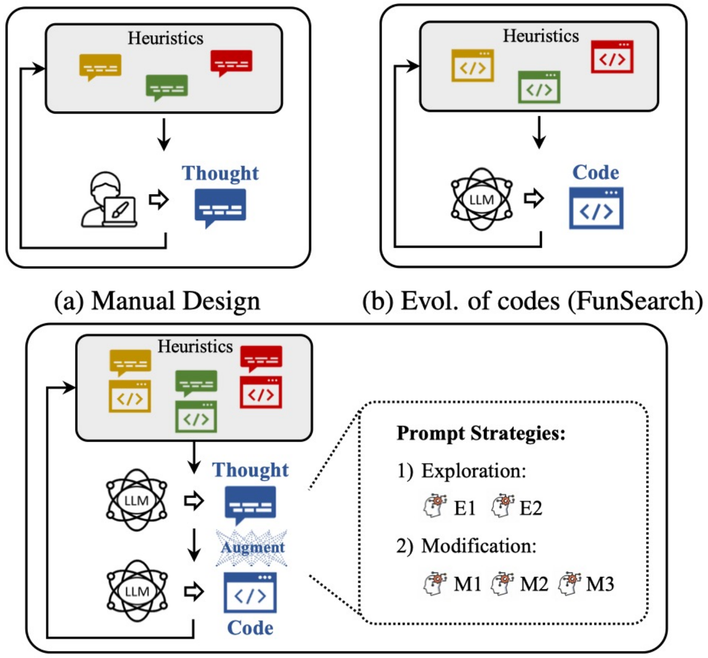
(b) Evol. of codes (FunSearch)

(a) EOH framework(Liu et al., 2024)

Advanced Method

- **[ICML2024 Oral] Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model**
  (Liu et al., 2024)

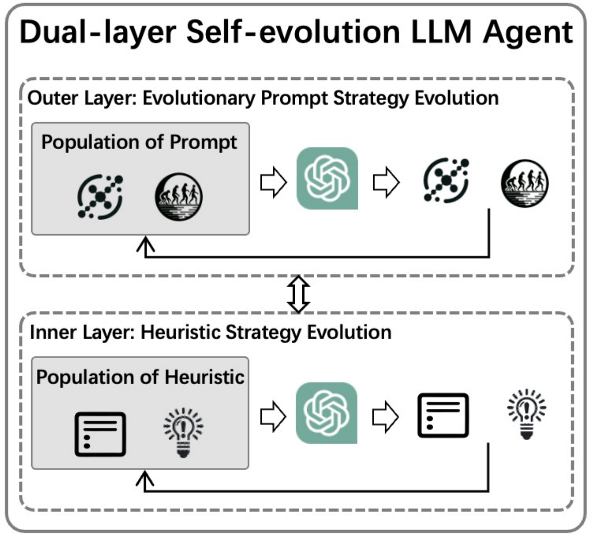
  - **Prompts' Key Idea:**
    - Use LLM as an evolutionary operator with carefully designed prompt strategies→ Mimic how humans generate new ideas
  - ✨ **Two Categories of Prompts:**
    - Exploration (E-series)
      - E1 – "Create something new"
      - E2 – "Same idea, new form"
    - Modification (M-series)
      - M1 – Improve it
      - M2 – Tune it
      - M3 – Simplify it



(a) EOH framework (Liu et al., 2024)

Advanced Method

- [ICML2025 Spotlight] Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems (Ye et al., 2025)
  - ❌ Problem in Prior Work
    - Prompt strategies are fixed or handcrafted
    - Evolution only happens at the strategy level
    - Leads to:
      - 🧬 Limited diversity in generated heuristics
      - 🌀 Easy stagnation in local optima
  - ✅ Key Innovation
    - Dual-layer Self-evolutionary Structure



(a) Dual-layer evolution

Layer	What It Evolves	Goal
🟠 **Outer Layer**	Prompt strategies *(how LLM evolves)*	**Exploration / Diversity**
🔵 **Inner Layer**	Heuristic strategies *(code + thought)*	**Exploitation / Convergence**

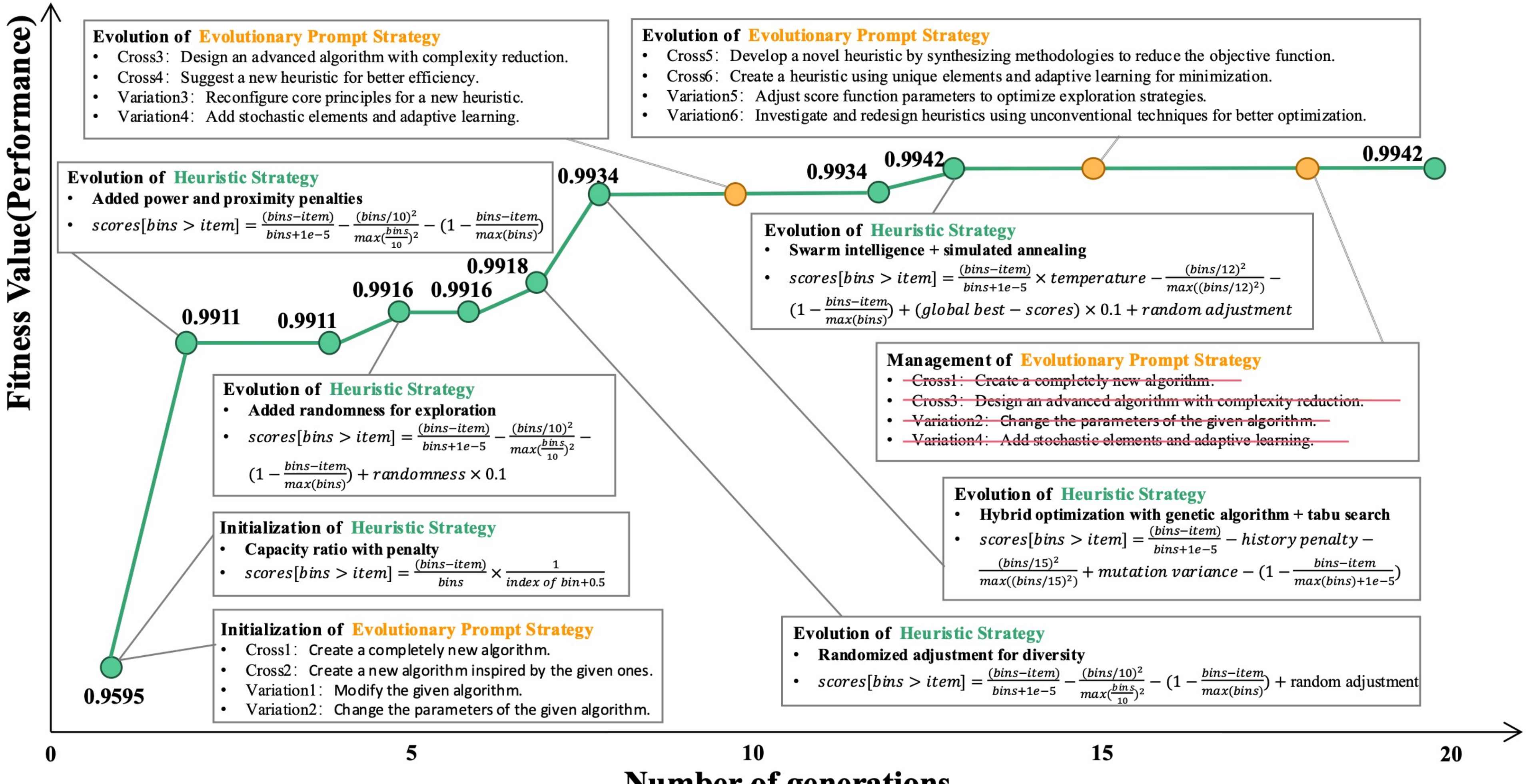
(b) Functional roles of the two layers in the self-evolutionary LLM agent

Advanced Method

- [ICML2025 Spotlight] Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems (Ye et al., 2025)
  - ❌ **Problem in Prior Work**
    - LLMs don't know which direction to evolve
    - Because previous generations only give them:
      - ✖ Few examples
      - ✖ No clear contrast between good and bad → Evolution is blind and directionless
  - ✅ **Key Innovation**
    - 🧠 Inspired by-Large Language Models as Optimizers (Yang et al., 2024)
      - LLMs can optimize without gradients just by seeing solution + score pairs and reasoning over natural language
    - **Differential Memory for Directional Evolution**
      - Provide the LLM with:
        - Multiple strategies + both their scores and ranks
        - Natural language thoughts
      - → So it can learn from differences and evolve better offspring strategies

Advanced Method

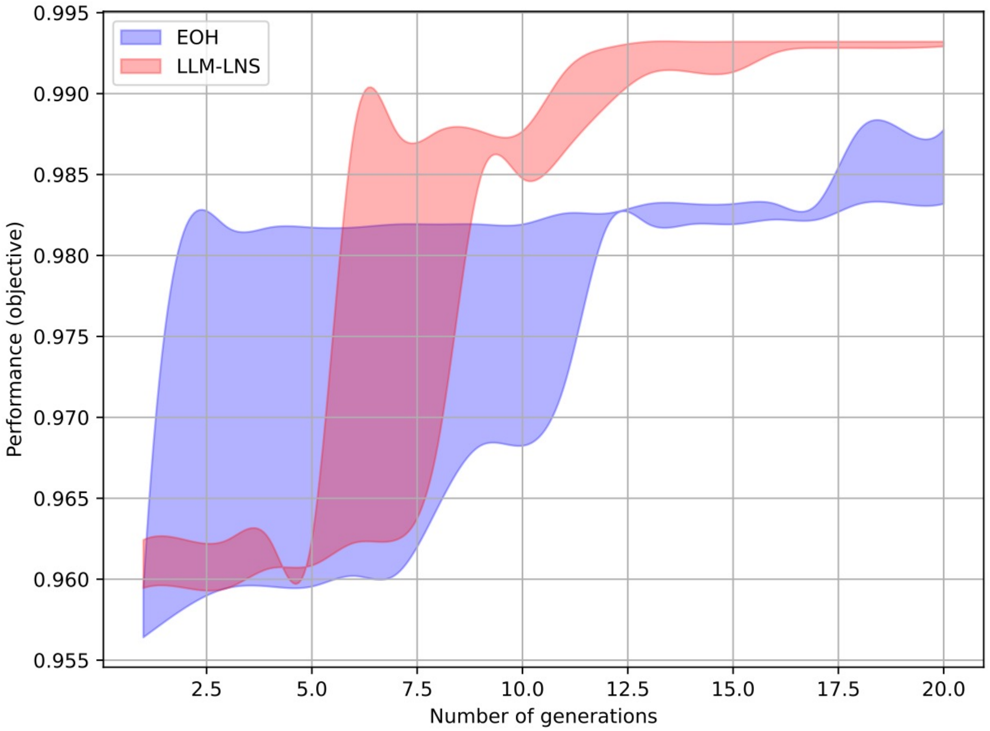
- [ICML2025 Spotlight] Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems (Ye et al., 2025)



(a) Evolution of Dual-layer Self-evolutionary LLM Agent for online bin packing (Ye et al., 2025)

Advanced Method

- [ICML2025 Spotlight] Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems (Ye et al., 2025)



(a) Evolutionary Progress of Heuristic Strategies in Online Bin Packing

Heuristic Designed by Dual-layer Self-evolution LLM Agent

Description

The new algorithm employs a hybrid optimization strategy that combines nonlinear penalties for historical usage, adaptive capacity scaling, and a relative size assessment, facilitating a balance between local and global search for optimal bin assignment.

Code

```
import numpy as np
def score(item, bins):
    feasible_bins = bins[bins > item]
    scores = np.zeros_like(bins)
    if len(feasible_bins) == 0:
        return scores
    # Nonlinear capacity scaling that enhances the desire for larger
       spaces
    remaining_capacity = feasible_bins - item
    capacity_scaling = np.log1p(remaining_capacity) * (
       remaining_capacity / np.max(remaining_capacity))
    # Relative size assessment: quadratic term comparing item size with
        bin capacities
    relative_size_effect = (item ** 2 / feasible_bins) * 50  # Scale to
        moderate impact
    # Nonlinear penalty based on historical usage counts to deter
       overutilization
    historical_count = np.arange(len(feasible_bins)) + 1  # Simulating
       historical usage
    penalty_factor = np.power(1.5, historical_count)  # Exponential
       penalty for higher usage
    # Combining scores: enhanced capacity scaling, moderated size
       assessment, and historical penalties
    scores[bins > item] = capacity_scaling - relative_size_effect -
       penalty_factor
    return scores
```

(b) Heuristic Designed by Dual-layer Self-evolution LLM Agent

Advanced Method

- [ICML2025 Spotlight] Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems (Ye et al., 2025)

Algorithm 2 Adaptive Large Neighborhood Search (ALNS)

Require: Initial solution $\mathbf{x}_0$, initial neighborhood size k, time limit T, threshold ϵ, iteration limit p, minimum and maximum neighborhood sizes $k_{\min}$, $k_{\max}$, decision variable count n, adjustment rate $u\%$ (percentage)

Initialize solution $\mathbf{x} \leftarrow \mathbf{x}_0$, set time $t \leftarrow 0$
while $t < T$ **do**
  Compute variable scores s_i using the LLM agent
  Select top-k variables to form neighborhood $\mathcal{N}$
  Solve subproblem within $\mathcal{N}$ using a solver
  Update solution $\mathbf{x}$ if an improvement is found
  if time spent in neighborhood exceeds predefined limit **then**
    $k \leftarrow \max(k_{\min}, k - \lceil u\% \cdot n \rceil)$
  else if improvement in objective value $< \epsilon$ for p consecutive iterations **then**
    $k \leftarrow \min(k_{\max}, k + \lceil u\% \cdot n \rceil)$
  end if
  Update time t
end while
return x

Heuristic (Obj Score: 5373.34904)
Develop a co-evolutionary heuristic approach that integrates genetic algorithms with local search techniques to enhance convergence speed and minimize the objective function for the specified problem.

Code

```
import numpy as np
def select_neighborhood(n, m, k, site, value, constraint,
      initial_solution, current_solution, objective_coefficient):
    neighbor_score = np.zeros(n)
    for i in range(m):
        lhs = sum(value[i][j] * current_solution[site[i][j]] for
            j in range(k[i]))
        for j in range(k[i]):
            var_index = site[i][j]
            difference = np.abs(current_solution[var_index] -
                initial_solution[var_index])
            neighbor_score[var_index] += (constraint[i] - lhs) *
                difference
    random_adjustment = np.random.rand(n)
    adaptive_mutation_rate = np.clip(np.abs(objective_coefficient
        ), 0.1, 1.0)
    neighbor_score += adaptive_mutation_rate * random_adjustment
    return neighbor_score
```

Advanced Method

- [ICML2025 Spotlight] Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems (Ye et al., 2025)

Table 4. **Comparison of objective values on large-scale MILP instances across different methods.** For each instance, the best-performing objective value is highlighted in bold. The - symbol indicates that the method was unable to generate samples for any instance within 30,000 seconds, while * indicates that the GNN&GBDT framework could not solve the MILP problem.

	SC_1(Min)	SC_2(Min)	MVC_1(Min)	MVC_2(Min)	MIS_1(Max)	MIS_2(Max)	$MIKS_1$(Max)	$MIKS_2$(Max)
Random-LNS	16140.6	169417.5	27031.4	276467.5	22892.9	223748.6	36011.0	351964.2
ACP	17672.1	182359.4	26877.2	274013.3	23058.0	226498.2	34190.8	332235.6
CL-LNS	-	-	31285.0	-	15000.0	-	-	-
Gurobi	17934.5	320240.4	28151.3	283555.8	21789.0	216591.3	32960.0	329642.4
SCIP	25191.2	385708.4	31275.4	491042.9	18649.9	9104.3	29974.7	168289.9
GNN&GBDT	16728.8	252797.2	27107.9	271777.2	22795.7	227006.4	*	*
Light-MILPOPT	16108.1	160015.5	26950.7	269571.5	22966.5	230432.9	36125.5	362265.1
LLM-LNS(Ours)	**15802.7**	**158878.9**	**26725.3**	**268033.7**	**23169.3**	**231636.9**	**36479.8**	**363749.5**

LLM-LNS achieves the best objective value on ALL tested large-scale instances, outperforming all baselines.

Table 7. Comparison of objective values on large-scale MILP instances across different methods using SCIP as optimizer. For each instance, the best-performing objective value is highlighted in bold. The - symbol indicates that the method was unable to generate samples for any instance within 30,000 seconds, while * indicates that the GNN&GBDT framework could not solve the MILP problem.

	SC_1	SC_2	MVC_1	MVC_2	MIS_1	MIS_2	$MIKS_1$	$MIKS_2$
Random-LNS	16164.2	171655.6	27049.6	277255.3	22892.9	222076.8	691.7	6870.1
ACP	17743.4	192791.2	27432.9	281862.4	23058.0	216008.8	29879.2	7913.5
CL-LNS	-	-	31285.0	-	15000.0	-	-	-
Gurobi	17934.5	320240.4	28151.3	283555.8	21789.0	216591.3	32960.0	329642.4
SCIP	25191.2	385708.4	31275.4	491042.9	18649.9	9104.3	29974.7	168289.9
GNN&GBDT	16728.8	261174.0	27107.9	271777.2	22795.7	227006.4	*	*
Light-MILPOPT	16147.2	166756.0	26956.8	269771.3	22963.6	230278.1	36125.5	**357483.8**
LLM-LNS(Ours)	**15950.2**	**161732.8**	**26763.4**	**268825.5**	**23137.19**	**230682.8**	**36147.7**	350468.7

Advanced Method

- [ICML2025 Spotlight] Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems (Ye et al., 2025)

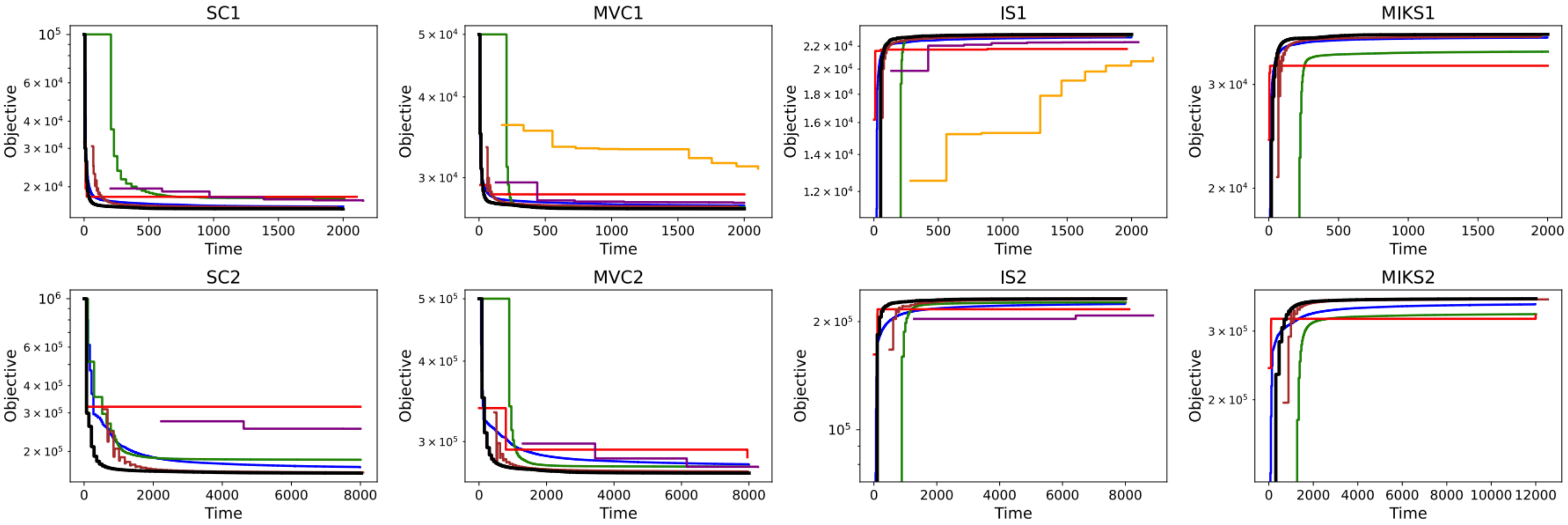


Table 5. **Comparison of primal integral on large-scale MILP instances across different methods.** For each instance, the best-performing primal integral is highlighted in bold.

	SC_1 (Min)	SC_2 (Min)	MVC_1 (Min)	MVC_2 (Min)	MIS_1 (Max)	MIS_2 (Max)	$MIKS_1$ (Max)	$MIKS_2$ (Max)
Random-LNS	3.41e+07	1.61e+09	5.50e+07	2.29e+09	4.47e+07	1.73e+09	7.08e+07	4.06e+09
ACP	5.43e+07	1.71e+09	5.90e+07	2.41e+09	4.11e+07	1.60e+09	6.02e+07	3.50e+09
CL-LNS	-	-	7.31e+07	-	3.17e+07	-	-	-
Gurobi	3.81e+07	2.65e+09	5.69e+07	2.36e+09	4.25e+07	1.75e+09	6.45e+07	3.86e+09
SCIP	5.18e+07	6.33e+09	6.39e+07	3.92e+09	3.34e+07	7.26e+07	5.77e+07	7.03e+08
GNN&GBDT	5.52e+07	3.75e+09	5.85e+07	2.55e+09	4.00e+07	1.40e+09	*	*
Light-MILPOPT	3.73e+07	1.80e+09	5.44e+07	2.27e+09	4.30e+07	1.70e+09	6.78e+07	4.03e+09
LLM-LNS(Ours)	**3.27e+07**	**1.38e+09**	**5.42e+07**	**2.19e+09**	**4.48e+07**	**1.81e+09**	**7.17e+07**	**4.18e+09**

Future Directions

- **Multimodal Optimization with LLMs**
  - Combine problem modeling and solution generation using multimodal inputs (e.g., text + code)
  - Enable end-to-end optimization pipelines via LLMs' multimodal understanding
- **Improving Diversity and Generalization**
  - Introduce continual learning mutation operators to adapt to changing problem spaces
  - Use performance-based feedback to evolve more effective mutation strategies
- **Modular Code Generation for Complex Logic**
  - Decompose complex logic into modular sub-tasks for better generation
  - Use interactive interfaces to guide LLMs and EAs in coordinated code generation

References

- **Meitei et al., 2020)** Meitei N Y, Baishnab K L, Trivedi G. 3D-IC partitioning method based on genetic algorithm[J]. IET Circuits, Devices & Systems, 2020, 14(7): 1104-1109.
- **(Dumas et al., 1991)** Dumas Y, Desrosiers J, Soumis F. The pickup and delivery problem with time windows[J]. European journal of operational research, 1991, 54(1): 7-22.
- **(Villa, 2001)** Villa A. Introducing some supply chain management problems[J]. International Journal of Production Economics, 2001, 73(1): 1-4.
- **(Achterberg, 2009)** Achterberg T. SCIP: solving constraint integer programs[J]. Mathematical Programming Computation, 2009, 1: 1-41.
- **(Biegler et al., 2009)** Biegler L T, Zavala V M. Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization[J]. Computers & Chemical Engineering, 2009, 33(3): 575-582.
- **(Pedroso, 2011)** Pedroso J P. Optimization with gurobi and python[J]. INESC Porto and Universidade do Porto, Porto, Portugal, 2011, 1.
- **(Bliek1ú et al., 2014)** Bliek1ú C, Bonami P, Lodi A. Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report[C]. Proceedings of the twenty-sixth RAMP symposium. 2014: 16-17.
- **(Morrison et al., 2016)** Morrison D R, Jacobson S H, Sauppe J J, et al. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning[J]. Discrete Optimization, 2016, 19: 79-102.
- **(Dey et al., 2018)** Dey S S, Molinaro M. Theoretical challenges towards cutting-plane selection[J]. Mathematical Programming, 2018, 170: 237-266.
- **(Song et al., 2020)** Song J, Yue Y, Dilkina B. A general large neighborhood search framework for solving integer linear programs[J]. Advances in Neural Information Processing Systems, 2020, 33: 20012-20023.
- **(Ye et al., 2023)** Ye H, Wang H, Xu H, et al. Adaptive constraint partition based optimization framework for large-scale integer linear programming (student abstract)[C]. Proceedings of the AAAI Conference on Artificial Intelligence. 2023, 37(13): 16376-16377.
- **(Liu et al., 2023)** Liu S, Lin Q, Li J, et al. A survey on learnable evolutionary algorithms for scalable multiobjective optimization[J]. IEEE Transactions on Evolutionary Computation, 2023, 27(6): 1941-1961.
- **(Yao et al., 2025)** Yao S, Wei J. The Second Half[EB/OL]. https://ysymyth.github.io/The-Second-Half/, accessed May 2, 2025.
- **(Wu et al., 2024)** Wu X, Wu S, Wu J, et al. Evolutionary computation in the era of large language model: Survey and roadmap[J]. IEEE Transactions on Evolutionary Computation, 2024.
- **(Romera-Paredes et al., 2024)** Romera-Paredes B, Barekatain M, Novikov A, et al. Mathematical discoveries from program search with large language models[J]. Nature, 2024, 625(7995): 468-475.
- **(Liu et al., 2024)** Liu F, Xialiang T, Yuan M, et al. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model[C]. International Conference on Machine Learning. PMLR, 2024: 32201-32223.
- **(Ye et al., 2025)** Ye H, Xu H, Yan A, et al. Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems[C]. International Conference on Machine Learning. 2025.
- **(Yang et al., 2024)** Yang C, Wang X, Lu Y, et al. Large Language Models as Optimizers[C]. The Twelfth International Conference on Learning Representations. 2024.