IEEE Congress on Evolutionary Computation
CEC 2025
Hangzhou

Large Language Model Driven Evolutionary Optimization

Hua Xu[1], Xiaodong Li[2], Yuan Yuan[3], Yuan Sun[4], Huigen Ye[1]

[1]Department of Computer Science&Technology, Tsinghua University

[2]School of Computing Technologies, RMIT University

[3]School of Computer Science and Engineering, Beihang University

[4]Research Center for Data Analytics and Cognition, La Trobe University

Tutorial at CEC2025, Hangzhou, China, June, 2025

Instructors

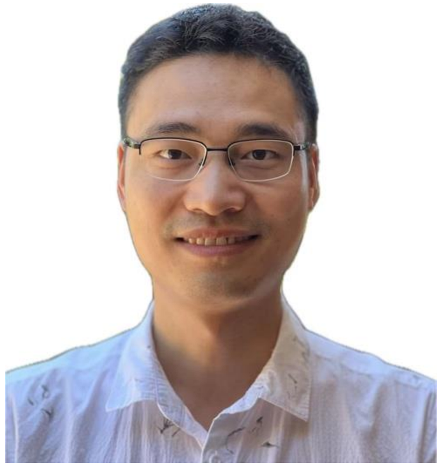
- **Hua Xu** is a tenured Associate Professor in the Department of Computer Science at Tsinghua University. His research explores intelligent optimization and human-machine interaction in AI. He has published extensively in top venues, authored influential monographs such as Intelligent Evolutionary Optimization (Elsevier, 2024), and holds numerous patents. His work has received major honors including the National Science and Technology Progress Award and the Beijing Science and Technology Award, and he currently serves as Editor-in-Chief of Intelligent Systems with Applications.

- **Xiaodong Li** is a Professor in the School of Computing Technologies at RMIT University, Melbourne. His research spans machine learning, evolutionary computation, swarm intelligence, and multiobjective optimization. He has served as Associate Editor for leading journals such as IEEE Transactions on Evolutionary Computation and Swarm Intelligence. A recipient of the ACM SIGEVO Impact Award and an IEEE Fellow, he has contributed extensively to the field through both research and leadership in IEEE CIS Task Forces.

Instructors

- **Yuan Yuan** is currently a Professor with the School of Computer Science and Engineering, Beihang University, China. His research interests include evolutionary computation, machine learning, multiobjective optimization and search-based software engineering. He has served as Associate Editor for leading journals such as IEEE Transactions on Evolutionary Computation and IEEE Transactions on Emerging Topics in Computational Intelligence.

- **Yuan Sun** is a Lecturer in Business Analytics and Artificial Intelligence at La Trobe University, Australia. He received his BSc in Applied Mathematics from Peking University, China, and his PhD in Computer Science from The University of Melbourne, Australia. His research interests include artificial intelligence, machine learning, operations research, and evolutionary computation. His research has contributed significantly to the emerging area of leveraging machine learning for combinatorial optimisation. He is the vice-chair of the IEEE task force on large-scale global optimisation and has organised special sessions and workshops, and delivered tutorials at the GECCO, PPSN, and CEC conferences.



Instructors

- **Huigen Ye** is a Ph.D. student at Tsinghua University, focusing on applying machine learning to accelerate large-scale optimization, particularly in mixed-integer programming. He has published papers in top conferences such as ICML, ICLR and AAAI. He is actively involved in academic service, serving as a reviewer for conferences like AISTATS, NeurIPS and ICLR.

Overview: Intelligent Evolutionary Optimization
- Guided by Deep Learning and Large Language Models

Hua Xu, Tsinghua University, Beijing, P. R. China

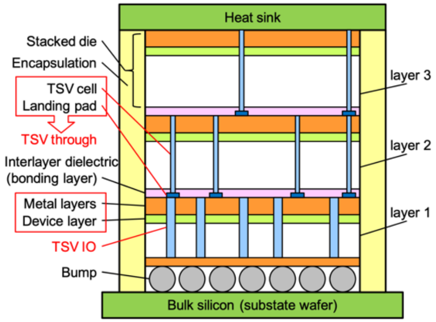
Contents

- Background
- Machine Learning Guided Evolutionary Optimization
- LLM Guided Evolutionary Optimization
- The Tutorial Organization

Background

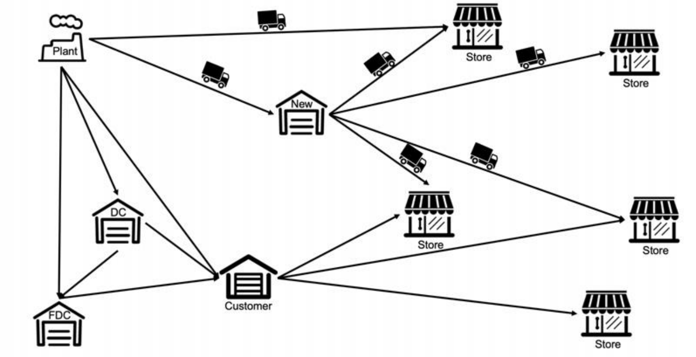
- **Several Optimization Cases**
  - 3D IC Partitioning Problem(Meitei et al., 2020)
  - Pickup and Delivery Problem with Time Windows(Dumas et al., 1991)
  - Supply Chain Management Problem(Villa, 2001)
- **Problem Definition**
  - Definition:

$$\begin{aligned} \min_{x \in \mathcal{X}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m, \\ & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned}$$



(a) IC Partitioning Problem

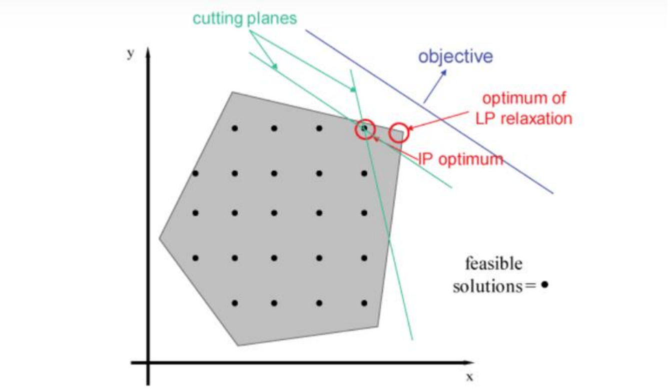
(b) Pickup and Delivery Problem



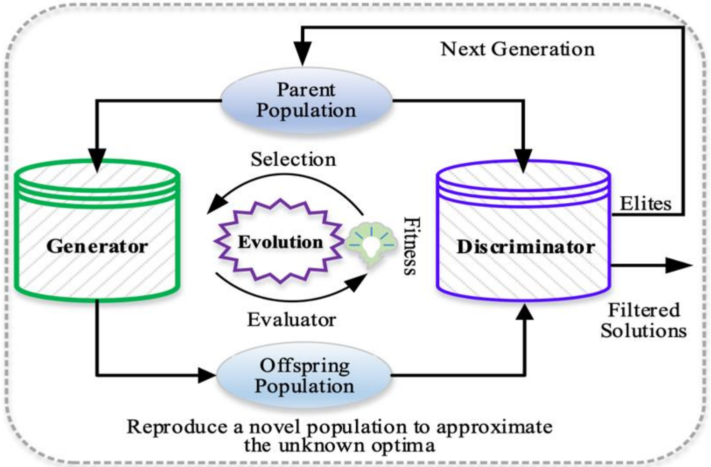
(c) Supply Chain Management

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)
    - **Evolution Optimization Method**(Liu et al., 2023)
- **Challenges**
  - **Exact Methods**: Scalability Issues & Exponential Complexity
  - **Heuristics Method:** Careful Parameter Tuning & Cold Start Issues



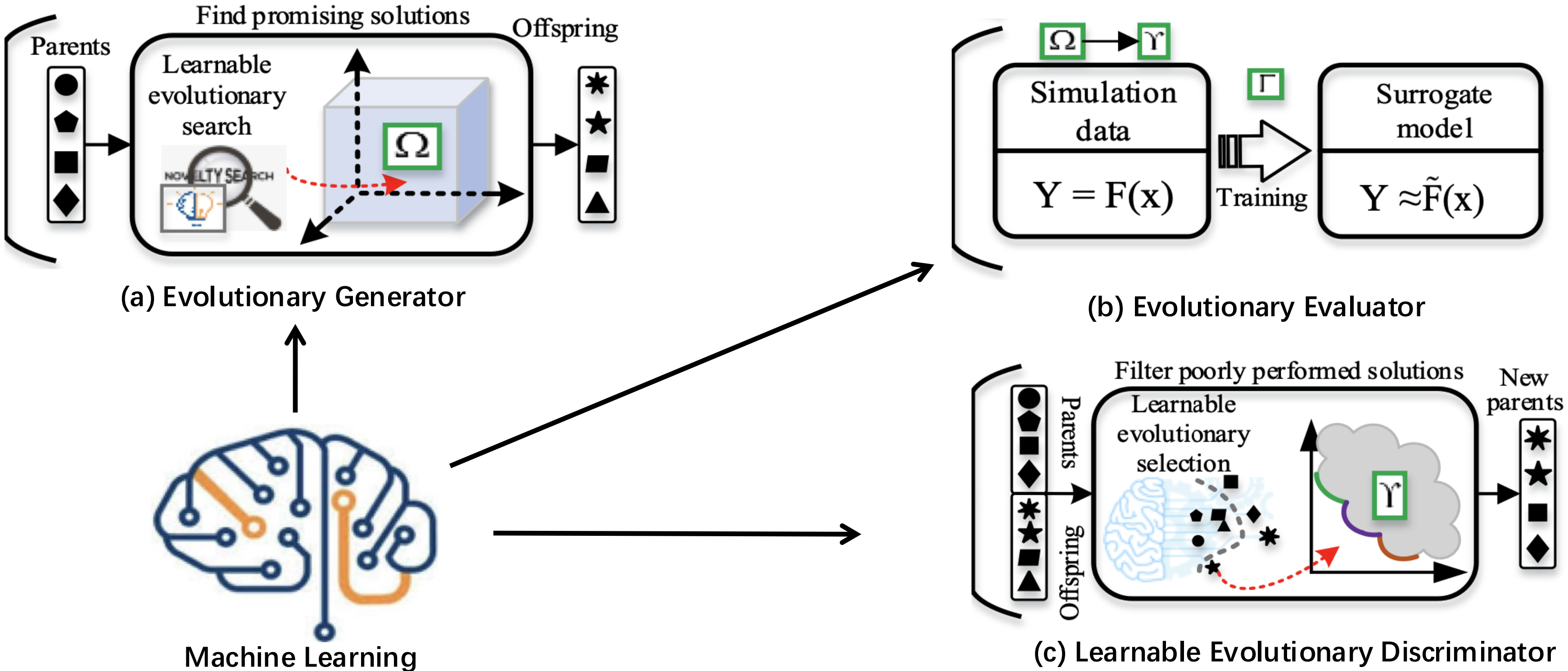
(a) Branch-and-Bound Techniques



(b) Evolution Optimization Method

Machine Learning Guided Evolutionary Optimization

- **Evolutionary processes assisted by machine learning**(Liu et al., 2023)
  - Evolutionary Generator
  - Evolutionary Evaluator
  - Learnable Evolutionary Discriminator
  - More details in **Topic I**



(a) Evolutionary Generator

(b) Evolutionary Evaluator

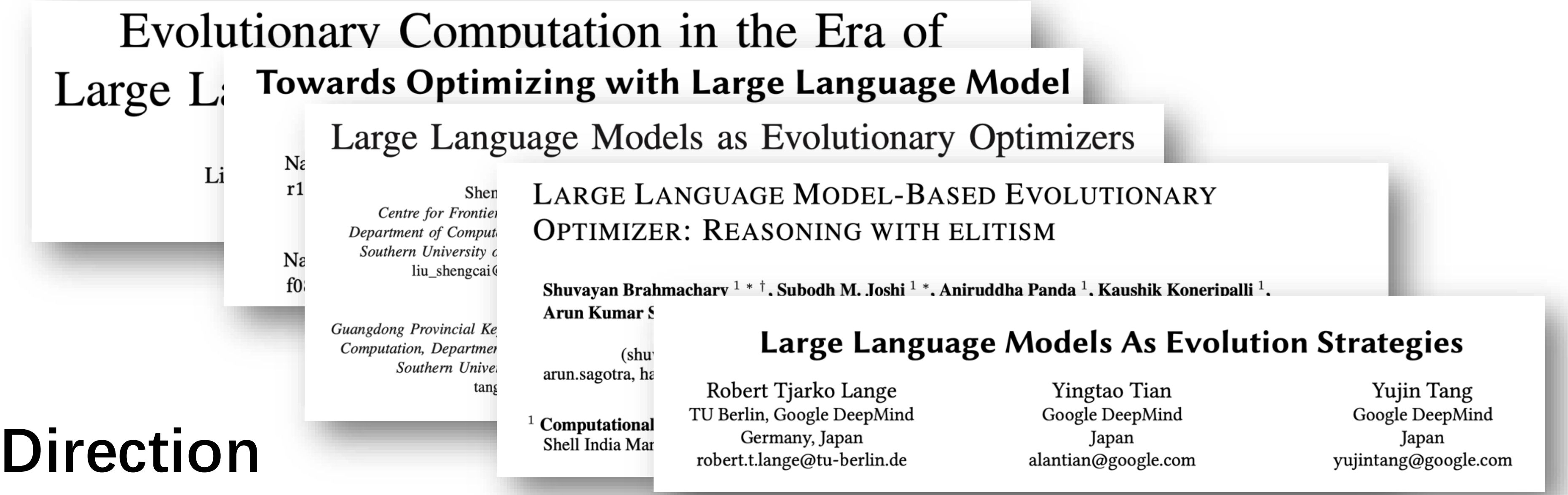
(c) Learnable Evolutionary Discriminator

Machine Learning Guided Evolutionary Optimization

- **Challenge**
  - Lack of diverse, large-scale training data for evolutionary learning
  - Current benchmarks are too small, simple, and fail to reflect real-world complexity
  - Traditional deep learning models are task-specific and require heavy retraining for each new optimization problem
- **What we need to do?**
  - Automated Data Generator(Yang et al., 2024)
  - Comprehensive Benchmark Test Suite(Ye et al., 2025a)
  - **Large Language Models (LLMs) to generalize and automate**

LLM Guided Evolutionary Optimization

- **Evolutionary processes assisted by LLM**



Evolutionary Computation in the Era of Large L

Li

Towards Optimizing with Large Language Model

Na
r1

Na
f0

Large Language Models as Evolutionary Optimizers

Shen
Centre for Frontie
Department of Comput
Southern University
liu_shengcai@

Guangdong Provincial Ke
Computation, Departme
Southern Unive
tang

LARGE LANGUAGE MODEL-BASED EVOLUTIONARY OPTIMIZER: REASONING WITH ELITISM

Shuvayan Brahmachary [1] * †, Subodh M. Joshi [1] *, Aniruddha Panda [1], Kaushik Koneripalli [1], Arun Kumar S

(shu
arun.sagotra, ha

[1] **Computational**
Shell India Mar

Large Language Models As Evolution Strategies

Robert Tjarko Lange
TU Berlin, Google DeepMind
Germany, Japan
robert.t.lange@tu-berlin.de

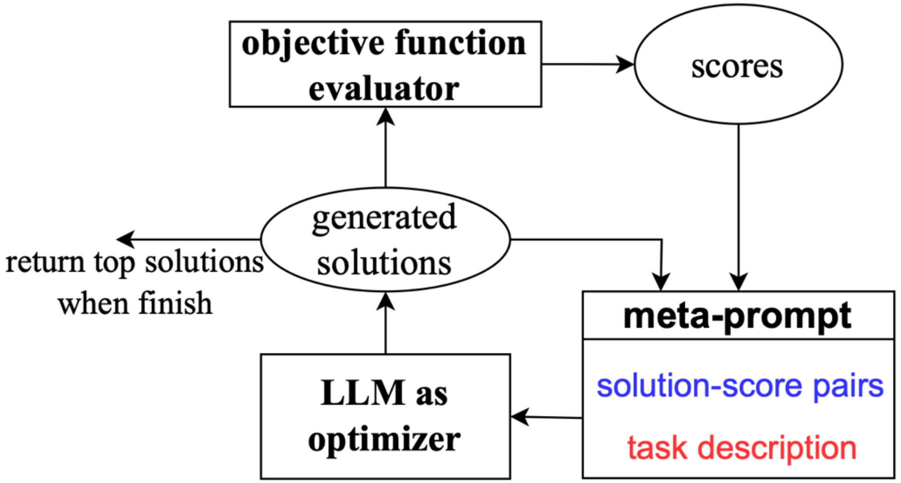
Yingtao Tian
Google DeepMind
Japan
alantian@google.com

Yujin Tang
Google DeepMind
Japan
yujintang@google.com

- **Key Direction**
  - LLM-assisted End-to-end Optimization
    - Automate both problem formulation and solving.
  - LLM-assisted Optimization Algorithm Generation
    - Generate heuristic operators for better search.
  - LLM-assisted Test Case Generation
    - Combine code synthesis with evolutionary search.

LLM Guided Evolutionary Optimization

- **LLM-assisted End-to-end Optimization**
  - LLMs show strong potential for black-box optimization.
    - **OPRO**(Yang et al., 2023): Iterative solution generation via optimization trajectories.
    - **LMEA**(Liu et al., 2024a): LLMs perform crossover and mutation in EA.
    - **LEO**(Brahmachary et al., 2025): Exploration and exploitation balanced through elitism.
  - More details in **Topic II**



(a) OPRO Framework

LLM Guided Evolutionary Optimization

- **LLM-assisted Optimization Algorithm Generation**
  - LLMs generate optimization algorithms beyond acting as operators.
    - Single-Round Generation:
      - **HybridMeta**(Pluhacek et al., 2023): LLMs design hybrid metaheuristics by combining known methods.
      - **Optimus**(AhmadiTeshnizi et al., 2023): LLMs automate MILP modeling, solving, and debugging.
    - Iterative Evolution:
      - **Funsearch**(Romera-Paredes et al., 2024): LLMs paired with evaluators to evolve interpretable programs solving combinatorial and algorithmic problems.
      - **EOH**(Liu et al., 2024b): LLMs co-evolve heuristic ideas and code structures.
      - **Reevo**(Ye et al., 2024): Reflective evolution enhances algorithm optimization via short- and long-term feedback.
  - More details in **Topic III**

LLM Guided Evolutionary Optimization

- **LLM-assisted Test Case Generation**
  - LLMs enhance test generation by combining code synthesis with evolutionary search.
    - **TitanFuzz**(Deng et al., 2023)
      - Uses Codex (for seeds) and InCoder (for mutations).
      - Fitness: dataflow depth, API diversity, repeated API penalties.
    - **CodaMOSA**(Lemieux et al., 2023)
      - EA halts at coverage plateau.
      - LLM generates tests for low-coverage functions.
      - EA resumes with improved seeds.
  - More details in **Topic IV**

The Tutorial Organization

- **Topic I**: Evolutionary Optimization Based on Machine Learning

  Prof. Xiaodong Li, IEEE Fellow, RMIT University

- **Topic II:** Business Optimization and Problem Formulation Using Large Language Models

  Prof. Yuan Sun, La Trobe University

- **Topic III:** Evolutionary Optimization Guided by Large Models

  Huigeng Ye, Tsinghua University

- **Topic IV:** Test Case Generation Using Large Language Models

  Prof. Yuan Yuan, Beihang University

References

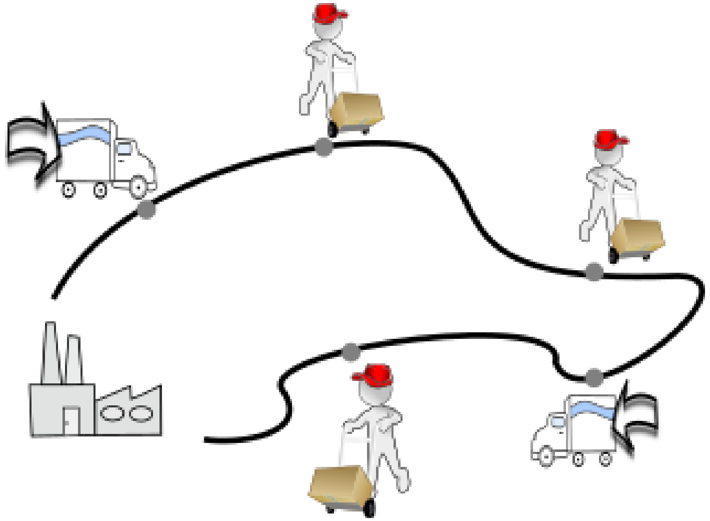
- **(Meitei et al., 2020)** Meitei N Y, Baishnab K L, Trivedi G. 3D-IC partitioning method based on genetic algorithm[J]. IET Circuits, Devices & Systems, 2020, 14(7): 1104-1109.
- **(Dumas et al., 1991)** Dumas Y, Desrosiers J, Soumis F. The pickup and delivery problem with time windows[J]. European 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.
- **(Yang et al., 2024)** Yang T, Ye H, Xu H. Learning to generate scalable milp instances[C].Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2024: 159-162.
- **(Ye et al., 2025a)** Ye H, Cheng Y, Xu H, et al. MILPBench: A Large-scale Benchmark Test Suite for Mixed Integer Linear Programming Problems[C]. Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2025.
- **(Yang et al., 2023)** Yang C, Wang X, Lu Y, et al. Large Language Models as Optimizers[C]. The Twelfth International Conference on Learning Representations.
- **(Liu et al., 2024a)** Liu S, Chen C, Qu X, et al. Large language models as evolutionary optimizers[C]. 2024 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2024: 1-8.
- **(Brahmachary et al., 2025)** Brahmachary S, Joshi S M, Panda A, et al. Large language model-based evolutionary optimizer: Reasoning with elitism[J]. Neurocomputing, 2025, 622: 129272.
- **(Pluhacek et al., 2023)** Pluhacek M, Kazikova A, Kadavy T, et al. Leveraging large language models for the generation of novel metaheuristic optimization algorithms[C].Proceedings of the Companion Conference on Genetic and Evolutionary Computation. 2023: 1812-1820.
- **(AhmadiTeshnizi et al., 2023)** AhmadiTeshnizi A, Gao W, Udell M. Optimus: Optimization modeling using mip solvers and large language models[J]. arXiv preprint arXiv:2310.06116, 2023.
- **(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., 2024b)** Liu F, Xialiang T, Yuan M, et al. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model[C]. Forty-first International Conference on Machine Learning.
- **(Ye et al., 2024)** Ye H, Wang J, Cao Z, et al. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution[C]. The Thirty-eighth Annual Conference on Neural Information Processing Systems.
- **(Deng et al., 2023)** Deng Y, Xia C S, Peng H, et al. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models[C]. Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis. 2023: 423-435.
- **(Lemieux et al., 2023)** Lemieux C, Inala J P, Lahiri S K, et al. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models[C]. 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023: 919-931.

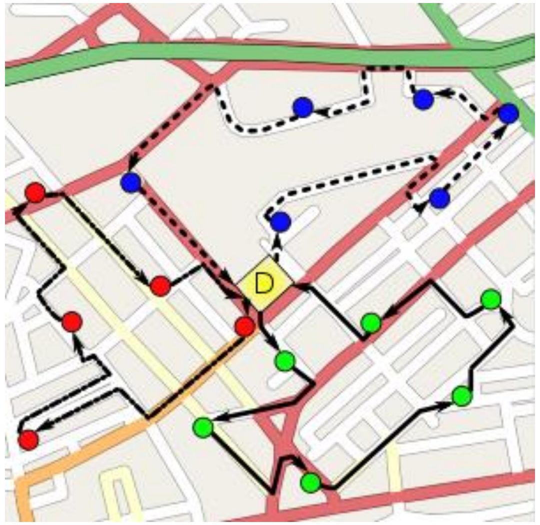
Evolutionary Optimization Enhanced by Machine Learning

Xiaodong Li, RMIT University, Melbourne, Victoria, Australia
Yuan Sun (Presenter), La Trobe University, Melbourne, Australia

Why machine learning can help?

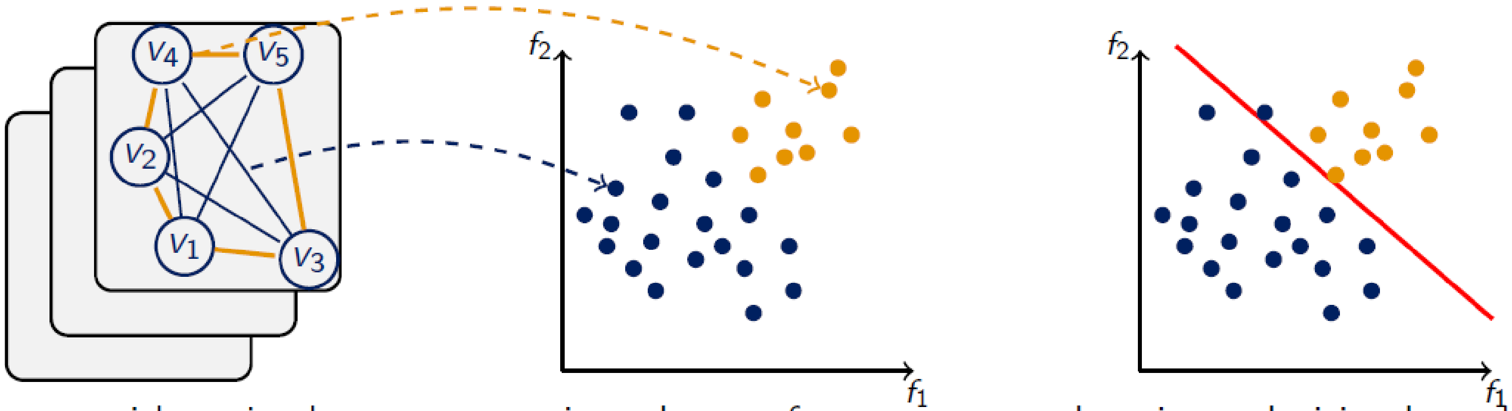
- Traditional optimization often relies on handcrafted rules for decision-making.
- Minor variations in problems frequently require redeveloping algorithms.
- In many industries, similar instances are solved repeatedly, often from scratch.
- There is an abundance of data, e.g., historical operations data collected over time.
- Collecting optimal solutions for training is feasible due to advanced solvers.
- Machine learning and deep learning are mature, powerful, and widely accessible.





Solution Prediction via Machine Learning: Training

- Solve a set of easy problem instances to optimality.
- Label decision variables using their optimal solution values.
- Extract features to characterize each decision variable.
- Train a machine learning model to predict optimal values for decision variables.



instances with optimal routes   mapping edges to feature space   learning a decision boundary

Features

- A list of features that may be useful:
  - Graph-based features: e.g., node weight, node degree, edge distance
  - Mathematical features: e.g., objective bound (sum of weights of a node and its neighbors for maximum weighted clique problems)
  - MIP formulation features: e.g., cost coefficients, number of non-zeros in the constraint matrix etc.
  - LP relaxation features: e.g., reduced costs, optimal LP solution values
  - Heuristic-based features: e.g., value-to-weight ratio in knapsack problems
  - Statistical features from sample solutions: e.g., correlation between variable values and objective value, frequency of variable values in high-quality solutions (similar to Estimation of Distribution Algorithms or Ant Colony Optimization).

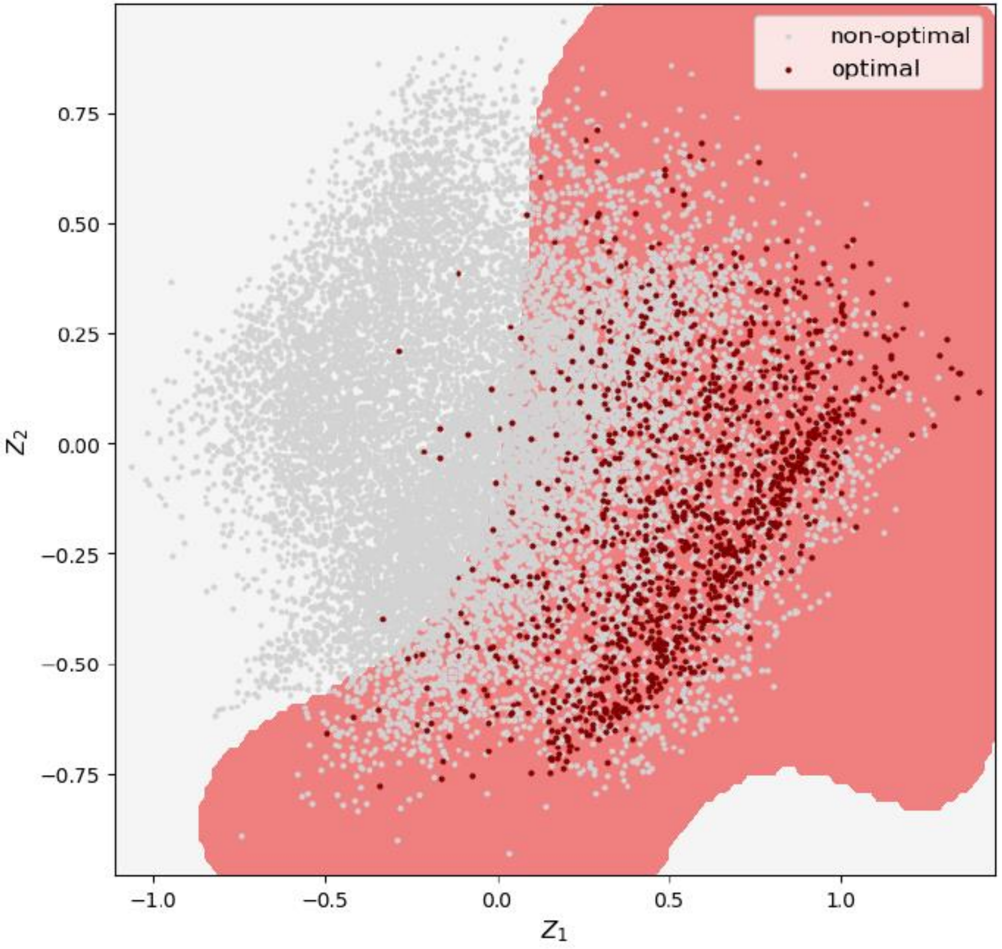
Machine learning models

Using Pre-defined Features

- Logistic Regression
- Support Vector Machines
- Deep Neural Networks

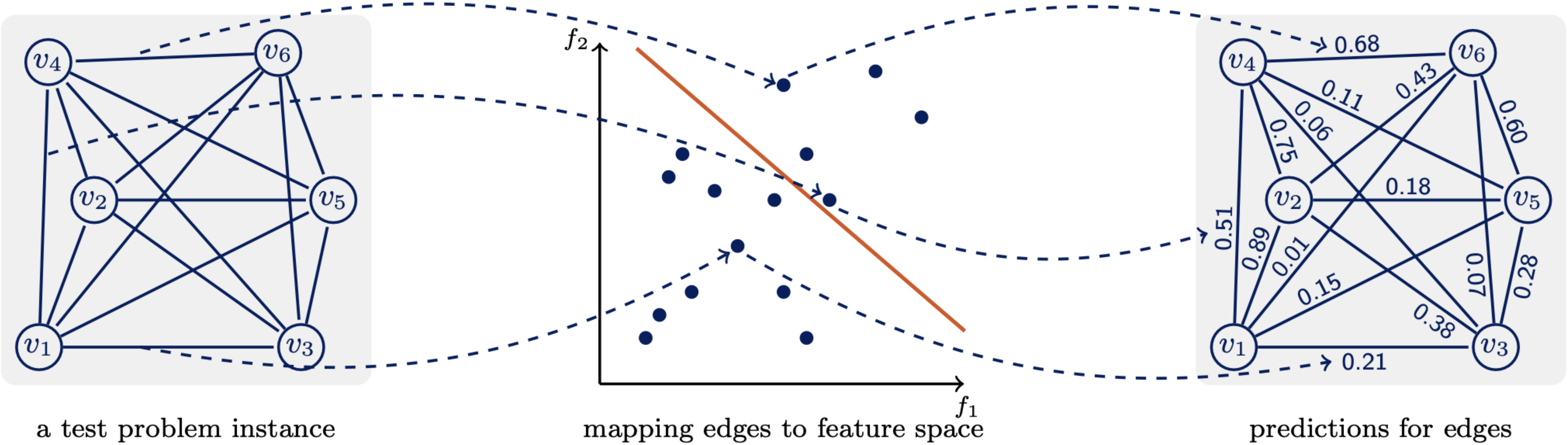
Automatically Learning Features

- Graph Neural Networks (GNNs): Leverage graph structure to learn node/edge-level representations
- Autoencoders: Learn compact, informative feature representations from raw inputs
- Transformers: Capture long-range dependencies, useful for sequences or sets



Solution Prediction via Machine Learning: Testing

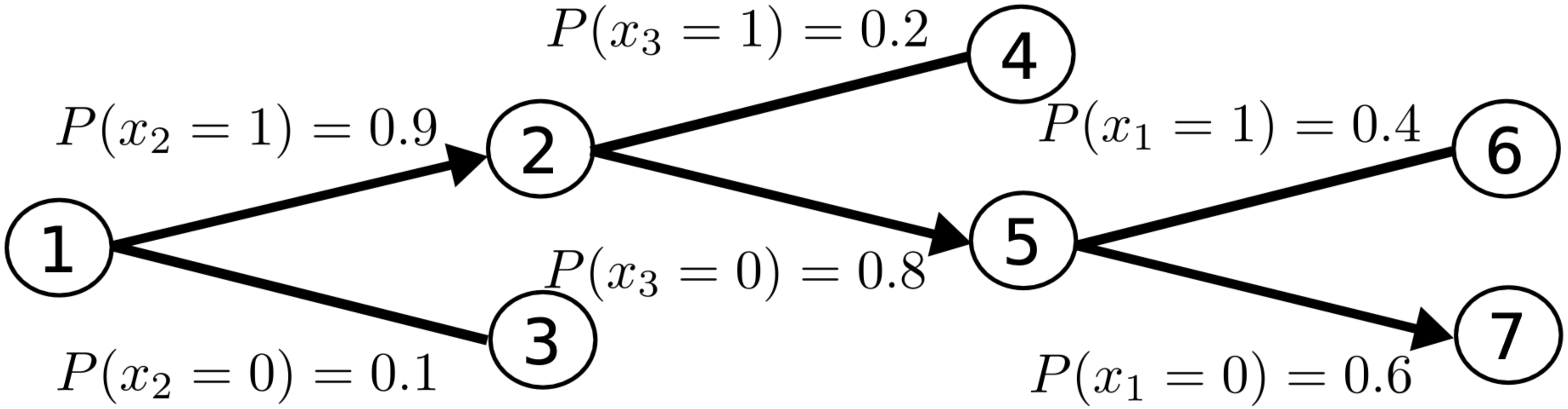
- Given a test problem instance, the trained machine learning model predicts the likelihood that each binary decision variable is part of the optimal solution (i.e., has a value of 1).



•Note: Any bounded integer variable can be represented as a set of binary variables, making this approach broadly applicable.

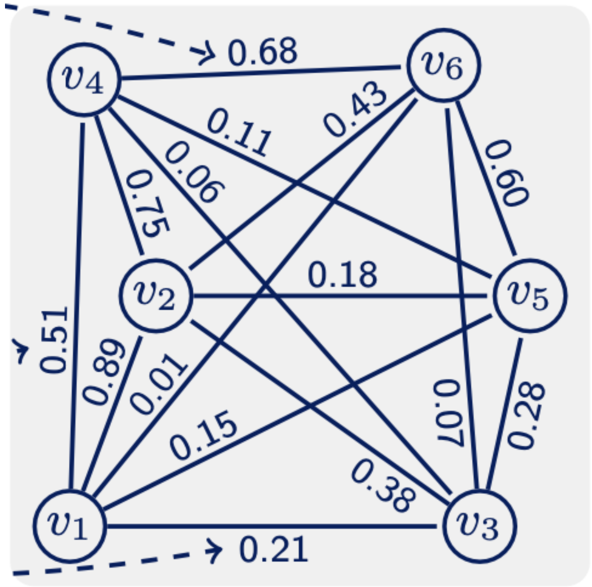
Constructing Solutions from ML Predictions

- Use predicted probabilities to greedily construct solutions, selecting at each step the variable most likely to be in the optimal solution.
- Expected to outperform hand-crafted heuristics, since such rules can be incorporated as features into the learning model.
- Can be combined with tree-based search methods (e.g., DFS) to prioritize high-quality regions of the solution space. (**NeurIPS'18, IJCNN'21**)(Li et al., 2018, Shen et al., 2021)

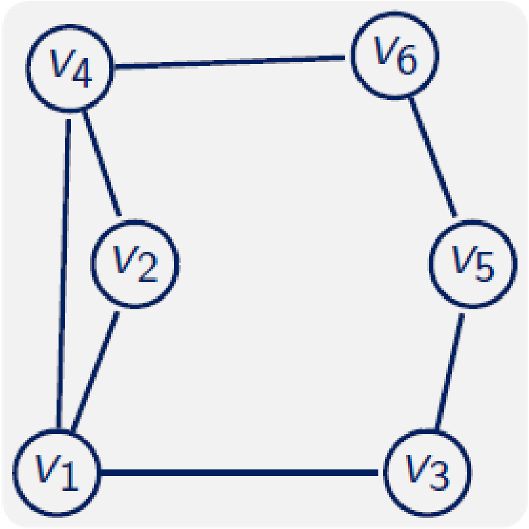


Pruning the Search Space with ML Predictions

- Fix or remove decision variables that are unlikely to be part of the optimal solution. Specifically, variables with predicted probabilities below a threshold can be fixed to 0. (**TPMAI'19, AAAI'19**)(Sun et al., 2019, Lauri et al., 2019)
- Apply a search algorithm to find a solution in the resulting reduced problem space, enabling faster and more focused optimization.



predictions for edges



removing low-quality edges

Extensions of ML-Guided Pruning

•Constraint-Based Pruning (AAAI 2020)(Ding et al., 2020)

- Train a GNN to predict binary variable values in MIPs. Prune the search space via a global inequality constraint that limits deviation from the predicted solution.

•Generalization to Unseen Instances (OR Spectrum 2021)(Sun et al., 2021)

- Train ML models on one category of instances and test across a variety of instances with different characteristics.

•Multi-Stage Pruning (Journal of Heuristics, 2023)(Lauri et al., 2023)

- Apply the trained ML model recursively to prune the search space. Each stage trains a new classifier to progressively eliminate harder-to-prune elements.

•ML-Guided Column Generation (ICLR 2023)(Sun et al., 2022a)

- Extend to problem formulations with exponentially many variables. Use the model to filter and select high-quality variables.

•Reduce-Then-Optimize (Transportation Science, 2025)(Spieckermann et al., 2025)

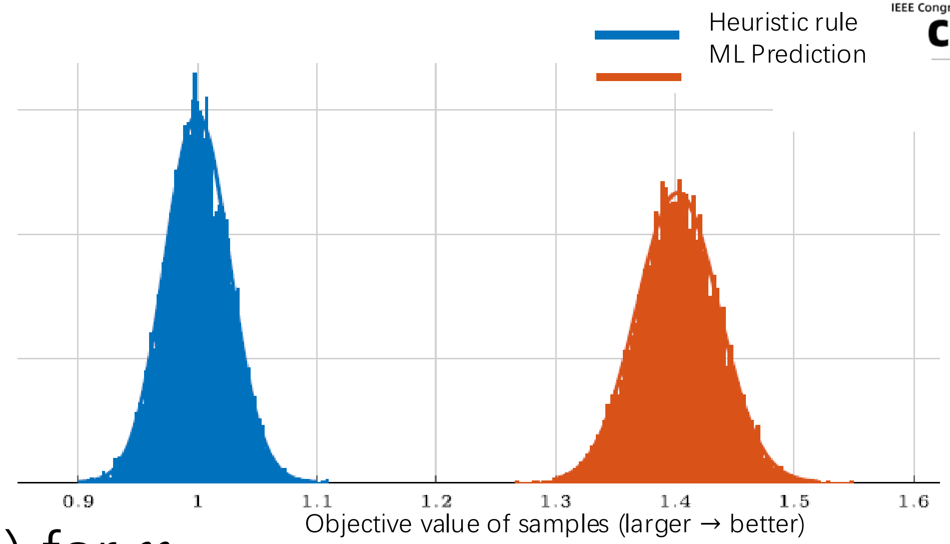
- Use a GNN to identify a relevant subset of variables in the Fixed-Charge Transportation Problem (FCTP), reducing problem size and boosting solver efficiency.

Sampling Solutions Based on ML Predictions

At each step, the probability of selecting variable v_i is:

$$p_i = \frac{y_i}{\sum_{j \in S} y_j}$$

where:



- y_i is the ML prediction (likelihood of inclusion) for v_i
- S is the set of feasible candidate variables that can be added to the solution.

Sampling solutions for the pricing problem in Column Generation (AAAI'22)(Shen et al., 2022)

- A diverse set of high-quality solutions is required — not just one optimal solution.
- Compared to traditional sampling, this approach yields better-quality columns;
- Compared to exact or heuristic methods, it generates more high-quality solutions.

Boosting ACO with ML Predictions

Ant Colony Optimization (ACO) is a probabilistic algorithm that samples solutions using:

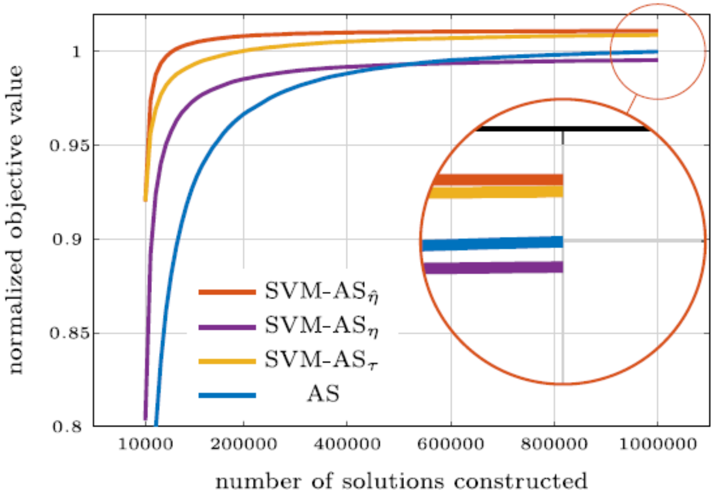
$$p_i = \frac{\tau_i \eta_i}{\sum_{j \in S} \tau_j \eta_j}$$

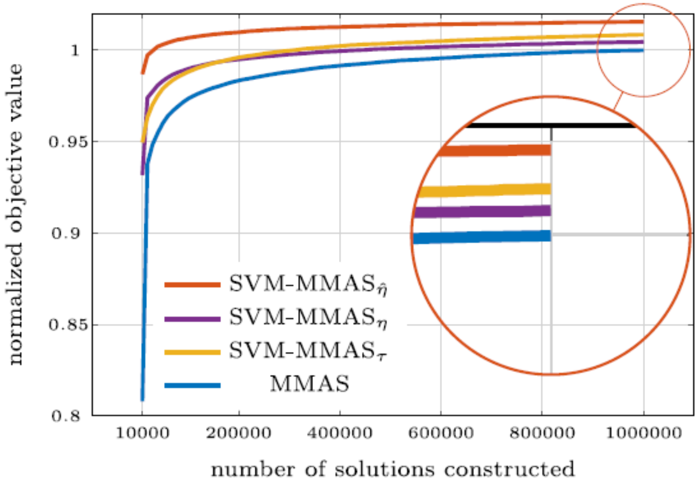
- $\boldsymbol{\eta}$: is heuristic rule,
- $\boldsymbol{\tau}$: pheromone trial reflecting the “evolved” quality of solution components.

Typically, $\boldsymbol{\tau}$ is initialized uniformly and $\boldsymbol{\eta}$ is set based on domain-specific heuristic rules.

ML-Enhanced Variants of ACO (COR 2022)(Sun et al., 2022a)

- SVM-ACO$_\eta$: Set $\eta_i = y_i$, where y_i is the ML predicted probability;
- SVM-ACOτ: Initialize $\tau_i = y_i$ using ML predictions;
- SVM-ACO$_{\hat{\eta}}$: Set the $\boldsymbol{\eta}$ value as a combination of ML predictions and a heuristic rule





Extensions of ML-Enhanced ACO

DeepACO (NeurIPS’23)(Ye et al., 2023)

- Use deep reinforcement learning to learn heuristic measures and is evaluated on eight combinatorial optimization problems.

LN-ACO (GECCO’23)(Liu et al., 2023)

- Use an “intelligent ant” with a pre-trained GNN to predict variable selection probabilities, forming a hybrid colony with traditional ants to guide the search process.

DLQ-ACO (GECCO’23)(Ramírez et al., 2023)

- Use GNN to generate variable selection probabilities and Q-learning to decide during solution construction whether to use ML-derived probabilities or traditional ACO heuristics.

ML-ACO for Column Generation (GECCO’24)(Xu et al., 2024)

- Incorporate ML predictions into the heuristic measures of ACO to efficiently generate multiple diverse, high-quality solutions for Column Generation.

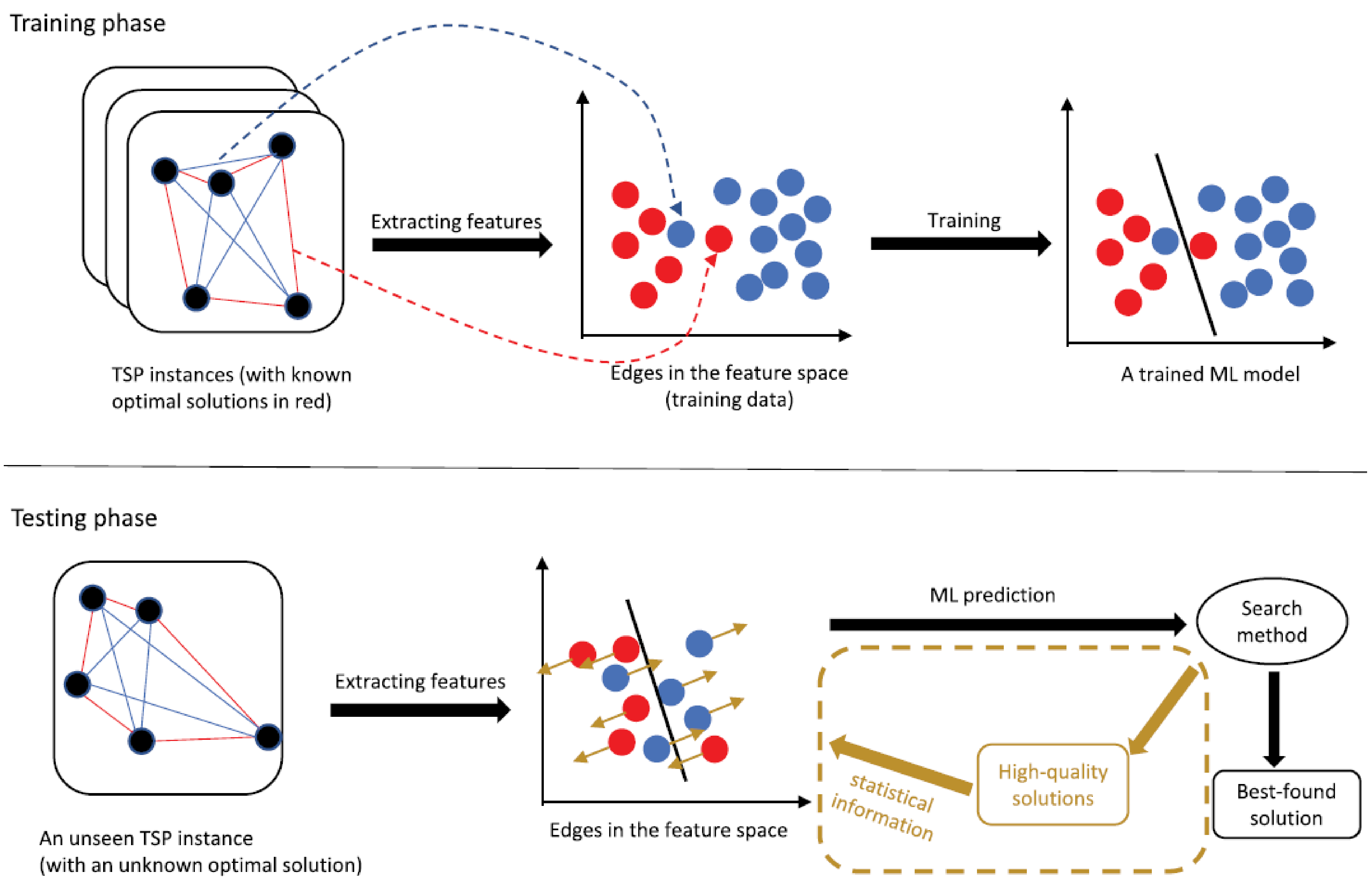
GFACS (AISTATS’25)(Kim et al., 2025)

- Uses Generative Flow Networks to learn a multi-modal prior distribution to set the heuristic matrix for ACO, evaluated on seven combinatorial optimization problems.

Adaptive Solution Prediction via Machine Learning

- **Limitations of Offline-Trained ML Models**: Accuracy is often limited by
  - Static, one-shot predictions that do not adapt during the search process.
  - Challenges in crafting features that effectively represent decision variables.
  - Distribution shift between training and testing instances.
- **Adaptive Solution Prediction (ASP) (EJOR 2023)**(Shen et al., 2023)
  - Refines ML predictions iteratively during the search on a problem instance.
  - Incorporates feedback from search to progressively enhance prediction.
  - Use statistical features extracted from sampled solutions to update predictions over time.
  - Examples of statistical features include:
    - Correlation between variable values and objective value
    - Frequency of variables used in high-quality solutions (e.g., pheromone update in ACO)
  - As the search generates better solutions, statistical features evolve, leading to more informed and accurate predictions.

Adaptive Solution Prediction via Machine Learning



Relationship Between ASP and EDAs

Similarities

- Both sample the solution space of a specific instance.
- Both update a probability distribution to guide toward better solutions.
- Both exhibit online learning through iterative refinement.

Differences

- ASP uses an offline-trained ML model with knowledge from historical instances;
- EDAs (Estimation of Distribution Algorithms) typically learn from scratch during each run.
- ASP generalizes EDAs:

  -For example, ACO updates probabilistic mode (pheromone) using fixed rules.

  -ASP builds its prediction model from data using multiple features (can include ACO-style rules).

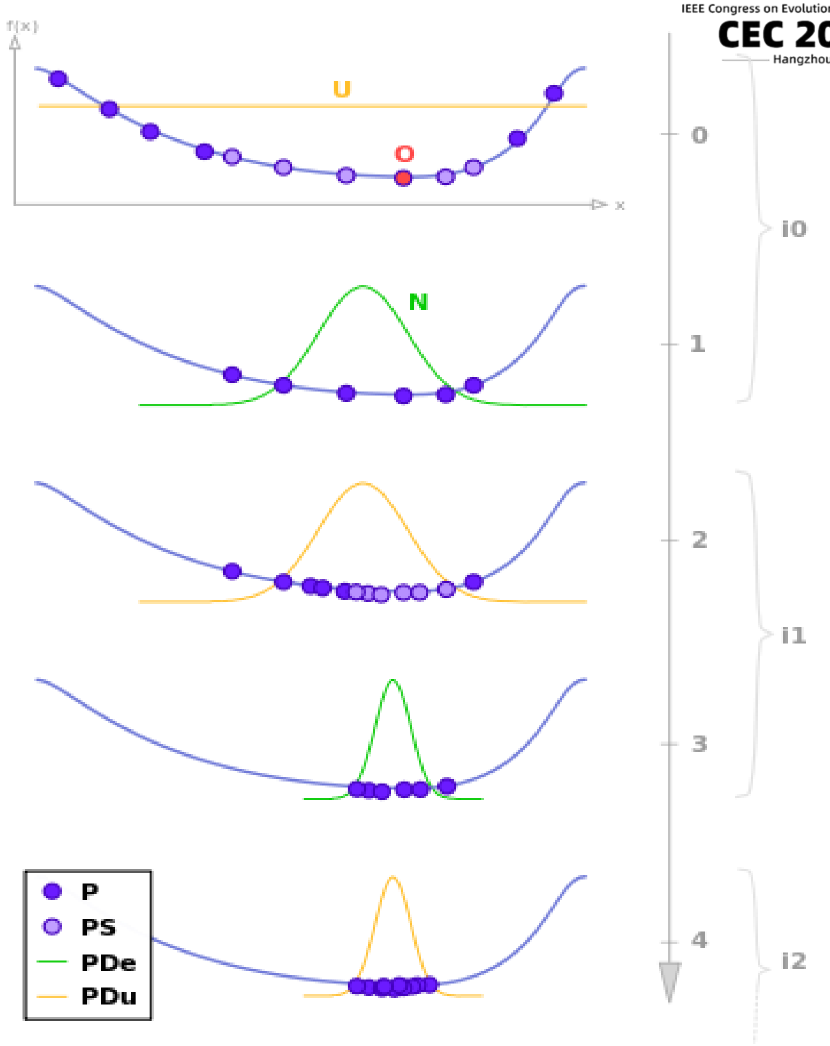


Figure: EDA builds and sample an explicit probabilistic model from a pool of promising candidate solutions (source: Wikipedia).

Relationship Between ASP and ML-ACO

Similarities

- Both integrate machine learning into metaheuristic search for combinatorial optimization.
- Both use ML predictions to guide solution construction or sampling.
- Both allow for adaptive behavior during problem-solving.

Differences

Aspect	ML-ACO	ASP
Integration Point	Injects ML into ACO components (pheromone, heuristic)	Predicts variable values to guide solution construction
Adaptivity	Often static once ML predictions are embedded	Dynamically refined using feedback from ongoing search
Learning Mode	Uses ML to improve ACO's components offline	Uses offline ML + online statistical adaptation
Generalization	Enhances specific ACO variants (e.g., SVM-ACO)	General framework applicable beyond ACO

Challenges and Future Directions

- 🚧 Key Challenges
  - **Generalization Across Problems:** Building ML models that work across a class of problems or broader MILP remains difficult.
  - **Feasibility & Guarantees:** ML predictions may violate constraints or yield infeasible solutions; providing optimality gap guarantee is challenging.
  - **Refining Predictions:** Most ML models produce static, one-shot predictions. Online refinement and instance-specific adaptation are underexplored.
- 🚀 Future Directions
  - **Expanding to Other Algorithms & Problems:** Apply ML to enhance e.g., PSO, GA, or handle dynamic and multi-objective optimization problems.
  - **Cross-Domain Generalization:** Develop generic ML models using meta-learning, instance-space analysis, or domain-agnostic feature representations.
  - **Adaptive Learning:** Move beyond ASP to integrate feedback (and possibly re-training) during search for better adaptability and ML predictions.
  - **Exploring New ML Paradigms:** Leverage LLMs for heuristic code generation and MILP model formulation (see next sections).

References

- **(Li et al., 2018)** Li, Z., Chen, Q., & Koltun, V. (2018). Combinatorial optimization with graph convolutional networks and guided tree search. Advances in Neural Information Processing Systems.
- **(Shen et al., 2021)** Shen, Y., Sun, Y., Eberhard, A., & Li, X. (2021, July). Learning primal heuristics for mixed integer programs. In 2021 International Joint Conference on Neural Networks (IJCNN) IEEE.
- **(Sun et al., 2019)** Sun, Y., Li, X., & Ernst, A. (2019). Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(5), 1746-1760.
- **(Lauri et al., 2019)** Lauri, J., & Dutta, S. (2019, July). Fine-grained search space classification for hard enumeration variants of subset problems. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, No. 01, pp. 2314-2321).
- **(Ding et al., 2020)** Ding, J. Y., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., & Song, L. (2020, April). Accelerating primal solution findings for mixed integer programs based on solution prediction. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 34, No. 02, pp. 1452-1459).
- **(Sun et al., 2021)** Sun, Y., Ernst, A., Li, X., & Weiner, J. (2021). Generalization of machine learning for problem reduction: a case study on travelling salesman problems. OR Spectrum, 43(3), 607-633.
- **(Lauri et al., 2023)** Lauri, J., Dutta, S., Grassia, M., & Ajwani, D. (2023). Learning fine-grained search space pruning and heuristics for combinatorial optimization. Journal of Heuristics, 29(2), 313-347.
- **(Sun et al., 2022a)** Sun, Y., Ernst, A. T., Li, X., & Weiner, J. (2022). Learning to generate columns with application to vertex coloring. In The Eleventh International Conference on Learning Representations (ICLR).
- **(Spieckermann et al., 2025)** Spieckermann, C., Minner, S., & Schiffer, M. (2025). Reduce-then-Optimize for the Fixed-Charge Transportation Problem. Transportation Science, 59(3), 540-564.
- **(Shen et al., 2022)** Shen, Y., Sun, Y., Li, X., Eberhard, A., & Ernst, A. (2022, June). Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 36, No. 9, pp. 9926-9934).
- **(Sun et al., 2022b)** Sun, Y., Wang, S., Shen, Y., Li, X., Ernst, A. T., & Kirley, M. (2022). Boosting ant colony optimization via solution prediction and machine learning. Computers & Operations Research, 143, 105769.
- **(Ye et al., 2023)** Ye, H., Wang, J., Cao, Z., Liang, H., & Li, Y. (2023). DeepACO: Neural-enhanced ant systems for combinatorial optimization. Advances in Neural Information Processing Systems (NeurIPS), 36, 43706-43728.
- **(Liu et al., 2023)** Liu, Y., Qiu, J., Hart, E., Yu, Y., Gan, Z., & Li, W. (2023, July). Learning-based neural ant colony optimization. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) (pp. 47-55).
- **(Ramírez et al., 2023)** Ramírez Sánchez, J. E., Chacón Sartori, C., & Blum, C. (2023, July). Q-Learning ant colony optimization supported by deep learning for target set selection. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) (pp. 357-366).
- **(Xu et al., 2024)** Xu, H., Shen, Y., Sun, Y., & Li, X. (2024, July). Machine Learning-Enhanced Ant Colony Optimization for Column Generation. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) (pp. 1073-1081).
- **(Kim et al., 2025)** Kim, M., Choi, S., Kim, H., Son, J., Park, J., & Bengio, Y. Ant Colony Sampling with GFlowNets for Combinatorial Optimization. In The 28th International Conference on Artificial Intelligence and Statistics (AISTATS).
- **(Shen et al., 2023)** Shen, Y., Sun, Y., Li, X., Eberhard, A., & Ernst, A. (2023). Adaptive solution prediction for combinatorial optimization. European Journal of Operational Research, 309(3), 1392-1408.

Two PhD Scholarships Available

- **Research Area**
- Enhancing Combinatorial Optimisation with Machine Learning

- **Scholarship Details**
- Stipend: AUD $35,886 per annum (tax-free)
- Duration: 3.5 years
- Tuition: Full tuition fee waiver
- Location: RMIT University (Collaborate with Monash & La Trobe), Melbourne

- **Contact:**

✉ Prof Xiaodong Li – xiaodong.li@rmit.edu.au
✉ Dr Yuan Sun – yuan.sun@latrobe.edu.au

Business Optimization and Problem Formulation Using Large Language Models

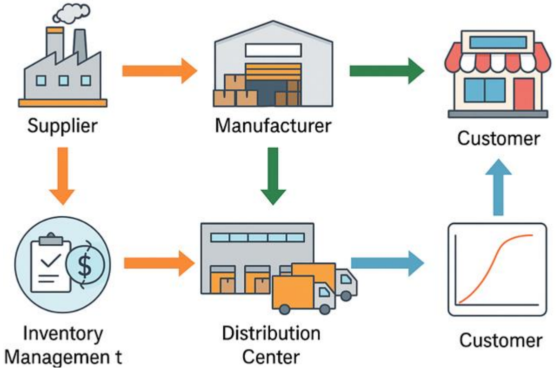
Yuan Sun, La Trobe University, Melbourne, Australia

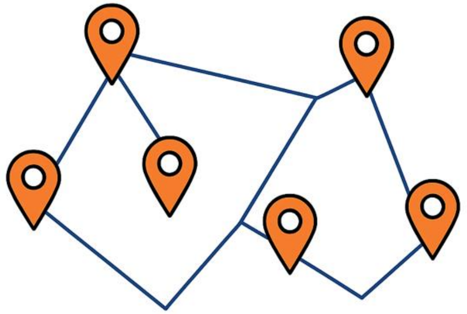
Business Optimization

Using mathematical models and analytical techniques to enhance decision-making, improve efficiency, reduce costs, and maximize profitability for a business.

Typical Business Optimization Problems:

- ➢ Supply Chain Optimization
- ➢ Job Shop Scheduling
- ➢ Inventory Management
- ➢ Vehicle Routing Problem
- ➢ Production Planning
- ➢ Portfolio Optimization
- ➢ Staff Scheduling





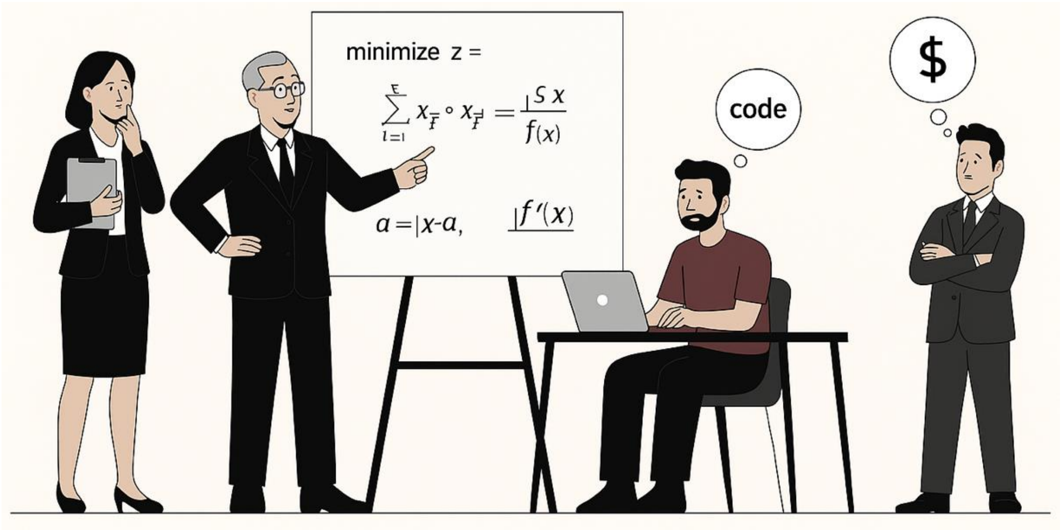
Example

- I am a retailer selling laptops.
- Currently, there are 1015 laptops in the store.
- Each laptop in inventory incurs a cost of \$1.0 per week.
- The forecast demand for laptops at the store in the next 8 weeks are {⋯}.
- Ordering laptops incurs a fixed cost of \$2000 per order with up to 6000 laptops.
- What is the best plan to order laptops for the next 8 weeks to minimise the inventory and logistic costs while satisfying demand?

Challenges

Traditional way of applying optimization to solve business problems:

- Requires expert knowledge to translate natural language into solvable models
- Experts must have strong programming skills
- Interpretation of results and turning them into business actions is non-trivial



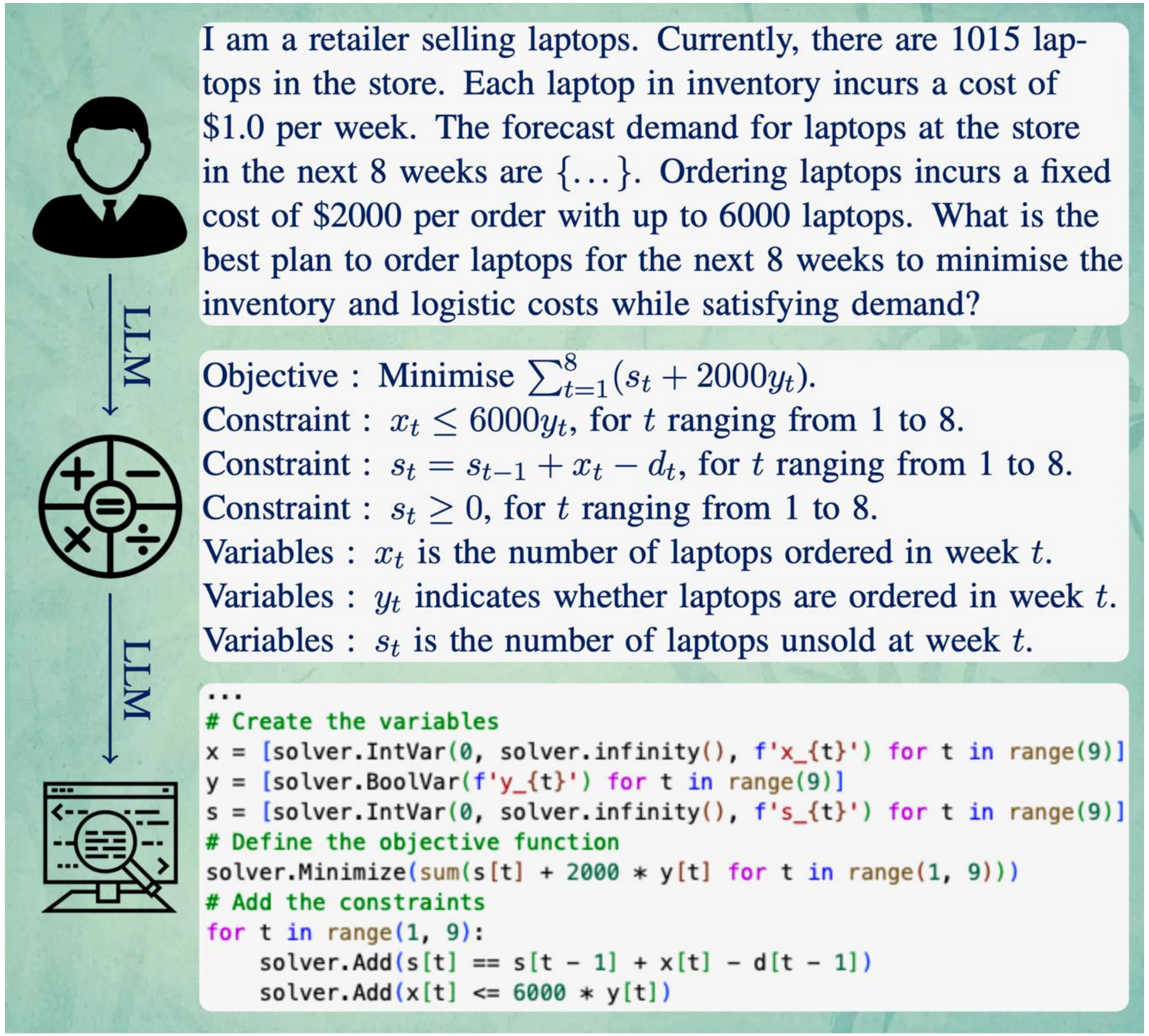
- ➢ Expertise is scarce, expensive, and often specialized to one technique.
- ➢ Optimization process is time-consuming even for skilled professionals.

Automating Problem Formulation Using LLMs

Aim to use LLMs to automatically convert natural language descriptions of business problems into:

- Mathematical models
- Computer programs

This automation reduces reliance on expert knowledge and speeds up the optimization process (Ramamonjison et al., 2022)



I am a retailer selling laptops. Currently, there are 1015 laptops in the store. Each laptop in inventory incurs a cost of \$1.0 per week. The forecast demand for laptops at the store in the next 8 weeks are $\{\dots\}$. Ordering laptops incurs a fixed cost of \$2000 per order with up to 6000 laptops. What is the best plan to order laptops for the next 8 weeks to minimise the inventory and logistic costs while satisfying demand?

Objective : Minimise $\sum_{t=1}^{8}(s_t + 2000y_t)$.
Constraint : $x_t \leq 6000y_t$, for t ranging from 1 to 8.
Constraint : $s_t = s_{t-1} + x_t - d_t$, for t ranging from 1 to 8.
Constraint : $s_t \geq 0$, for t ranging from 1 to 8.
Variables : x_t is the number of laptops ordered in week t.
Variables : y_t indicates whether laptops are ordered in week t.
Variables : s_t is the number of laptops unsold at week t.

```
...
# Create the variables
x = [solver.IntVar(0, solver.infinity(), f'x_{t}') for t in range(9)]
y = [solver.BoolVar(f'y_{t}') for t in range(9)]
s = [solver.IntVar(0, solver.infinity(), f's_{t}') for t in range(9)]
# Define the objective function
solver.Minimize(sum(s[t] + 2000 * y[t] for t in range(1, 9)))
# Add the constraints
for t in range(1, 9):
    solver.Add(s[t] == s[t - 1] + x[t] - d[t - 1])
    solver.Add(x[t] <= 6000 * y[t])
```

Early Efforts and Examples

🎓 Academic:

- Stanford University
- University of Cambridge
- Zhejiang University
- Chinese University of Hong Kong

Augmenting Operations Research with Auto-Formulation of Optimization Models from Problem Descriptions

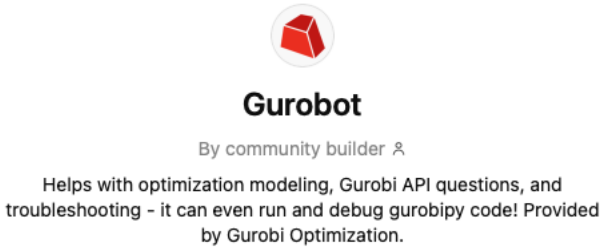
Rindranirina Ramamonjison, Haley Li, Timothy T. Yu, Shiqi He, et al.
EMNLP 2022

OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models

Ali Ahmaditeshnizi, Wenzhi Gao, Madeleine Udell
ICML 2024

🏢 Industry:

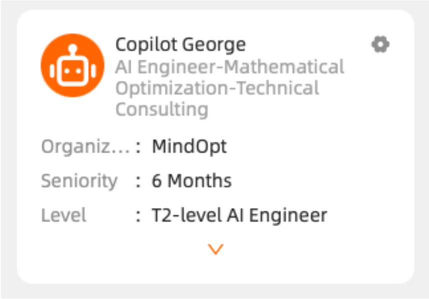
- Huawei
- Gurobi
- Alibaba
- Microsoft



Gurobot

By community builder

Helps with optimization modeling, Gurobi API questions, and troubleshooting - it can even run and debug gurobipy code! Provided by Gurobi Optimization.



Copilot George
AI Engineer-Mathematical Optimization-Technical Consulting

Organiz... : MindOpt
Seniority : 6 Months
Level : T2-level AI Engineer

Method Overview

🧠 Prompt-Based Techniques

- Use carefully crafted prompts to guide LLMs in translating natural language into optimization models.

🔄 Prompt Optimization

- Iteratively refine prompts based on model output and feedback to improve formulation accuracy.

🔧 Fine-Tuning Approaches

- Train LLMs on domain-specific datasets (fully or efficiently) to enhance performance and consistency.

🤖 Workflow Architectures

- Structure the problem-solving process using single-agent or multi-agent systems for modular reasoning.

🔍 Retrieval-Augmented Generation (RAG)

- Dynamically fetch and insert relevant past examples or context to improve task relevance and accuracy.

Zero-Shot Prompting

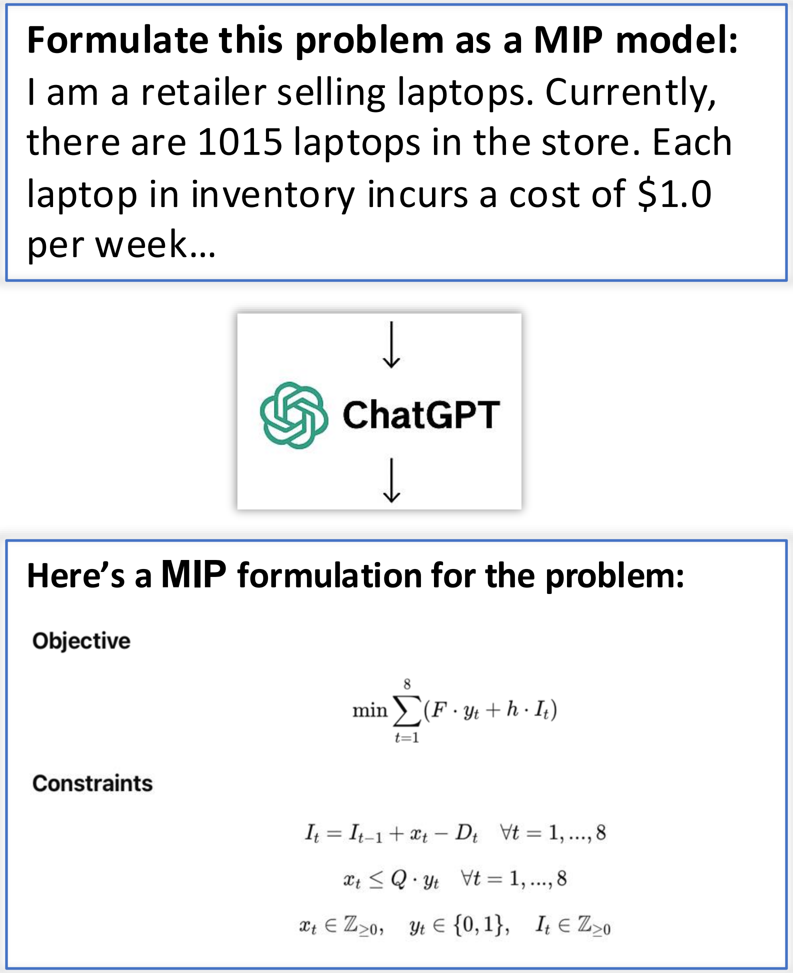
Use a LLM to formulate an optimization model from natural language with **no** examples(Ahmed et al., 2024)

✅ Strengths:

- Easy to use and implement
- No labelled data or fine-tuning
- Compatible with commercial LLMs

⚠ Limitations:

- Accuracy may drop on complex tasks
- Highly sensitive to prompt phrasing



Formulate this problem as a MIP model:
I am a retailer selling laptops. Currently, there are 1015 laptops in the store. Each laptop in inventory incurs a cost of $1.0 per week…

Here's a MIP formulation for the problem:

Objective

$$\min \sum_{t=1}^{8} (F \cdot y_t + h \cdot I_t)$$

Constraints

$$I_t = I_{t-1} + x_t - D_t \quad \forall t = 1, ..., 8$$

$$x_t \leq Q \cdot y_t \quad \forall t = 1, ..., 8$$

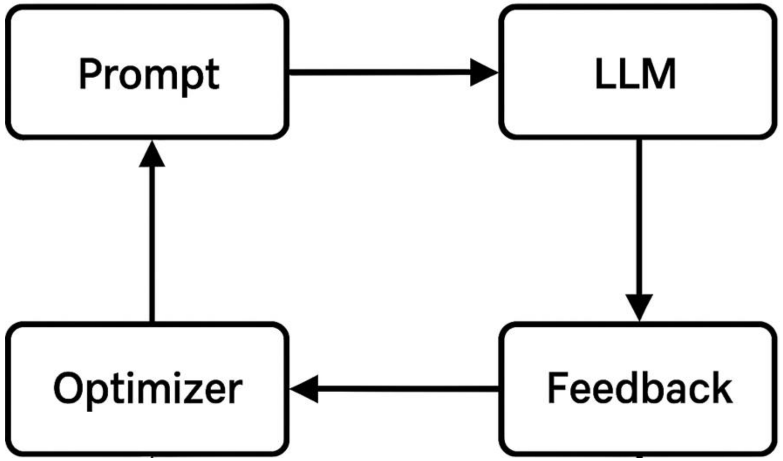
$$x_t \in \mathbb{Z}_{\geq 0}, \quad y_t \in \{0, 1\}, \quad I_t \in \mathbb{Z}_{\geq 0}$$

Prompt Optimization

- The process of automatically improving prompts to maximize LLM performance on a task.
- Altering the wording or structure of prompts to better align with model expectations.

🧪 Notable Methods:

- OPRO(Yang et al., 2024) – Large Language Models as Optimizers: Uses LLMs to propose and refine prompts iteratively based on performance feedback.
- EvoPrompt(Guo et al., 2024) – LLMs + Evolutionary Algorithms: Applies evolutionary search to discover high-performing prompts through generations.



🎯 Benefit:

Automates the search for high-quality prompts.

Few-Shot Prompting

- ➢ LLM is shown a **few** examples of input-output pairs before being asked to solve a new task.
- ➢ Demonstrates how a problem description maps to formulation.

✅ Benefits:

- Enables LLMs to learn task patterns
- Potentially more accurate than zero-shot prompting

⚠ Limitations:

- Limited by prompt token length (can only include a few examples)
- Performance highly depends on quality and diversity of examples

Few-Shot Prompt

Example

Q: A bakery makes two items: bread and cake. Bread takes 1 hour to bake and cake takes 2 hours. The oven is available for 40 hours per week. Bread earns $4 profit, and cake earns $7. Formulate an integer linear programming model to maximize profit.

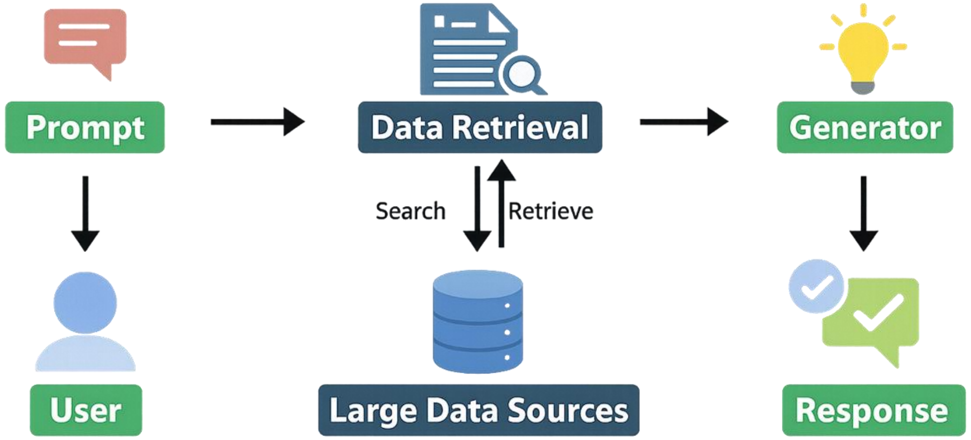
A:
$$\begin{aligned} \max\quad & 4B + 7C \\ \text{s.t.}\quad & B + 2C \le 40 \\ & B, C \in \mathbb{N}_0 \end{aligned}$$

New Task

Q: A company produces products A and B. Each A requires 2 hours of labor, each B requires 1 hour. With 100 total labor hours available and profits of $30 for A and $20 for B, formulate a MIP model to maximize profit.

Retrieval-Augmented Generation (RAG)

- ➢ Enhances LLMs by retrieving relevant documents and inserting them into prompts.
- ➢ Helps handle complex or domain-specific tasks with limited LLM internal knowledge.



🚛 **DRoC (Decomposed Retrieval of Constraints)**(Jiang et al., 2025a) **:**

- Decomposes complex optimization problems into individual constraints.
- Retrieves relevant documentation/code for each constraint.
- Merges retrieved content to assist LLMs in generating accurate solver-compatible programs.

Chain-of-Thought Prompting

Encourages LLMs to reason step-by-step before generating the final formulation(Wei et al., 2022)

Mimics human thought processes by decomposing complex tasks into logical steps

Chain-of-Thought Prompt

Example

Q: A bakery makes two items: bread and cake. Bread takes 1 hour to bake and cake takes 2 hours. The oven is available for 40 hours per week. Bread earns $4 profit, and cake earns $7. Formulate an integer linear programming model to maximize profit.

A: 1) Let B = number of bread loaves, C = number of cakes.
2) These must be non-negative integers.
3) Objective: Maximize profit → 4B + 7C
4) Constraint: Bread takes 1 hour, cake takes 2 → B + 2C ≤ 40

Final model:

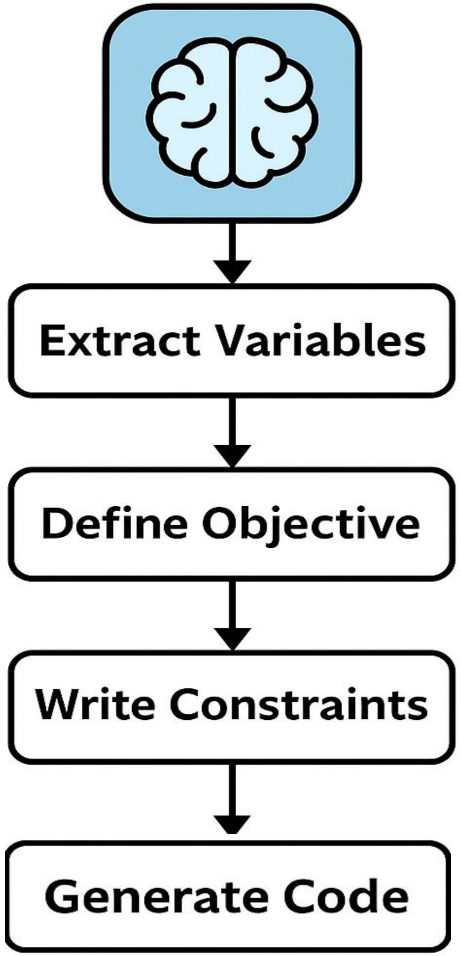
$$\begin{aligned} \max\ & 4B + 7C \\ \text{s.t.}\ & B + 2C \le 40 \\ & B, C \in \mathbb{N}_0 \end{aligned}$$

New Task

Q: A company produces products A and B. Each A requires 2 hours of labor, each B requires 1 hour. With 100 total labor hours available and profits of $30 for A and $20 for B, formulate a MIP model to maximize profit.

Single-Agent Workflow

- A single LLM agent is used to handle the entire problem formulation task in a modular, step-by-step manner.
- The LLM is prompted to sequentially complete subtasks: extract variables, define objective, write constraints, and generate code.
- More scalable for long problems due to reduced context length
- Compatible with prompt-based or fine-tuned models



Multi-Agent Workflow

- ➢ Inspired by expert collaboration and chain-of-responsibility paradigms.
- ➢ Multiple LLMs or role-specialized agents collaborate in a structured workflow to perform different subtasks in problem formulation.
- ➢ Roles may include: Conductor (Coordinator), Interpreter, Formulator, Programmer, and Validator.

✅ **Advantages**:

- Improved performance via task specialization
- Easier to debug or refine each step/agent

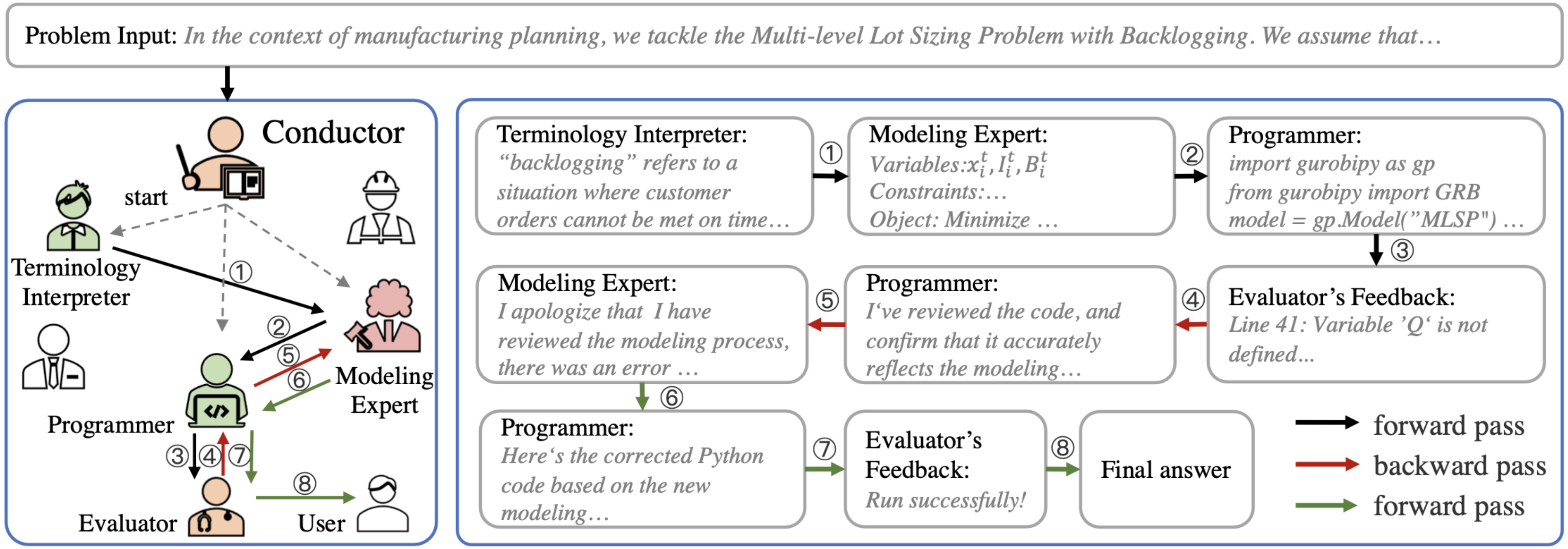
⚠ **Limitations**:

- Higher complexity and resource-intensive (compute, cost, latency)
- Requires careful design of communication between agents

Multi-Agent Workflow Example

Chain-of-Experts (CoE)(Xiao et al., 2024)

- A central Conductor coordinates specialized LLM agents (e.g., Interpreter, Modeler, Programmer).
- Agents collaborate iteratively to analyze, formulate, and verify optimization models.
- Combines forward reasoning (expert-driven modeling) with backward reflection (feedback-based revision).



Problem Input: *In the context of manufacturing planning, we tackle the Multi-level Lot Sizing Problem with Backlogging. We assume that…*

Terminology Interpreter: *"backlogging" refers to a situation where customer orders cannot be met on time…*

Modeling Expert: *Variables:x_i^t, I_i^t, B_i^t Constraints:… Object: Minimize …*

Programmer: *import gurobipy as gp from gurobipy import GRB model = gp.Model("MLSP") …*

Evaluator's Feedback: *Line 41: Variable 'Q' is not defined...*

Programmer: *I've reviewed the code, and confirm that it accurately reflects the modeling…*

Modeling Expert: *I apologize that I have reviewed the modeling process, there was an error …*

Programmer: *Here's the corrected Python code based on the new modeling…*

Evaluator's Feedback: *Run successfully!*

Final answer

→ forward pass

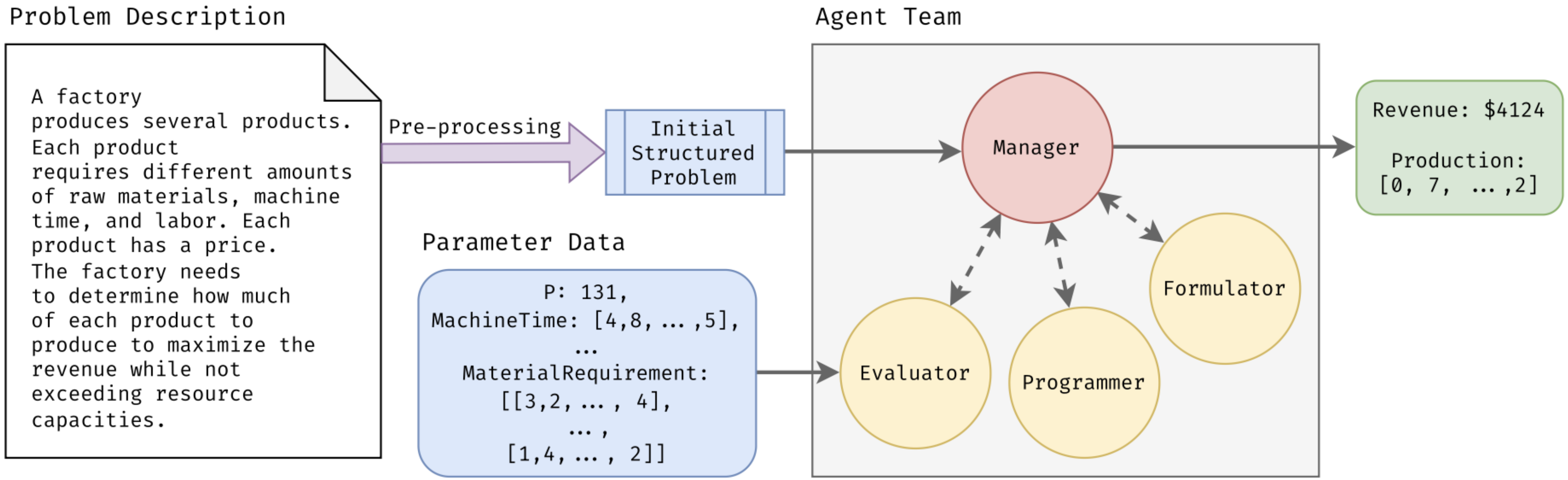
→ backward pass

→ forward pass

Multi-Agent Workflow Example

OptiMUS(AhmadiTeshnizi et al., 2024)

- Manager: Oversees the workflow and coordinates agent interaction.
- Preprocessor: Extracts variables, objectives, and constraints from text.
- Formulator: Converts each clause into formal math (e.g., LaTeX).
- Programmer: Generates and debugs solver code.
- Evaluator: Runs code and checks for correctness.

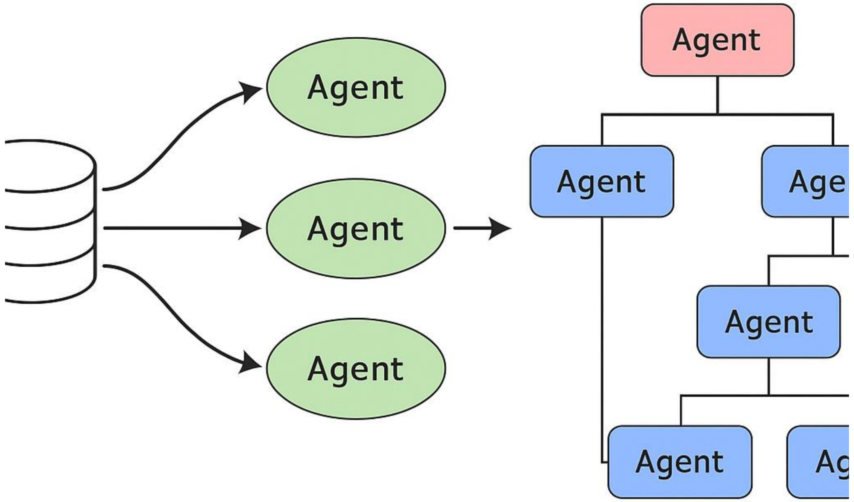

Problem Description
A factory produces several products. Each product requires different amounts of raw materials, machine time, and labor. Each product has a price. The factory needs to determine how much of each product to produce to maximize the revenue while not exceeding resource capacities.
Pre-processing
Initial Structured Problem
Parameter Data
P: 131,
MachineTime: [4,8, … ,5],
…
MaterialRequirement:
[[3,2, … , 4],
… ,
[1,4, … , 2]]
Agent Team
Manager
Formulator
Programmer
Evaluator
Revenue: $4124
Production:
[0, 7, … ,2]

Agentic Workflow Optimisation

- Traditional multi-agent LLM workflows require manual design of agent roles and communication strategies.
- Agentic workflow optimisation aims to automatically generate and refine multi-agent workflows for complex tasks like problem formulation.
- Enhances scalability, adaptability, and performance without hand-engineered pipelines.

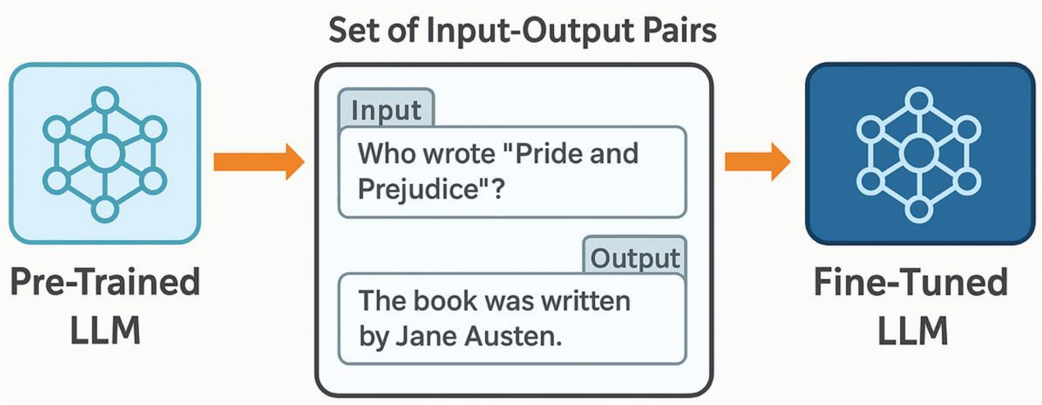
🚀**AFLOW**(Zhang et al., 2025)**: Automating Agentic Workflow Generation**

- Represents workflows as graphs of modular, reusable LLM-invoking nodes.
- Uses Monte Carlo Tree Search (MCTS) to explore and optimize workflows.



Fine-Tuning

- ➢ Refers to training a pre-trained language model on a specific dataset of input-output pairs.
- ➢ Input: problem description
- ➢ Output: formulation & code


Set of Input-Output Pairs
Pre-Trained LLM
Input
Who wrote "Pride and Prejudice"?
Output
The book was written by Jane Austen.
Fine-Tuned LLM

✅ Strengths

- Improves performance on domain-specific or complex tasks
- More reliable and consistent than prompt-only methods

⚠ Limitations

- Requires labelled datasets (e.g., NL + LP/MIP pairs)
- Computationally expensive and time-consuming

Fine-Tuning Examples

Full Fine-Tuning: Updates all model parameters on domain-specific data

Efficient Fine-Tuning: Updates a subset of parameters or adds lightweight modules:

- LoRA(Hu et al., 2022) – Low-rank adaptation to attention weights
- Adapters – Trainable modules inserted into the transformer stack

Recent Efforts

- LM4OPT(Ahmed et al., 2024): Fine-tunes LLaMA-2-7B to convert natural language into optimization models using the NL4Opt dataset.
- OptLLM(Zhang et al., 2024): Trains Qwen-based agents with multi-turn interaction and solver feedback for optimization modeling.
- ORLM(Huang et al., 2024): Fine-tunes several open-source 7B-scale LLMs using the open-instruct framework and OR-Instruct data.
- LMBO(Amarasinghe et al., 2023): Applies full fine-tuning to real-world production scheduling, demonstrating improved task performance and code generation.
- LLMOPT(Jiang et al., 2025b): Combines structured five-part representations with supervised fine-tuning for general-purpose optimization modeling.

Benchmark Datasets

Evaluate the capability and accuracy of LLMs in formulating optimization models

Support fine-tuning and instruction alignment for task specialization

🧪 Existing Benchmark Datasets:

- NL4Opt(Ramamonjison et al., 2023): 1101 annotated LP problems across 6 domains and tasks.
- Mamo(Huang et al., 2025): 863 MILP problem instances (652 Easy, 211 Complex)
- NLP4LP(AhmadiTeshnizi et al., 2024): 65 textbook-sourced LP and MILP instances
- ComplexOR(Xiao et al., 2024): 37 expert-annotated OR problems from diverse real-world sources
- IndustryOR(Huang et al., 2024): 100 real-world OR problems, 5 types, 3 difficulty levels
- OPTIBENCH(Yang et al., 2025): 605 verified problems covering LP, NLP, MIP, and tabular data
- Sched(Amarasinghe et al., 2023): Two scheduling datasets with 1,700 instances via modular expansion.

Synthetic Datasets

Motivation

- Lack of large-scale, high-quality labelled datasets for optimization tasks.
- Manual annotation is expensive, time-consuming, and requires domain expertise.

🔧 Synthetic Dataset & Approach

- **LLMOPT**(Jiang et al., 2025b): augments 1,763 seed problems using diverse instruction templates via GPT-4, followed by expert filtering and detailed labeling.
- **OR-Instruct**(Huang et al., 2024)**:** Generated by expanding 686 seed cases using GPT-4, structured prompts, augmentation (e.g., rephrasing, constraint variation), and human filtering to ensure correctness.
- **RESOCRATIC-29K**(Yang et al., 2025): Generated by reverse-constructing optimization problems from formulations, then translating and filtering them for correctness and diversity.

Challenges and Future Directions

⚠ Challenges

- Ambiguity and missing details in natural language descriptions
- Vast and diverse space of business optimization problem types
- Limited availability of high-quality, labeled datasets for training and evaluation
- Prompt sensitivity, hallucination, and generation of solver-incompatible models
- High cost and complexity of running or accessing large language models

🚀 Future Directions

- Scalable synthetic data generation (e.g., ReSocratic) for robust training
- Enriched benchmarks covering various real-world business optimization tasks
- Enhanced prompt optimization and retrieval-augmented generation methods
- Agentic workflow automation (e.g., AFLOW) for multi-agent coordination
- Integration of validation tools and solver feedback for self-debugging
- Development of cost-efficient LLMs with strong optimization performance

References

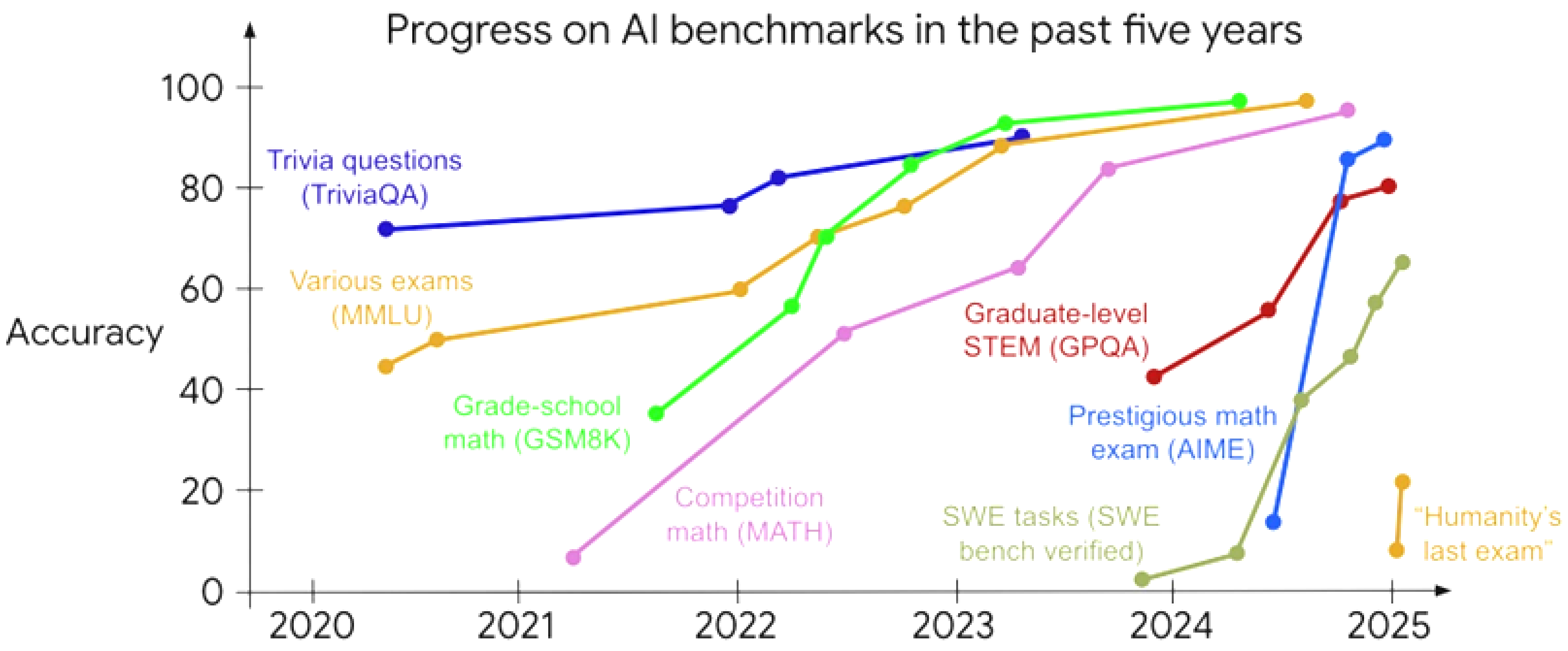
- **(Ramamonjison et al., 2022)** Ramamonjison, R., Li, H., Yu, T., He, S., Rengan, V., Banitalebi-Dehkordi, A., ... & Zhang, Y. (2022, December). Augmenting Operations Research with Auto-Formulation of Optimization Models From Problem Descriptions. In Proceedings of the EMNLP: Industry Track (pp. 29-62).
- **(Ahmed et al., 2024)** Ahmed, T., & Choudhury, S. (2024). LM4OPT: Unveiling the potential of Large Language Models in formulating mathematical optimization problems. INFOR: Information Systems and Operational Research, 62(4), 559-572.
- **(Yang et al., 2024)** Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., & Chen, X. (2024). Large Language Models as Optimizers. In The Twelfth International Conference on Learning Representations (ICLR).
- **(Guo et al., 2024)** Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., ... & Yang, Y. (2024) Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers. In The Twelfth International Conference on Learning Representations (ICLR).
- **(Jiang et al., 2025a)** Jiang, X., Wu, Y., Zhang, C., & Zhang, Y. (2025). DRoC: Elevating Large Language Models for Complex Vehicle Routing via Decomposed Retrieval of Constraints. In 13th International Conference on Learning Representations, ICLR 2025.
- **(Wei et al., 2022)** Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., ... & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems (NeurIPS), 35, 24824-24837.
- **(Xiao et al., 2024)** Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., ... & Chen, G. (2024). Chain-of-experts: When LLMs meet complex operations research problems. In The twelfth International Conference on Learning Representations (ICLR).
- **(AhmadiTeshnizi et al., 2024)** AhmadiTeshnizi, A., Gao, W., & Udell, M. (2024, July). OptiMUS: scalable optimization modeling with (MI)LP solvers and large language models. In Proceedings of the 41st International Conference on Machine Learning (pp. 577-596).
- **(Zhang et al., 2025)** Zhang, J., Xiang, J., Yu, Z., Teng, F., Chen, X., Chen, J., ... & Wu, C. (2025). AFlow: Automating agentic workflow generation. In The Thirteenth International Conference on Learning Representations (ICLR).
- **(Hu et al., 2022)** Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In International Conference on Learning Representations ICLR, 1(2), 3.
- **(Zhang et al., 2024)** Zhang, J., Wang, W., Guo, S., Wang, L., Lin, F., Yang, C., & Yin, W. (2024, June). Solving General Natural-Language-Description Optimization Problems with Large Language Models. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track) (pp. 483-490).
- **(Huang et al., 2024)** Huang, C., Tang, Z., Ge, D., Hu, S., Jiang, R., Wang, B., ... & Zheng, X. (2024). ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling. Accepted by Operations Research.
- **(Amarasinghe et al., 2023)** Amarasinghe, P. T., Nguyen, S., Sun, Y., & Alahakoon, D. (2023). Language Models for Business Optimisation with a Real World Case Study in Production Scheduling. arXiv preprint arXiv:2309.13218.
- **(Jiang et al., 2025b)** Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., & Yu, Y. (2025). LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch. In The Thirteenth International Conference on Learning Representations (ICLR).
- **(Ramamonjison et al., 2023)** Ramamonjison, R., Yu, T., Li, R., Li, H., Carenini, G., Ghaddar, B., ... & Zhang, Y. (2023, August). NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In NeurIPS 2022 Competition Track (pp. 189-203). PMLR.
- **(Huang et al., 2025)** Huang, X., Shen, Q., Hu, Y., Gao, A., & Wang, B. (2025, April). LLMs for Mathematical Modeling: Towards Bridging the Gap between Natural and Mathematical Languages. In Findings of the Association for Computational Linguistics: NAACL 2025(pp. 2678-2710).
- **(Yang et al., 2025)** Yang, Z., Wang, Y., Huang, Y., Guo, Z., Shi, W., Han, X., ... & Tang, J. (2025). OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In The Thirteenth International Conference on Learning Representations (ICLR).

Evolutionary Optimization Guided by Large Models

Huigen Ye, Tsinghua University, Beijing, P. R. China

Background

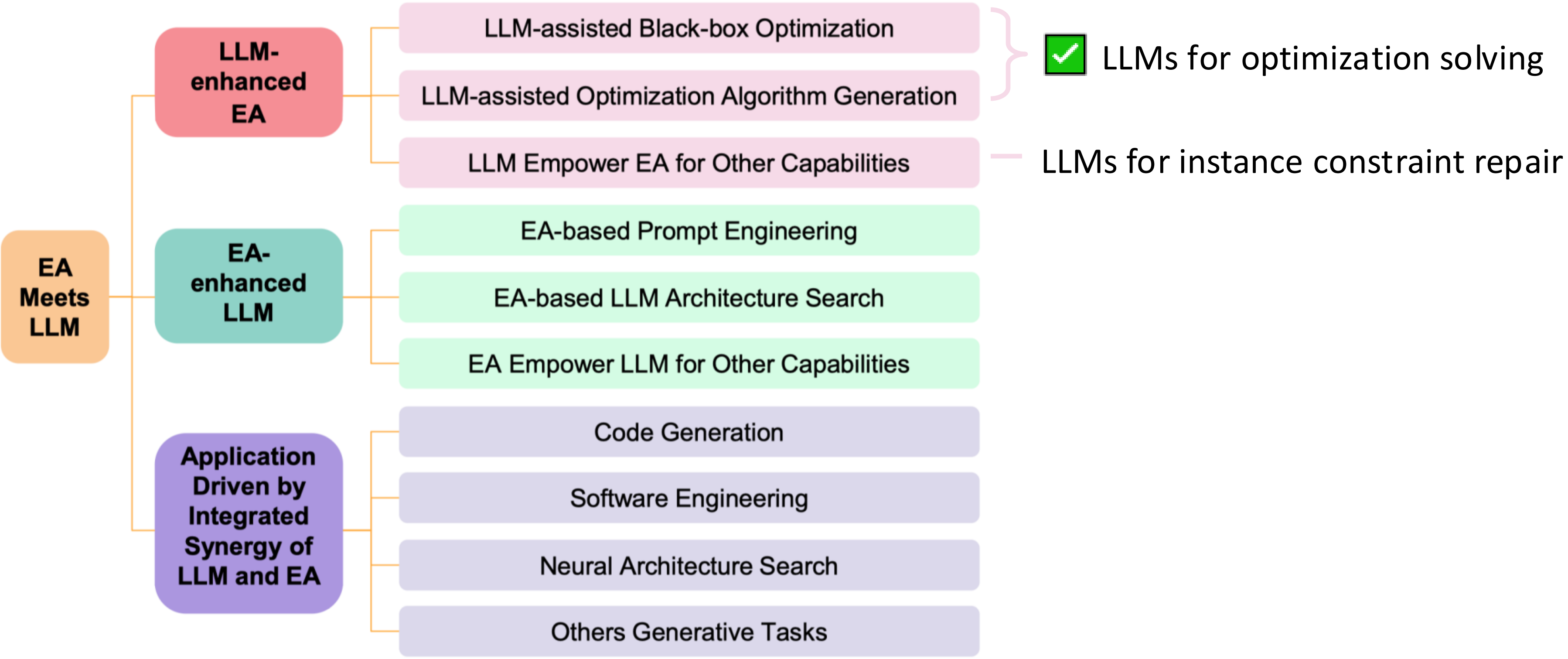
- **Rapid Progress of Large Models**
  - LLM capabilities have accelerated rapidly
  - Complex tasks are now within reach
  - “Impossible” problems are becoming solvable



(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

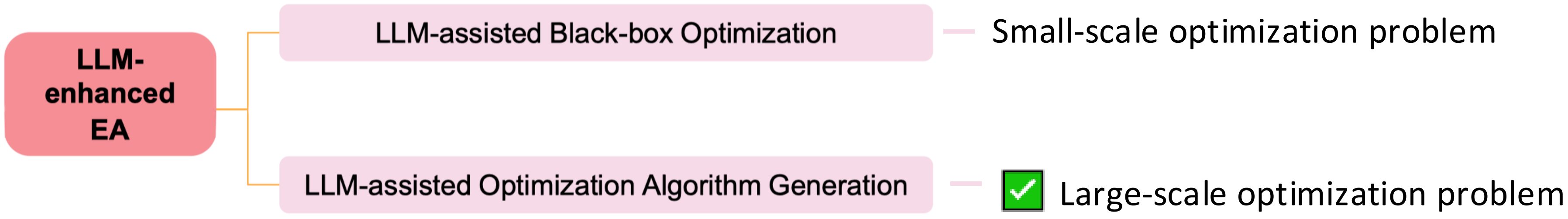


EA Meets LLM
LLM-enhanced EA
LLM-assisted Black-box Optimization
LLM-assisted Optimization Algorithm Generation
LLM Empower EA for Other Capabilities
EA-enhanced LLM
EA-based Prompt Engineering
EA-based LLM Architecture Search
EA Empower LLM for Other Capabilities
Application Driven by Integrated Synergy of LLM and EA
Code Generation
Software Engineering
Neural Architecture Search
Others Generative Tasks
✓ LLMs for optimization solving
LLMs for instance constraint repair

(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



LLM-enhanced EA

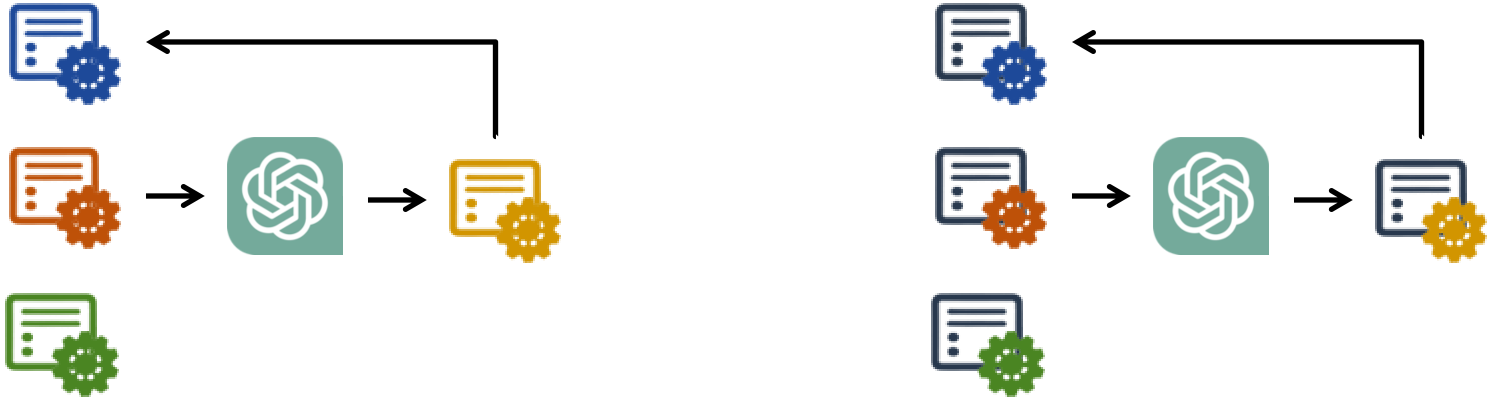
LLM-assisted Black-box Optimization — Small-scale optimization problem

LLM-assisted Optimization Algorithm Generation — ✅ Large-scale optimization problem

(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

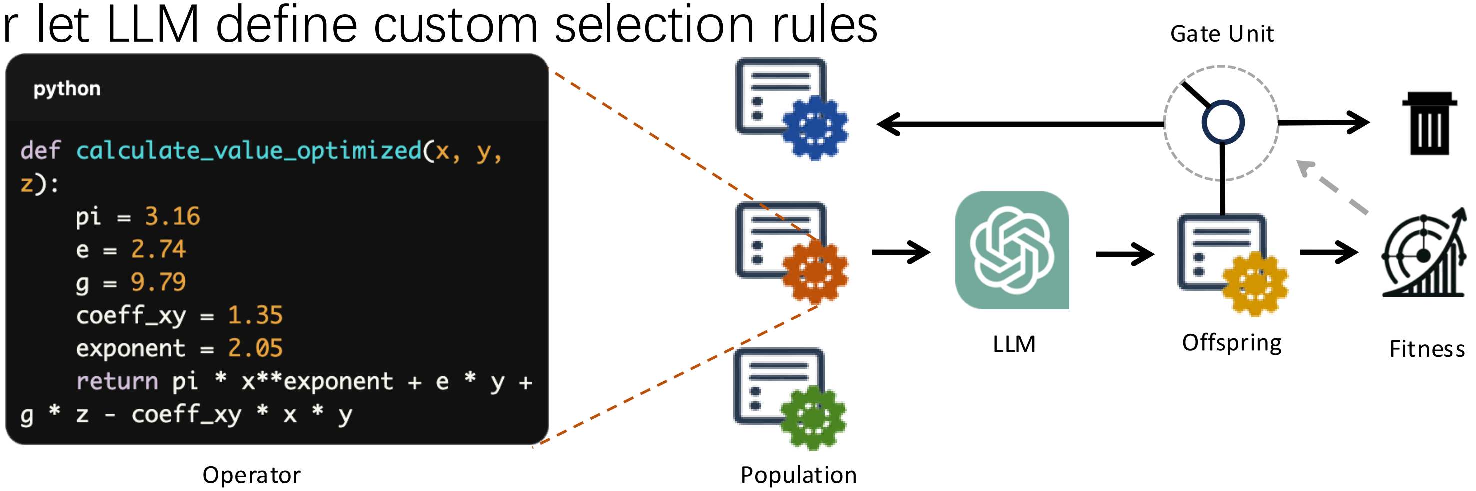


(a) Evolve the entire heuristic algorithm

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

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



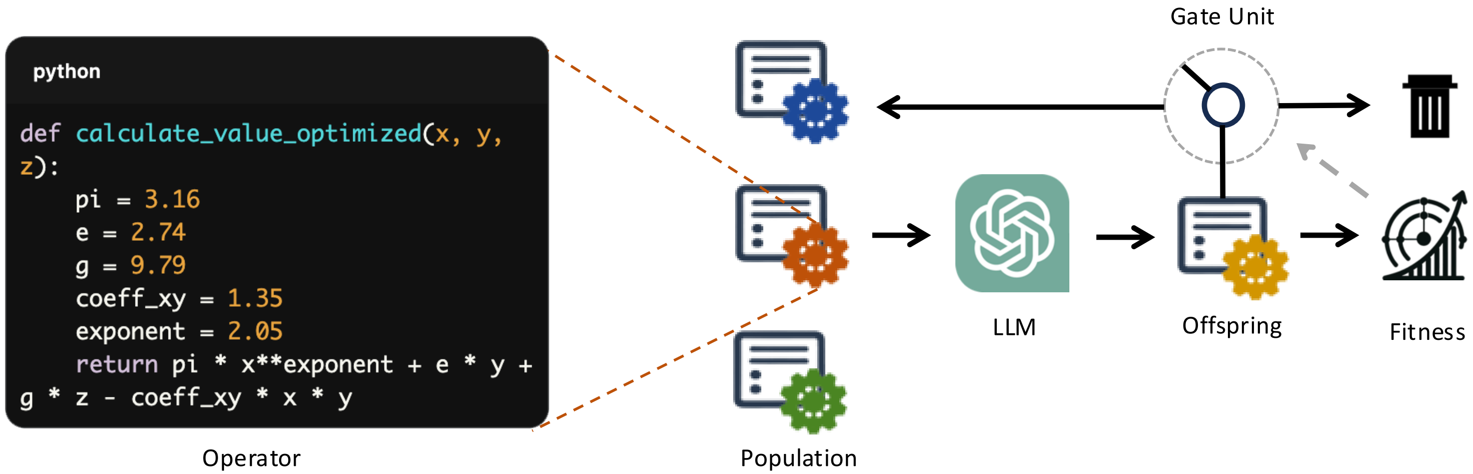
python

```python
def calculate_value_optimized(x, y,
z):
    pi = 3.16
    e = 2.74
    g = 9.79
    coeff_xy = 1.35
    exponent = 2.05
    return pi * x**exponent + e * y +
g * z - coeff_xy * x * y
```

(a) EA + LLM workflow for evolving heuristic operators

Basic Method

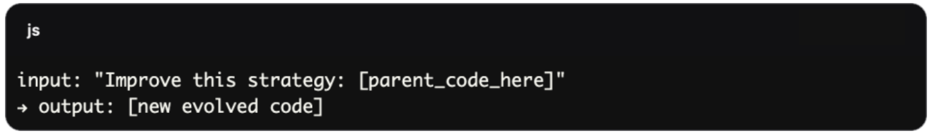
- Practical Workflow of EA + LLM for Code Evolution
  - Generate Offspring
    - Use LLM as a crossover/mutation operator
    - Input parent code, output new candidate strategy
  - Evaluate and Filter
    - Run offspring on tasks to compute fitness
    - Gate unit decides whether to keep, replace, or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



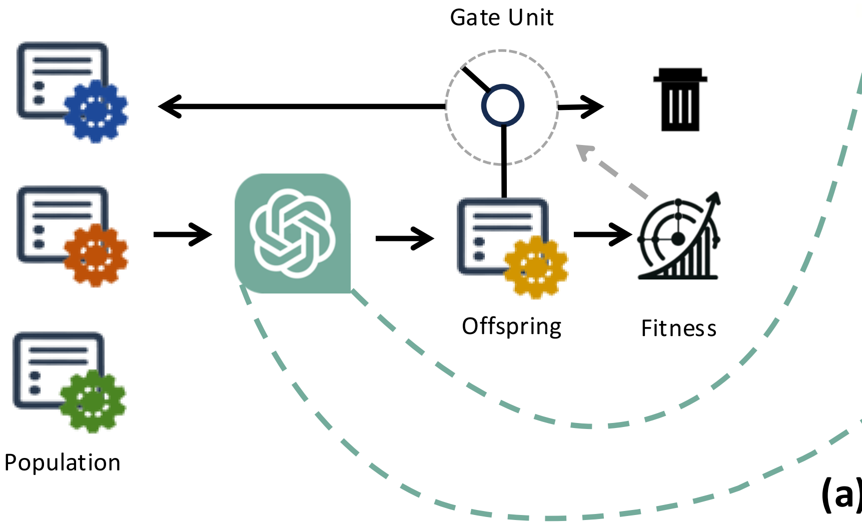
```
python

def calculate_value_optimized(x, y,
z):
    pi = 3.16
    e = 2.74
    g = 9.79
    coeff_xy = 1.35
    exponent = 2.05
    return pi * x**exponent + e * y +
g * z - coeff_xy * x * y
```

(a) EA + LLM workflow for evolving heuristic operators

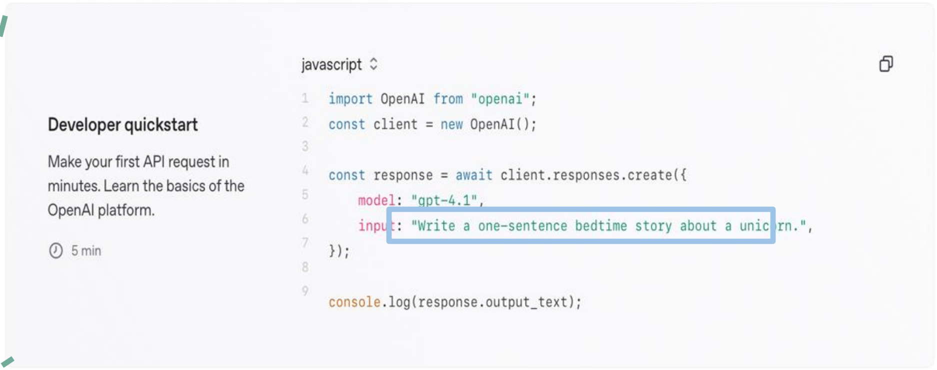
Basic Method

- 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]
```



OpenAI developer platform



Developer quickstart

Make your first API request in minutes. Learn the basics of the OpenAI platform.

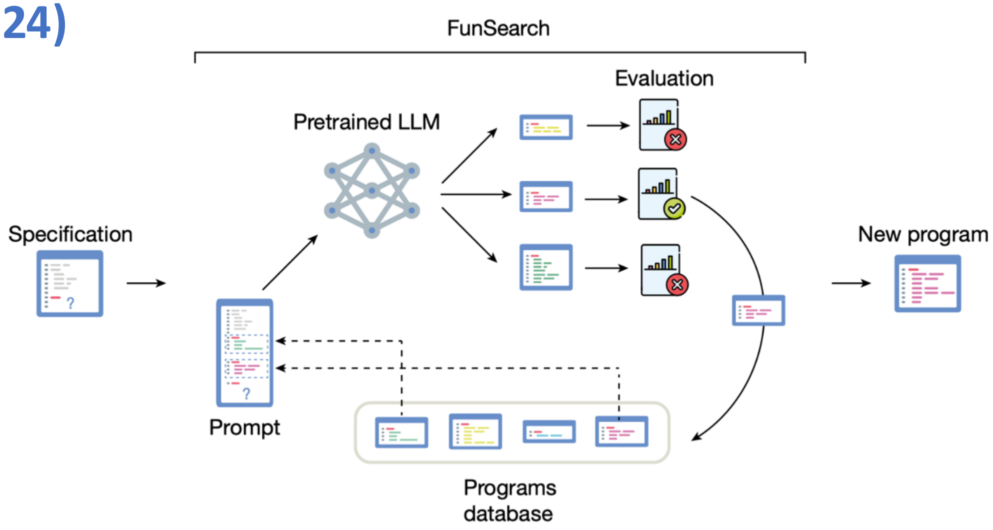
5 min

```javascript
import OpenAI from "openai";
const client = new OpenAI();

const response = await client.responses.create({
    model: "gpt-4.1",
    input: "Write a one-sentence bedtime story about a unicorn.",
});

console.log(response.output_text);
```

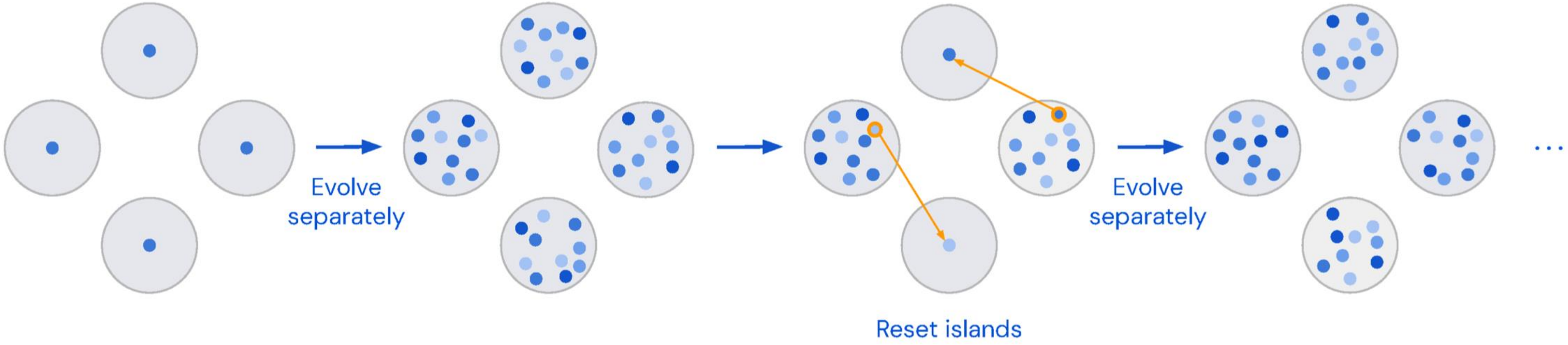
(a) Parent code + prompt → GPT → New offspring code

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



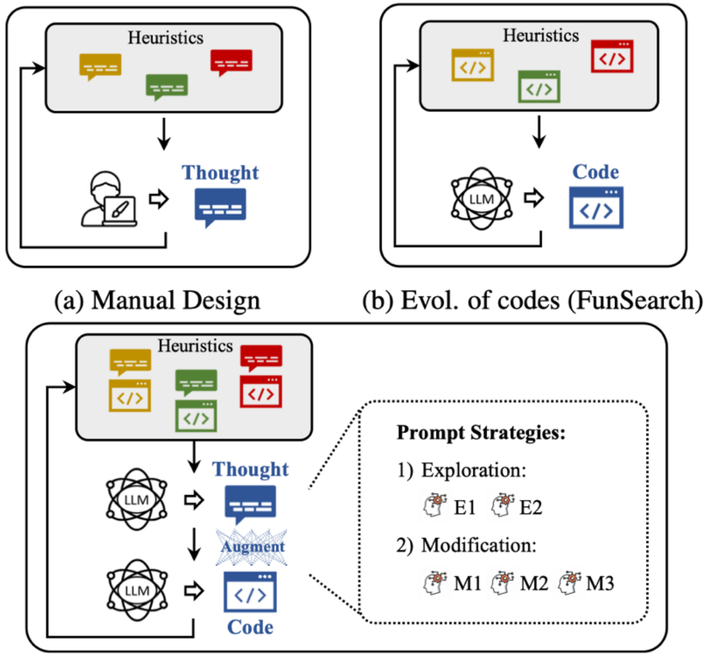
(a) FunSearch framework[3]



(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



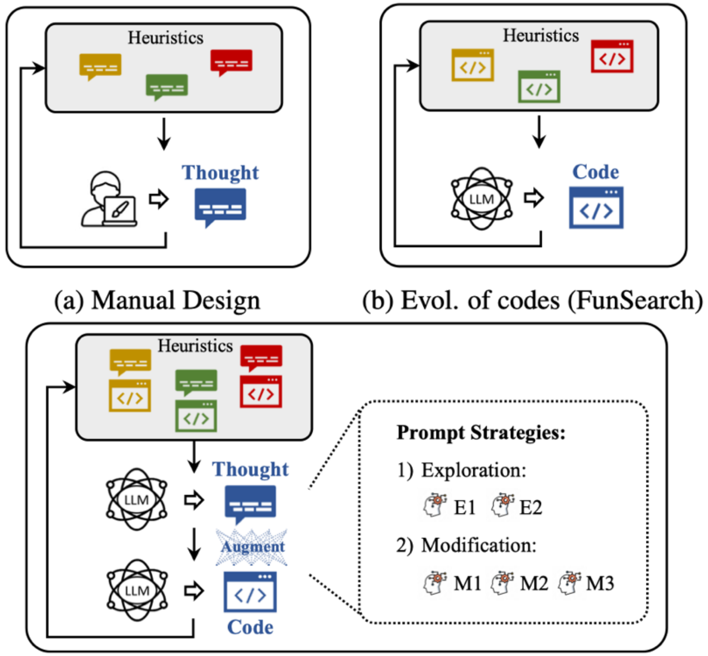
(a) Manual Design

(b) Evol. of codes (FunSearch)

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

Advanced Method

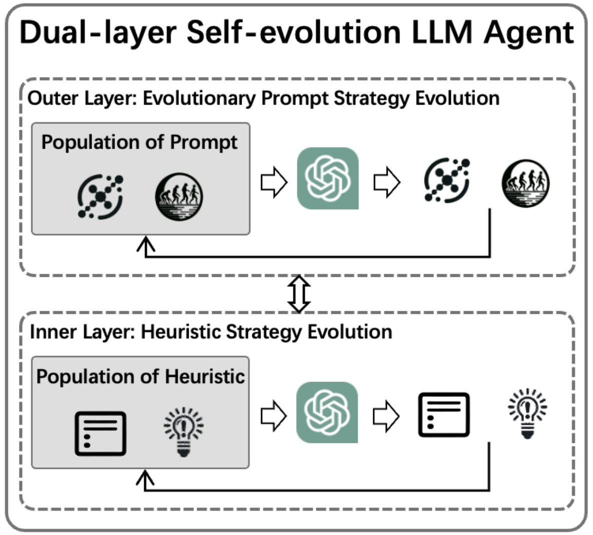
- [ICML2024 Oral] Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model (Liu et al., 2024)
  - Prompts' Key Idea:
    - Use LLM as an evolutionary operator with carefully designed prompt strategies→ Mimic how humans generate new ideas
  - ✨ Two Categories of 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



Dual-layer Self-evolution LLM Agent

Outer Layer: Evolutionary Prompt Strategy Evolution

Population of Prompt

Inner Layer: Heuristic Strategy Evolution

Population of Heuristic

(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**

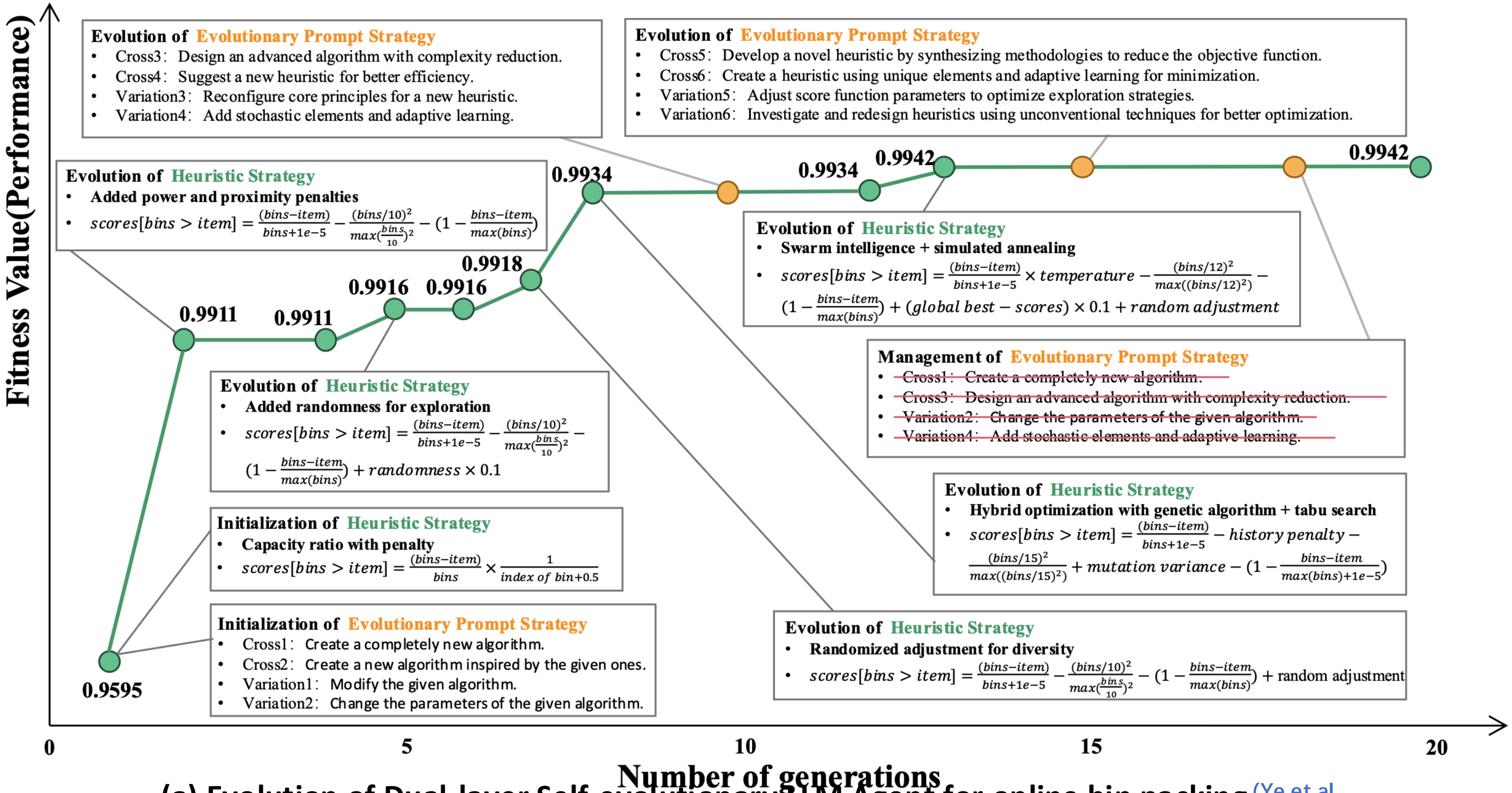
(b) Functional roles of the two layers in the self-evolutionary LLM agent

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)



Initialization of Evolutionary Prompt Strategy

- Cross1: Create a completely new algorithm.
- Cross2: Create a new algorithm inspired by the given ones.
- Variation1: Modify the given algorithm.
- Variation2: Change the parameters of the given algorithm.

Initialization of Heuristic Strategy

- **Capacity ratio with penalty**
- $scores[bins > item] = \frac{(bins-item)}{bins} \times \frac{1}{index\ of\ bin+0.5}$

Evolution of Heuristic Strategy

- **Added power and proximity penalties**
- $scores[bins > item] = \frac{(bins-item)}{bins+1e-5} - \frac{(bins/10)^2}{max(\frac{bins}{10})^2} - (1 - \frac{bins-item}{max(bins)})$

Evolution of Heuristic Strategy

- **Added randomness for exploration**
- $scores[bins > item] = \frac{(bins-item)}{bins+1e-5} - \frac{(bins/10)^2}{max(\frac{bins}{10})^2} - (1 - \frac{bins-item}{max(bins)}) + randomness \times 0.1$

Evolution of Heuristic Strategy

- **Randomized adjustment for diversity**
- $scores[bins > item] = \frac{(bins-item)}{bins+1e-5} - \frac{(bins/10)^2}{max(\frac{bins}{10})^2} - (1 - \frac{bins-item}{max(bins)})$ + random adjustment

Evolution of Heuristic Strategy

- **Hybrid optimization with genetic algorithm + tabu search**
- $scores[bins > item] = \frac{(bins-item)}{bins+1e-5} - history\ penalty - \frac{(bins/15)^2}{max((bins/15)^2)} + mutation\ variance - (1 - \frac{bins-item}{max(bins)+1e-5})$

Evolution of Evolutionary Prompt Strategy

- Cross3: Design an advanced algorithm with complexity reduction.
- Cross4: Suggest a new heuristic for better efficiency.
- Variation3: Reconfigure core principles for a new heuristic.
- Variation4: Add stochastic elements and adaptive learning.

Evolution of Heuristic Strategy

- **Swarm intelligence + simulated annealing**
- $scores[bins > item] = \frac{(bins-item)}{bins+1e-5} \times temperature - \frac{(bins/12)^2}{max((bins/12)^2)} - (1 - \frac{bins-item}{max(bins)}) + (global\ best - scores) \times 0.1 + random\ adjustment$

Evolution of Evolutionary Prompt Strategy

- Cross5: Develop a novel heuristic by synthesizing methodologies to reduce the objective function.
- Cross6: Create a heuristic using unique elements and adaptive learning for minimization.
- Variation5: Adjust score function parameters to optimize exploration strategies.
- Variation6: Investigate and redesign heuristics using unconventional techniques for better optimization.

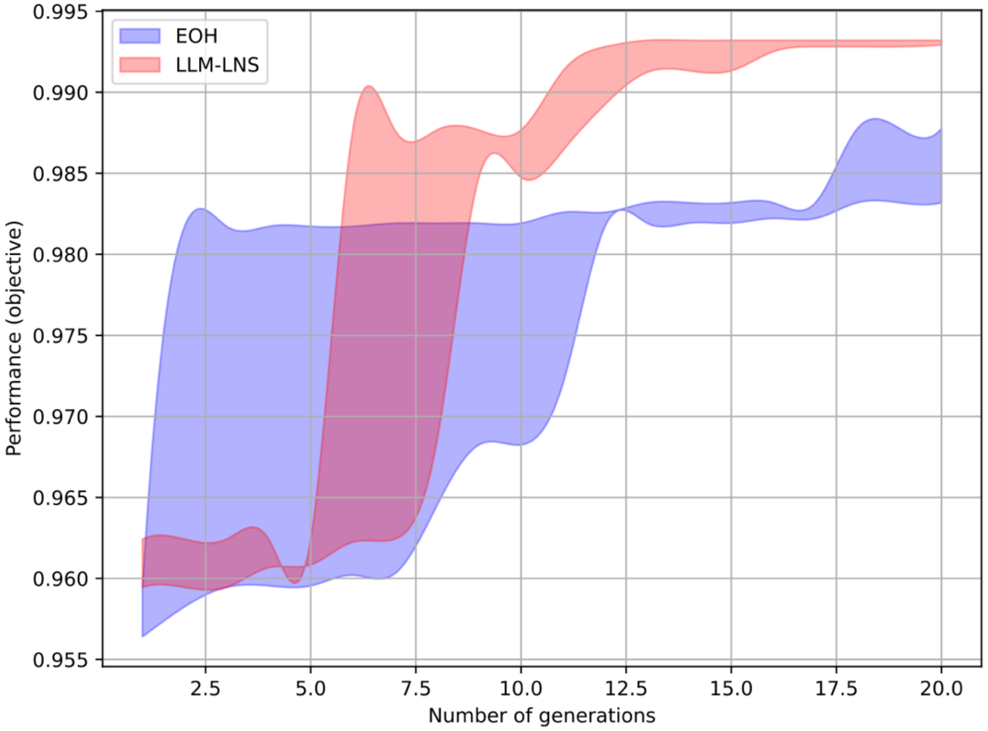
Management of Evolutionary Prompt Strategy

- ~~Cross1: Create a completely new algorithm.~~
- ~~Cross3: Design an advanced algorithm with complexity reduction.~~
- ~~Variation2: Change the parameters of the given algorithm.~~
- ~~Variation4: Add stochastic elements and adaptive learning.~~

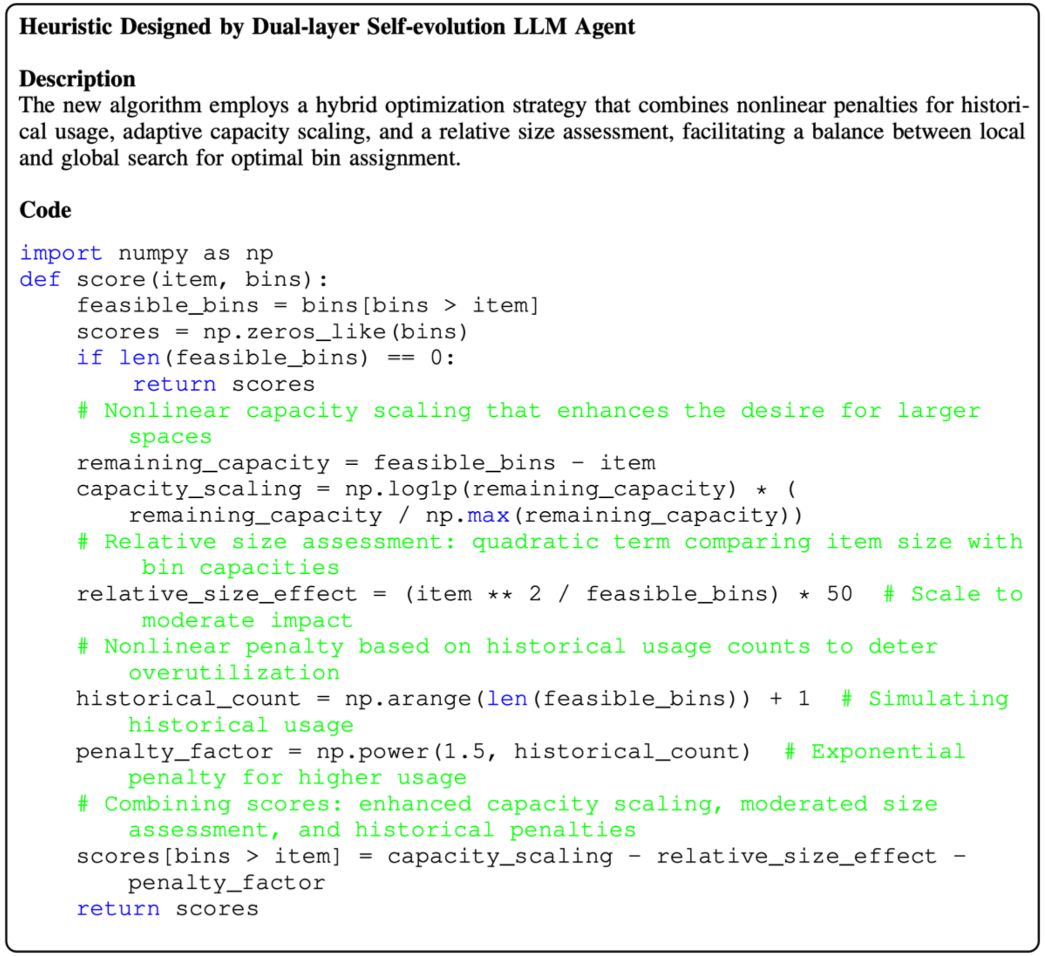
(a) Evolution of Dual-layer Self-evolutionary LLM Agent for online bin packing (Ye et al., 2025)

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

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

- **(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.

About Us

THUIAR Group(Website: https://thuiar.github.io/)

- Department of Computer Science and Technology, **Tsinghua University**

Main Research Direction

What we are doing

Multi-modal Intelligent Information Processing

Emotion recognition, sentiment analysis and intention recognition based on multi-modal information such as text, audio, video (picture).

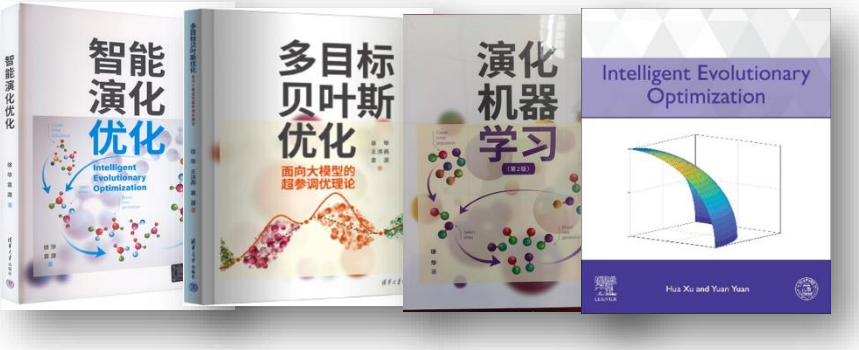
Key Technologies of Intelligent Mobile Robots

Human-machine dialogue technology, intelligent mobile robot control technology and scene applications (business intelligent service robot, dual-mode intelligent disinfection robot).

Research on Intelligent Optimization Method

Evolution-based optimization theory, (high-dimensional, sparse, expensive) multi-objective optimization problem.

- Academic Publications:



Test Case Generation Using Large Language Models

Yuan Yuan, Beihang University, Beijing, P. R. China

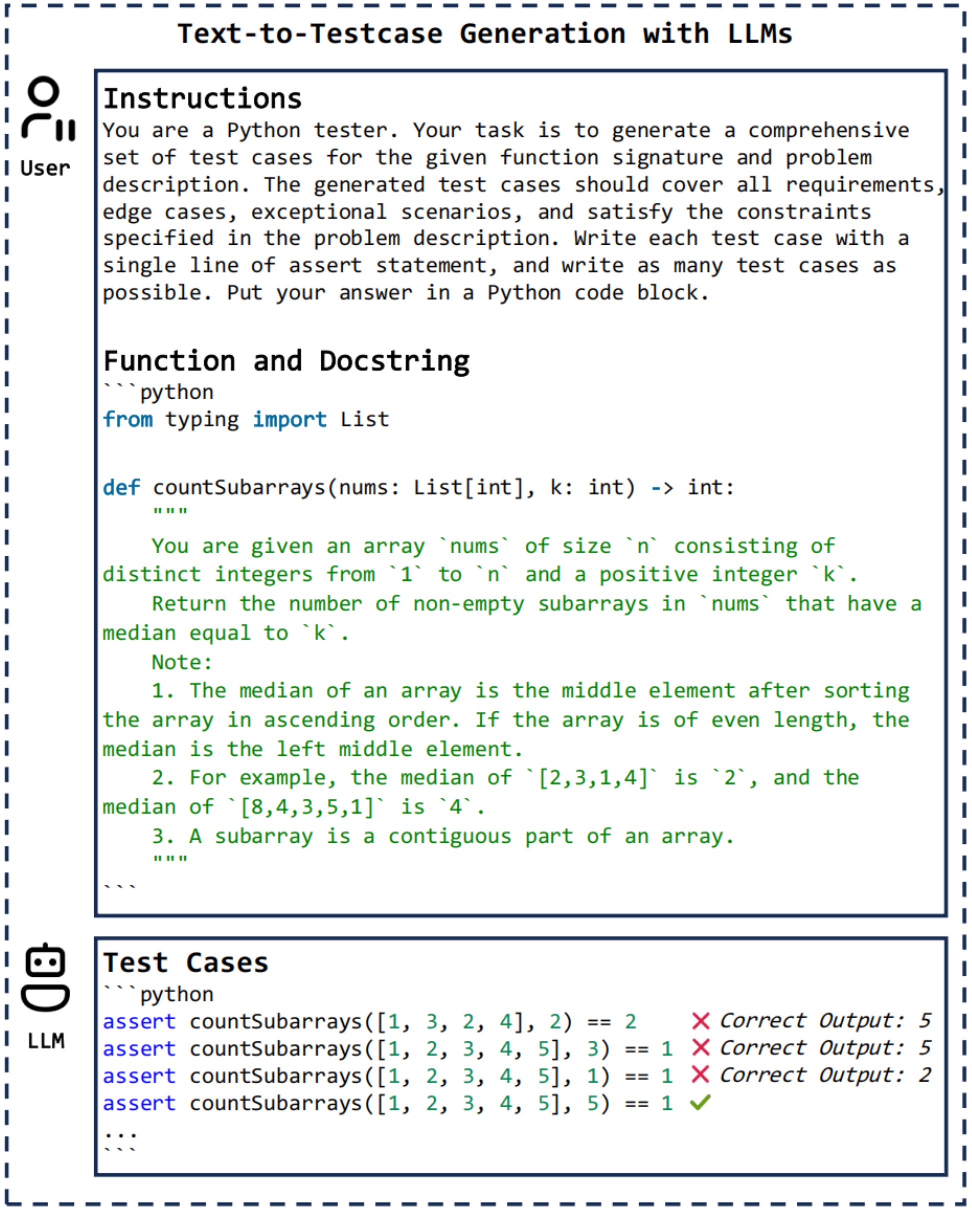
Contents

- Introduction & Motivation
- Background & Prior Work
- TestChain Framework
- Experiments
- Discussion

Introduction & Motivation

Text-to-Testcase Generation

- Converts natural-language problem descriptions into executable test code
- **Input:** Problem description, target function definition
- **Output:** Test cases that can be used to evaluate programs



Text-to-Testcase Generation with LLMs

Instructions

You are a Python tester. Your task is to generate a comprehensive set of test cases for the given function signature and problem description. The generated test cases should cover all requirements, edge cases, exceptional scenarios, and satisfy the constraints specified in the problem description. Write each test case with a single line of assert statement, and write as many test cases as possible. Put your answer in a Python code block.

Function and Docstring

~~~
```python
from typing import List

def countSubarrays(nums: List[int], k: int) -> int:
    """
    You are given an array `nums` of size `n` consisting of
distinct integers from `1` to `n` and a positive integer `k`.
    Return the number of non-empty subarrays in `nums` that have a
median equal to `k`.
    Note:
    1. The median of an array is the middle element after sorting
the array in ascending order. If the array is of even length, the
median is the left middle element.
    2. For example, the median of `[2,3,1,4]` is `2`, and the
median of `[8,4,3,5,1]` is `4`.
    3. A subarray is a contiguous part of an array.
    """
```
~~~

Test Cases

~~~
```python
assert countSubarrays([1, 3, 2, 4], 2) == 2    ✗ Correct Output: 5
assert countSubarrays([1, 2, 3, 4, 5], 3) == 1 ✗ Correct Output: 5
assert countSubarrays([1, 2, 3, 4, 5], 1) == 1 ✗ Correct Output: 2
assert countSubarrays([1, 2, 3, 4, 5], 5) == 1 ✓
...
```
~~~

Fig. 1 Illustration of the process of text-to-testcase generation.

Introduction & Motivation

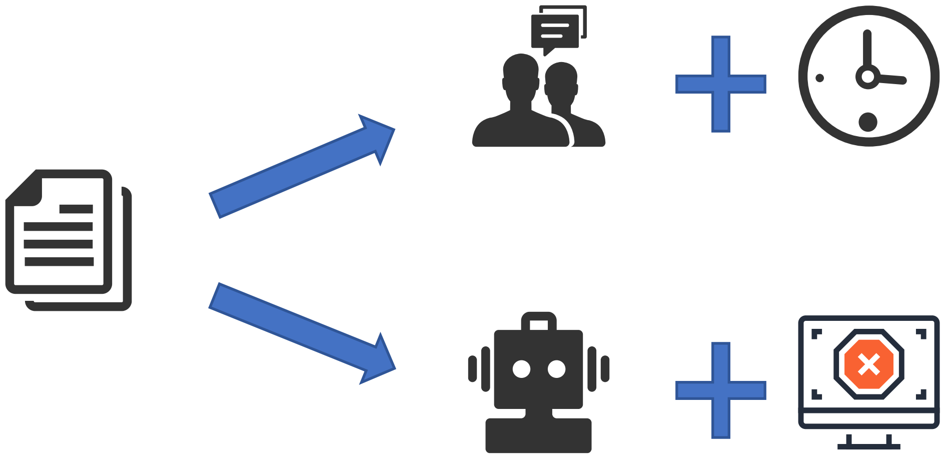
Key Features (Text-to-Testcase Generation)

- **No Reference Program:** Infer expected behavior solely from natural-language descriptions, without a ground-truth program to validate against
- **Risk of Incorrect Test Cases:** LLMs may hallucinate or misinterpret the specification
- **Beyond Traditional Rule-Based Methods:** Rule-driven approaches cannot easily scale to arbitrary problem descriptions
- **Need for Semantic Understanding:** Generating valid tests requires deep comprehension of requirements

Introduction & Motivation

Why Automatic Text-to-TestCase Generation Matters

- Test cases are critical for ensuring code quality and reliability
- Manual test writing is time-consuming, LLMs offer promise but often produce incorrect tests



Requires a lot of manual work

Easy and fast, but may contain errors

Introduction & Motivation

- **Key Challenge**
  - LLMs need to predict the correct expected test output, which is difficult when the reference program is available
- **Main Contributions**
  - **A multi-agent framework**: TestChain
  - **Code-Assisted Reasoning** to eliminates LLM hallucinations and input-output mismatches

Background & Prior Work

- **Traditional Approaches**
  - Rule-based or template methods lack flexibility
- **LLM-Based Approaches**
  - **CodeT**(Chen et al., 2022a) **/ Reflexion**(Shinn et al., 2023)**:** Generate tests from prompts, but ignore correctness
  - **Pytester-770M**(Takerngsaksiri et al., 2025)**:** RL-trained, still prone to copying patterns or missing edge cases

TestChain Framework

- **Framework Overview**
  - **Core Idea:** Factorize generation into two specialized agents
  - **Pipeline FLow:** Prompt → 2. Designer → {Candidate Inputs} → 3. Reasoning → {Final Test Cases}

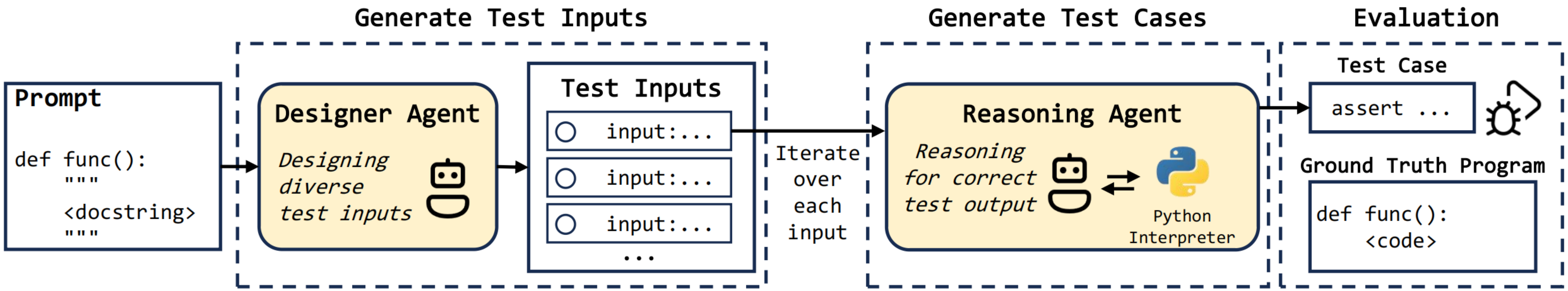


Fig.2 Illustration of the TestChain framework.

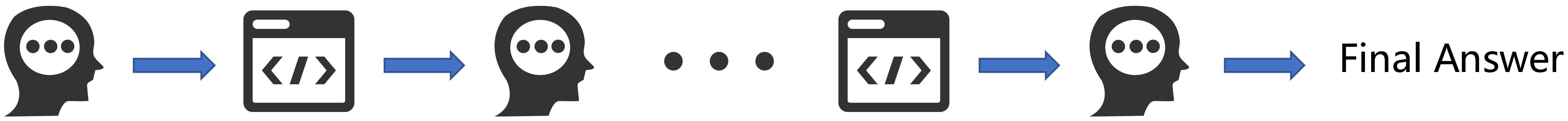
TestChain Framework

- **Designer Agent**
  - **Task:** Generate diverse, representative test inputs
  - **Prompt Template (Example):**

~~~
You are a Python tester. Your task is to generate a comprehensive set of test inputs for
the given function signature and problem description.
The generated test inputs should cover all requirements, edge cases, exceptional
scenarios, and satisfy the constraints specified in the problem description.
Write each test input in a single line and start with a `input:` prefix, and write as many
test inputs as possible. Put your answer in a text block, for example:
```text
# test inputs here
```
~~~

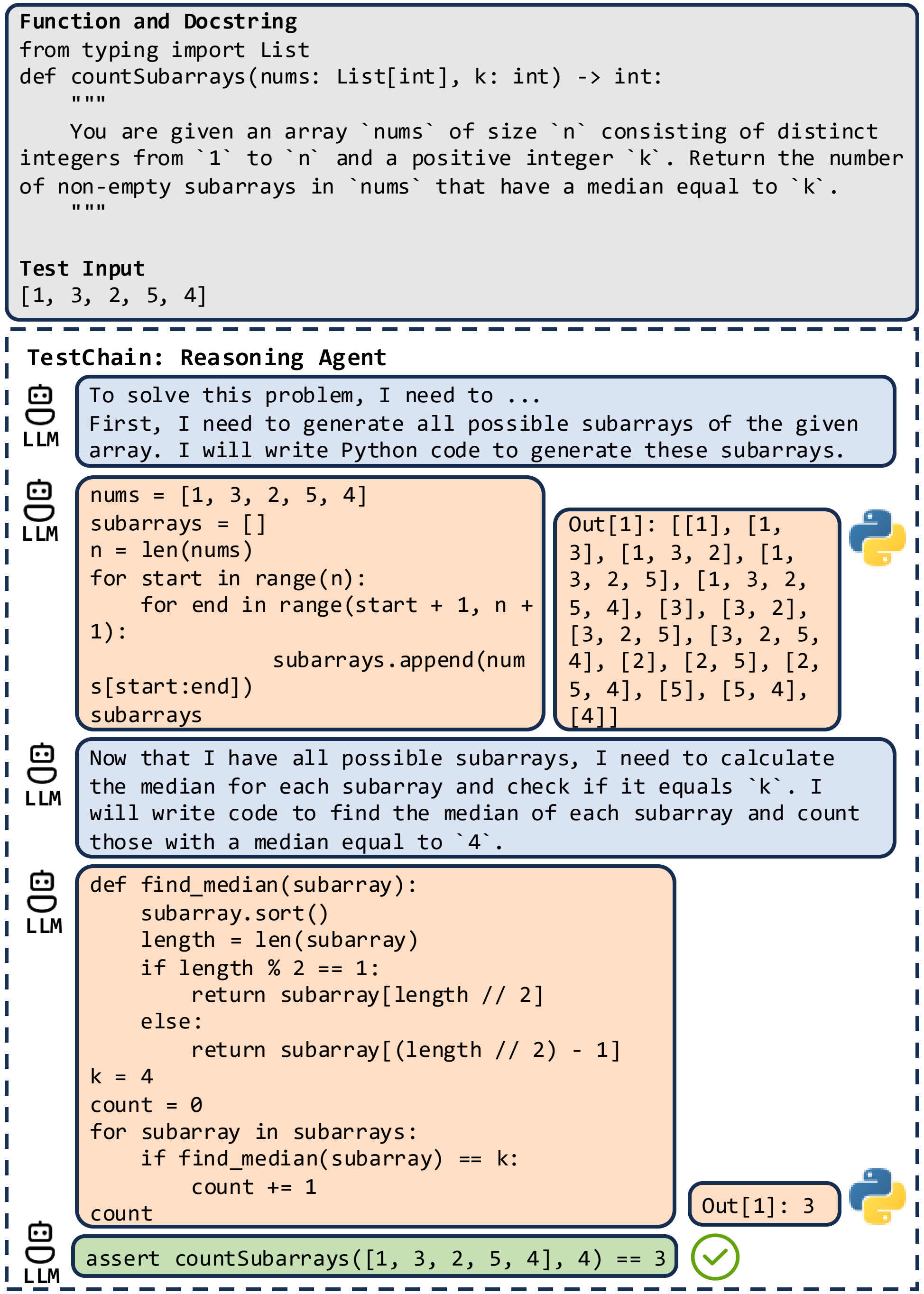
TestChain Framework

- **Reasoning Agent**
  - **Task:** For each input tuple, compute the correct output step-by-step
  - **Two-Phase Strategy:**
    1. **Decomposition:** LLM generates natural-language reasoning steps (e.g., “Sort array, find median, count subarrays”).
    2. **Code-Assisted Execution:** Alternate between reasoning and code-execution



TestChain Framework

- **Code-Assisted Flow (Reasoning Agent)**
    - LLM writes small Python snippets to carry out sub-computations
    - Snippets are executed in a live interpreter, actual numeric results are fed back
    - Final output is computed based on verified intermediate values



Function and Docstring

```
from typing import List
def countSubarrays(nums: List[int], k: int) -> int:
    """
    You are given an array `nums` of size `n` consisting of distinct
integers from `1` to `n` and a positive integer `k`. Return the number
of non-empty subarrays in `nums` that have a median equal to `k`.
    """
```

Test Input

```
[1, 3, 2, 5, 4]
```

TestChain: Reasoning Agent

To solve this problem, I need to ...
First, I need to generate all possible subarrays of the given array. I will write Python code to generate these subarrays.

```
nums = [1, 3, 2, 5, 4]
subarrays = []
n = len(nums)
for start in range(n):
    for end in range(start + 1, n +
1):
            subarrays.append(num
s[start:end])
subarrays
```

```
Out[1]: [[1], [1,
3], [1, 3, 2], [1,
3, 2, 5], [1, 3, 2,
5, 4], [3], [3, 2],
[3, 2, 5], [3, 2, 5,
4], [2], [2, 5], [2,
5, 4], [5], [5, 4],
[4]]
```

Now that I have all possible subarrays, I need to calculate the median for each subarray and check if it equals `k`. I will write code to find the median of each subarray and count those with a median equal to `4`.

```
def find_median(subarray):
    subarray.sort()
    length = len(subarray)
    if length % 2 == 1:
        return subarray[length // 2]
    else:
        return subarray[(length // 2) - 1]
k = 4
count = 0
for subarray in subarrays:
    if find_median(subarray) == k:
        count += 1
count
```

```
Out[1]: 3
```

```
assert countSubarrays([1, 3, 2, 5, 4], 4) == 3
```

Fig.3 Illustration of an end-to-end example.

Experiments

- **Experimental Procedure**
  - **Test case types**
    1. **Correct (Acc.):** Validates correct behavior without errors
    2. **Duplicate (Dup.):** Repeats previous check
    3. **Syntactically Incorrect (Syn.):** Fails to par due to syntax errors
    4. **Input-Output MisMatch (Mis.):** Asserts wrong output for given input
    5. **Other Error (Oth.):** Causes unexpected runtime exceptions

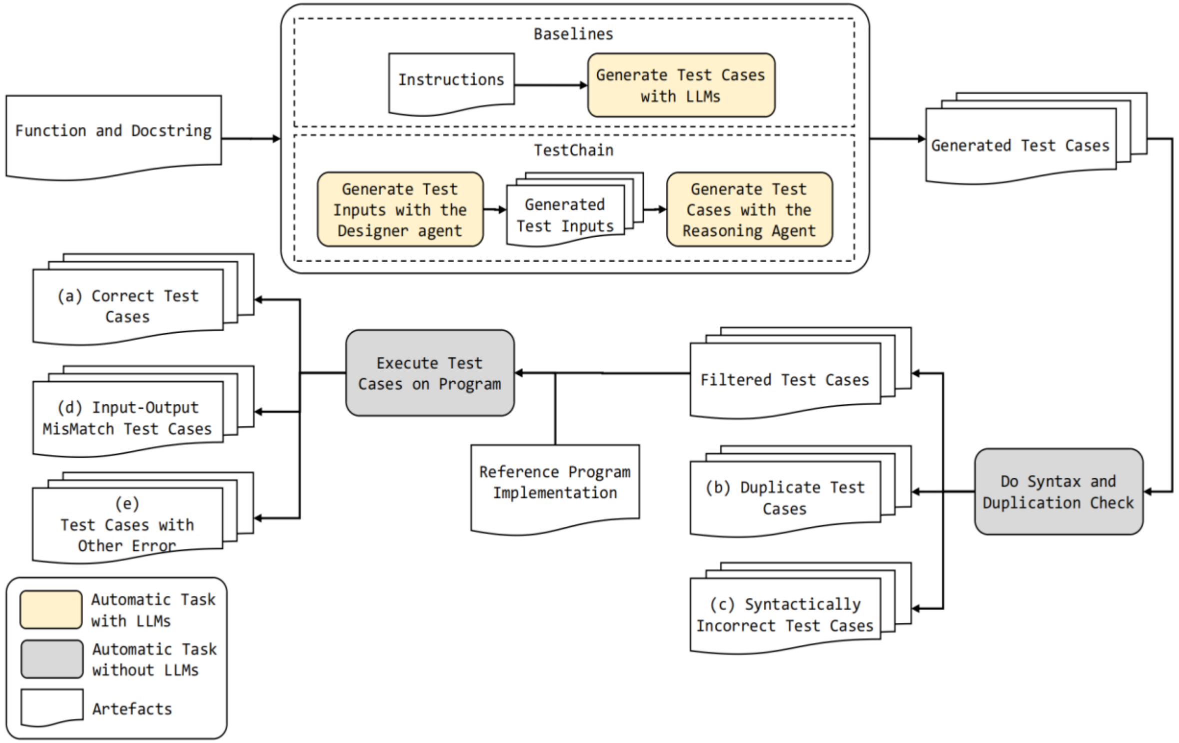

Baselines
Instructions
Generate Test Cases with LLMs
Function and Docstring
TestChain
Generate Test Inputs with the Designer agent
Generated Test Inputs
Generate Test Cases with the Reasoning Agent
Generated Test Cases
(a) Correct Test Cases
(d) Input-Output MisMatch Test Cases
(e) Test Cases with Other Error
Execute Test Cases on Program
Filtered Test Cases
Reference Program Implementation
(b) Duplicate Test Cases
Do Syntax and Duplication Check
(c) Syntactically Incorrect Test Cases
Automatic Task with LLMs
Automatic Task without LLMs
Artefacts

Fig.4 Illustration of the Experimental Procedure.

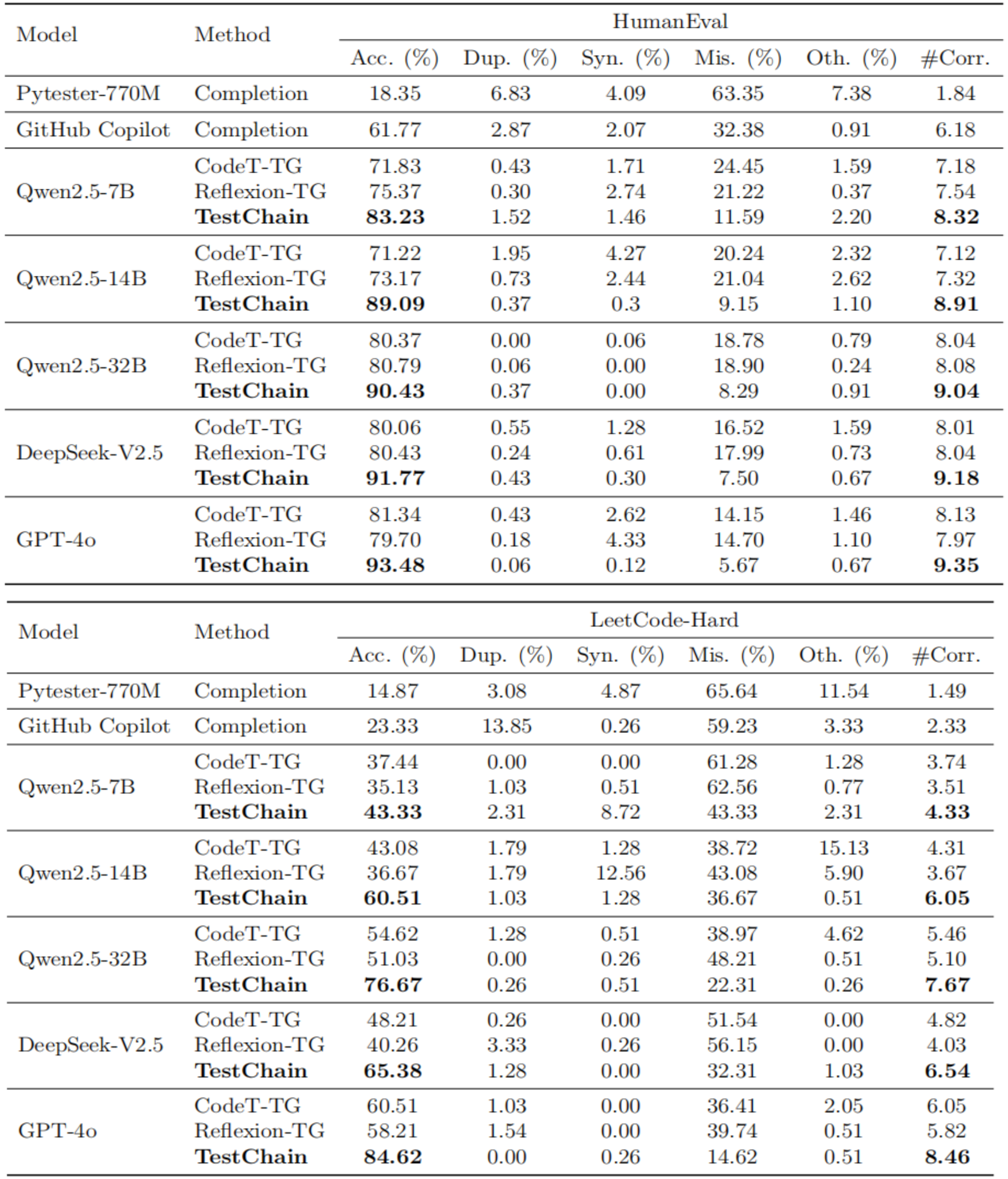
Experiments

- Datasets
  - HumanEval(Chen et al., 2021) (Simple Functions), LeetCode-Hard(Shinn et al., 2023) (Complex Functions)
- Baselines
  - Pytester-770M
  - **CodeT-TG:** Test case generation module of CodeT
  - **Reflexion-TG:** Test case genertion module of Reflexion
- LLMs
  - Qwen2.5-7B, 14B, 32B, DeepSeek-V2.5, GPT-4o

Experiments

- **RQ1: Test Case Accuracy**
  - **Acc.:** Outperform baselines across all LLMs and datasets
  - **Mis.:** Drops significantly
  - **Syn. & Dup. & Oth.:** Rates remain low

TestChain generates more correct test cases than baselines.



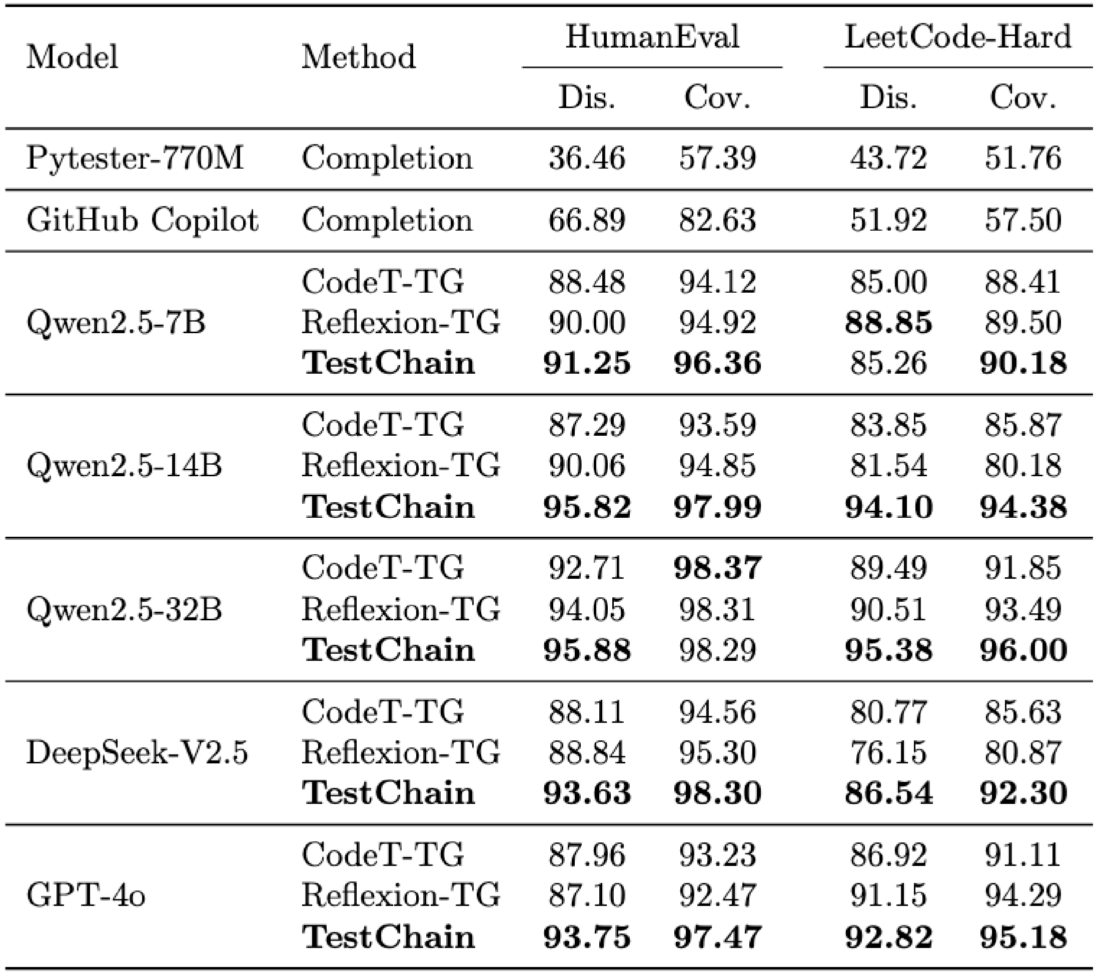
Model	Method	HumanEval					
		Acc. (%)	Dup. (%)	Syn. (%)	Mis. (%)	Oth. (%)	#Corr.
Pytester-770M	Completion	18.35	6.83	4.09	63.35	7.38	1.84
GitHub Copilot	Completion	61.77	2.87	2.07	32.38	0.91	6.18
Qwen2.5-7B	CodeT-TG	71.83	0.43	1.71	24.45	1.59	7.18
	Reflexion-TG	75.37	0.30	2.74	21.22	0.37	7.54
	TestChain	**83.23**	1.52	1.46	11.59	2.20	**8.32**
Qwen2.5-14B	CodeT-TG	71.22	1.95	4.27	20.24	2.32	7.12
	Reflexion-TG	73.17	0.73	2.44	21.04	2.62	7.32
	TestChain	**89.09**	0.37	0.3	9.15	1.10	**8.91**
Qwen2.5-32B	CodeT-TG	80.37	0.00	0.06	18.78	0.79	8.04
	Reflexion-TG	80.79	0.06	0.00	18.90	0.24	8.08
	TestChain	**90.43**	0.37	0.00	8.29	0.91	**9.04**
DeepSeek-V2.5	CodeT-TG	80.06	0.55	1.28	16.52	1.59	8.01
	Reflexion-TG	80.43	0.24	0.61	17.99	0.73	8.04
	TestChain	**91.77**	0.43	0.30	7.50	0.67	**9.18**
GPT-4o	CodeT-TG	81.34	0.43	2.62	14.15	1.46	8.13
	Reflexion-TG	79.70	0.18	4.33	14.70	1.10	7.97
	TestChain	**93.48**	0.06	0.12	5.67	0.67	**9.35**

Model	Method	LeetCode-Hard					
		Acc. (%)	Dup. (%)	Syn. (%)	Mis. (%)	Oth. (%)	#Corr.
Pytester-770M	Completion	14.87	3.08	4.87	65.64	11.54	1.49
GitHub Copilot	Completion	23.33	13.85	0.26	59.23	3.33	2.33
Qwen2.5-7B	CodeT-TG	37.44	0.00	0.00	61.28	1.28	3.74
	Reflexion-TG	35.13	1.03	0.51	62.56	0.77	3.51
	TestChain	**43.33**	2.31	8.72	43.33	2.31	**4.33**
Qwen2.5-14B	CodeT-TG	43.08	1.79	1.28	38.72	15.13	4.31
	Reflexion-TG	36.67	1.79	12.56	43.08	5.90	3.67
	TestChain	**60.51**	1.03	1.28	36.67	0.51	**6.05**
Qwen2.5-32B	CodeT-TG	54.62	1.28	0.51	38.97	4.62	5.46
	Reflexion-TG	51.03	0.00	0.26	48.21	0.51	5.10
	TestChain	**76.67**	0.26	0.51	22.31	0.26	**7.67**
DeepSeek-V2.5	CodeT-TG	48.21	0.26	0.00	51.54	0.00	4.82
	Reflexion-TG	40.26	3.33	0.26	56.15	0.00	4.03
	TestChain	**65.38**	1.28	0.00	32.31	1.03	**6.54**
GPT-4o	CodeT-TG	60.51	1.03	0.00	36.41	2.05	6.05
	Reflexion-TG	58.21	1.54	0.00	39.74	0.51	5.82
	TestChain	**84.62**	0.00	0.26	14.62	0.51	**8.46**

Experiments

- **RQ2: Test Case Effectiveness**
  - **Two metrics**
    - **Error Program Discrimination (Dis.):** The percentage of error programs detected (Similar to mutation testing)
    - **Line Coverage (Cov.):** Covered lines / Total lines
  - **Experimental Results**
    - TestChain achieved the highest Dis. and Cov. in most cases

TestChain produces test cases that better catch faulty programs and execute more code.

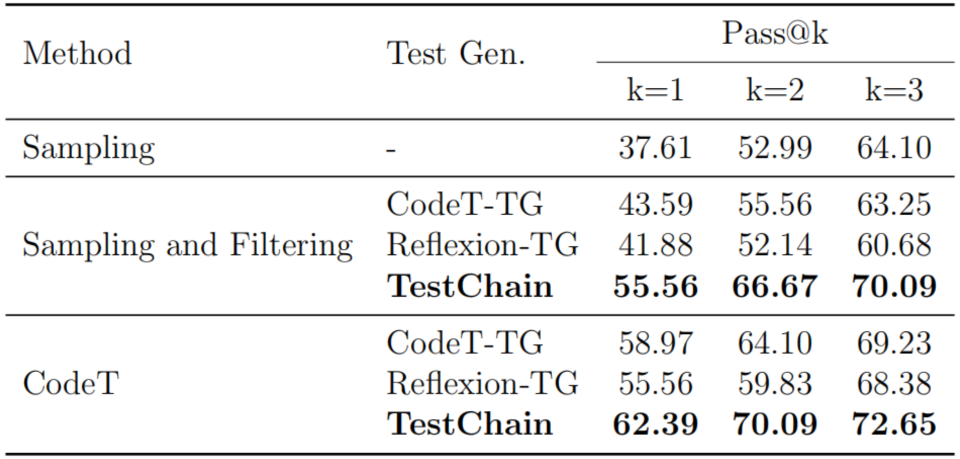


Model	Method	HumanEval		LeetCode-Hard	
		Dis.	Cov.	Dis.	Cov.
Pytester-770M	Completion	36.46	57.39	43.72	51.76
GitHub Copilot	Completion	66.89	82.63	51.92	57.50
Qwen2.5-7B	CodeT-TG	88.48	94.12	85.00	88.41
	Reflexion-TG	90.00	94.92	**88.85**	89.50
	TestChain	**91.25**	**96.36**	85.26	**90.18**
Qwen2.5-14B	CodeT-TG	87.29	93.59	83.85	85.87
	Reflexion-TG	90.06	94.85	81.54	80.18
	TestChain	**95.82**	**97.99**	**94.10**	**94.38**
Qwen2.5-32B	CodeT-TG	92.71	**98.37**	89.49	91.85
	Reflexion-TG	94.05	98.31	90.51	93.49
	TestChain	**95.88**	98.29	**95.38**	**96.00**
DeepSeek-V2.5	CodeT-TG	88.11	94.56	80.77	85.63
	Reflexion-TG	88.84	95.30	76.15	80.87
	TestChain	**93.63**	**98.30**	**86.54**	**92.30**
GPT-4o	CodeT-TG	87.96	93.23	86.92	91.11
	Reflexion-TG	87.10	92.47	91.15	94.29
	TestChain	**93.75**	**97.47**	**92.82**	**95.18**

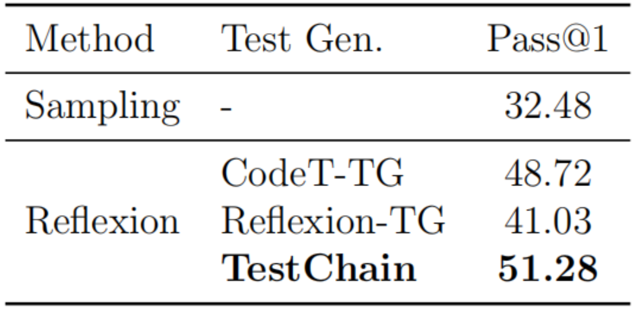
Experiments

- **RQ3: Improving Code Generation**
  - **Case 1:** Evaluate and selection based methods, generate 50 programs
    - Sampling, Sampling+Filtering(Li et al., 2022), CodeT(Chen et al., 2022b)
  - **Case 2:** Repair-based methods
    - **Sampling:** Generate 5 programs
    - **Reflexion**(Shinn et al., 2023)**:** Generate 1 program and fix 4 times

TestChain improves performance of code generation methods.



Method	Test Gen.	Pass@k		
		k=1	k=2	k=3
Sampling	-	37.61	52.99	64.10
Sampling and Filtering	CodeT-TG	43.59	55.56	63.25
	Reflexion-TG	41.88	52.14	60.68
	TestChain	**55.56**	**66.67**	**70.09**
CodeT	CodeT-TG	58.97	64.10	69.23
	Reflexion-TG	55.56	59.83	68.38
	TestChain	**62.39**	**70.09**	**72.65**

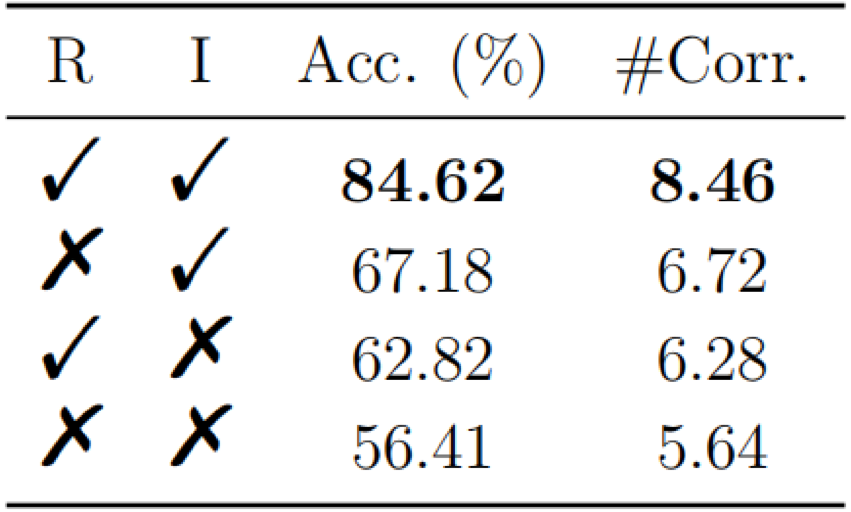


Method	Test Gen.	Pass@1
Sampling	-	32.48
Reflexion	CodeT-TG	48.72
	Reflexion-TG	41.03
	TestChain	**51.28**

Experiments

- RQ4: Ablation Study on Code-Assisted Flow
  - Two variants
    - R: Reasoning process.
    - I: Code interpreter interaction process.
  - Experimental Results
    - Full Setup (+R, +I) achieves the best performance

Confirms that both the reasoning process and code interpreter interaction process are essential. **The combination is key.**



R	I	Acc. (%)	#Corr.
✓	✓	**84.62**	**8.46**
✗	✓	67.18	6.72
✓	✗	62.82	6.28
✗	✗	56.41	5.64

Discussion

IEEE Congress on Evolutionary Computation
CEC 2025
Hangzhou

- **Current Limitations**
  - **Language Scope:** Only Python, function-level
  - **Compute Overhead:** Interpreter calls increase latency in large-scale pipelines
- **Future Directions**
  - **Multi-Language Extension:** Java, C++, Go
  - **Prompt Optimization:** Automated prompt tuning
  - **Hierarchical Test Suites:** Generate not only function tests but also integration and system-level tests

References

- **(Chen et al., 2022a)** Chen B, Zhang F, Nguyen A, et al. Codet: Code generation with generated tests[J]. arXiv preprint arXiv:2207.10397, 2022.
- **(Shinn et al., 2023)** Shinn N, Cassano F, Gopinath A, et al. Reflexion: Language agents with verbal reinforcement learning[J]. Advances in Neural Information Processing Systems, 2023, 36: 8634-8652.
- **(Takerngsaksiri et al., 2025)** Takerngsaksiri W, Charakorn R, Tantithamthavorn C, et al. Pytester: Deep reinforcement learning for text-to-testcase generation[J]. Journal of Systems and Software, 2025, 224: 112381.
- **(Chen et al., 2021)** Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code[J]. arXiv preprint arXiv:2107.03374, 2021.
- **(Li et al., 2022)** Li Y, Choi D, Chung J, et al. Competition-level code generation with alphacode[J]. Science, 2022, 378(6624): 1092-1097.
- **(Chen et al., 2022b)** Chen B, Zhang F, Nguyen A, et al. Codet: Code generation with generated tests[J]. arXiv preprint arXiv:2207.10397, 2022.

Thanks

Any Question?