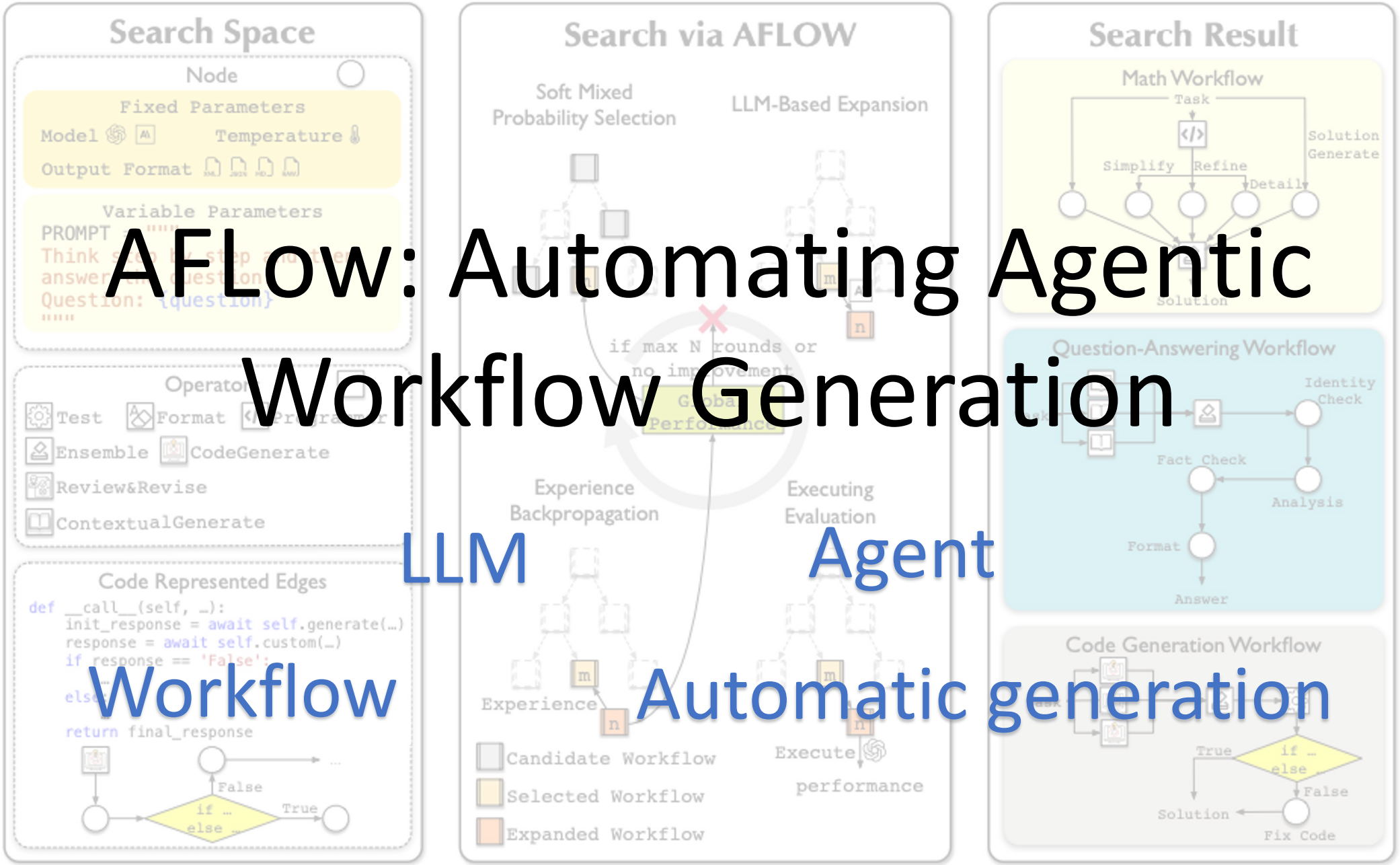


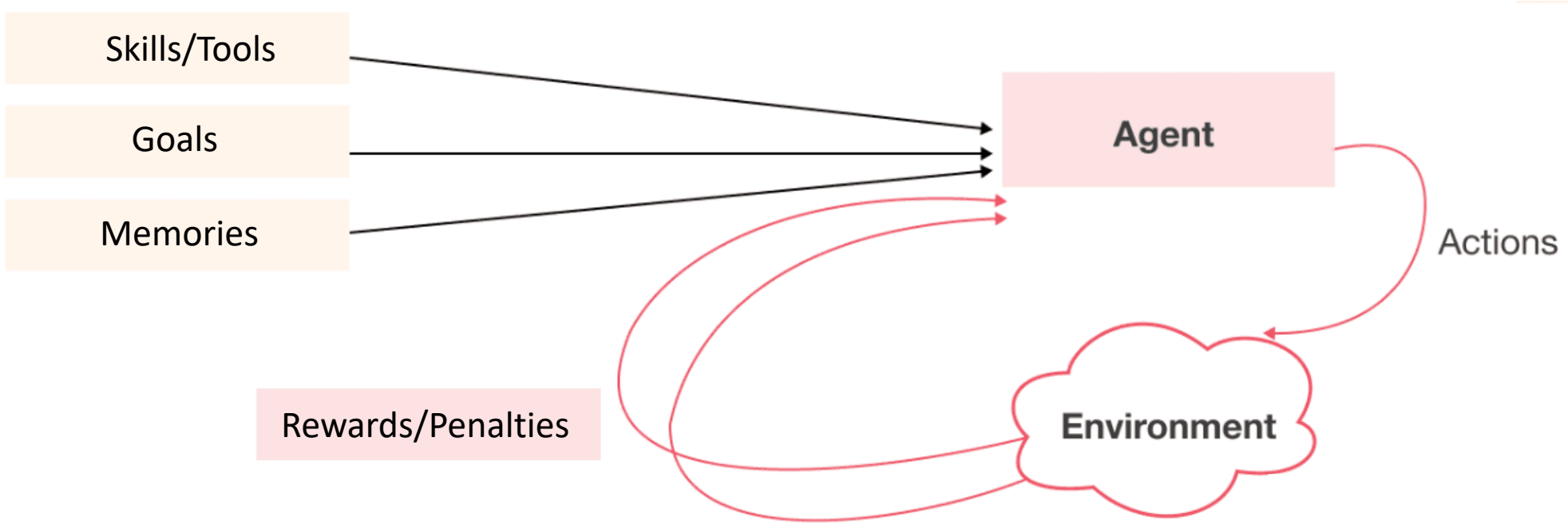

# AFlow: Automating Agentic Workflow Generation

LLM Agent

Workflow Automatic generation

# What is an Agent?

An Agents is an **autonomous entity**, capable to interact with an environment, receive rewards or penalties, and learn an optimal policy for decision-making. Originate from Reinforcement Learning!



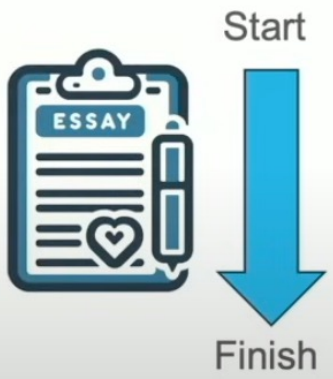
# What is Agentic Workflow

DATA CLOUD SUMMIT 24

## LLM-based agents

Non-agentic workflow (zero-shot):

Please type out an essay on topic X from start to finish in one go, without using backspace.




Agentic workflow:

Write an essay outline on topic X

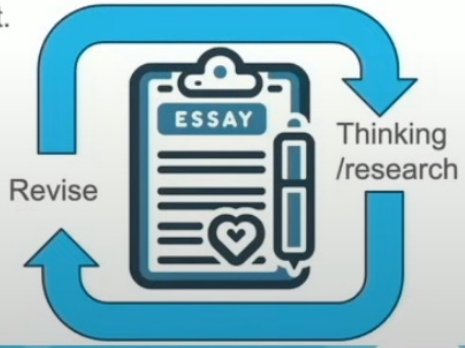
Do you need any web research?

Write a first draft.

Consider what parts need revision or more research.

Revise your draft.

....




# Pre-defined agentic workflow

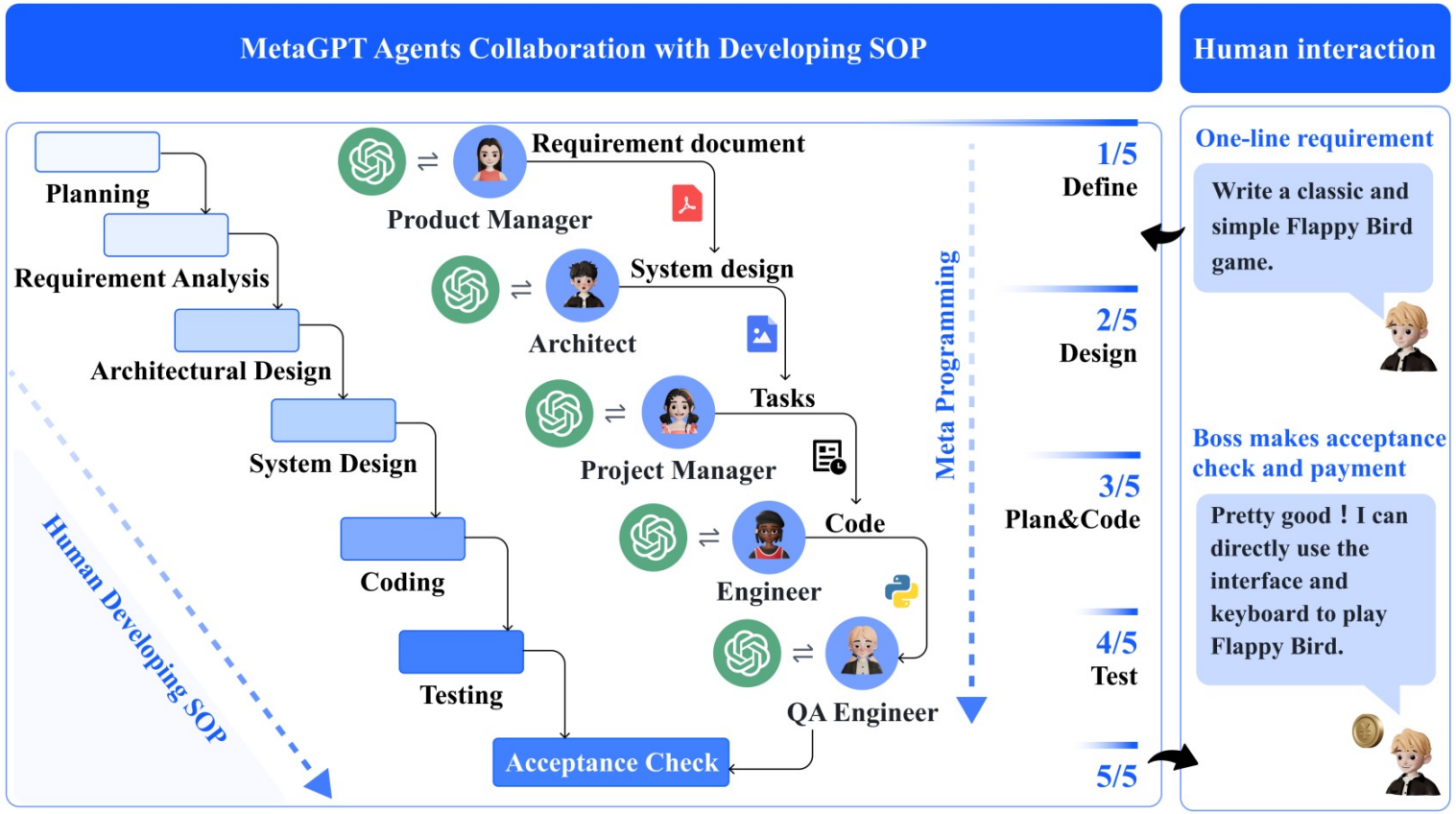



Figure 1: **The software development SOPs between MetaGPT and real-world human teams.** In software engineering, SOPs promote collaboration among various roles. MetaGPT showcases its ability to decompose complex tasks into specific actionable procedures assigned to various roles (e.g., Product Manager, Architect, Engineer, etc.).

# Automatic generated agentic workflow

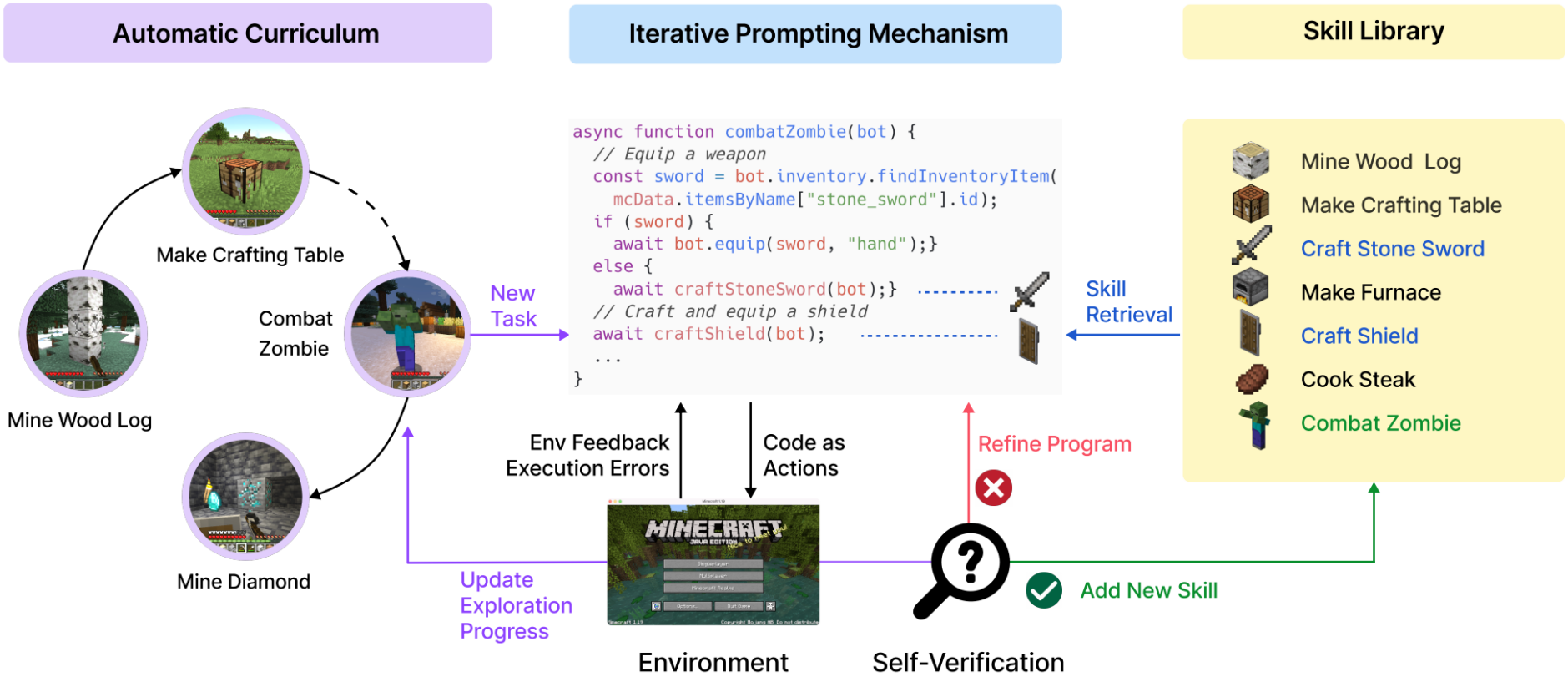
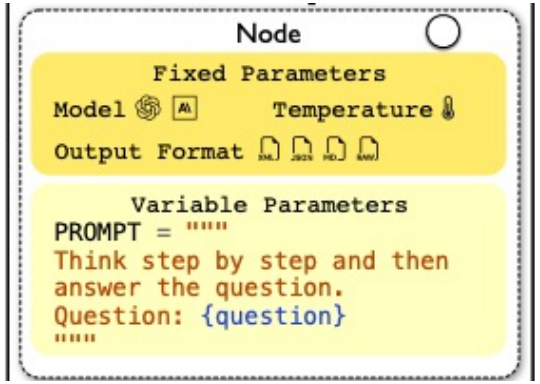



Figure 2: VOYAGER consists of three key components: an automatic curriculum for open-ended exploration, a skill library for increasingly complex behaviors, and an iterative prompting mechanism that uses code as action space.

Existing work has manually discovered numerous effective agentic workflows, but it’s challenging to exhaust various tasks across different domains

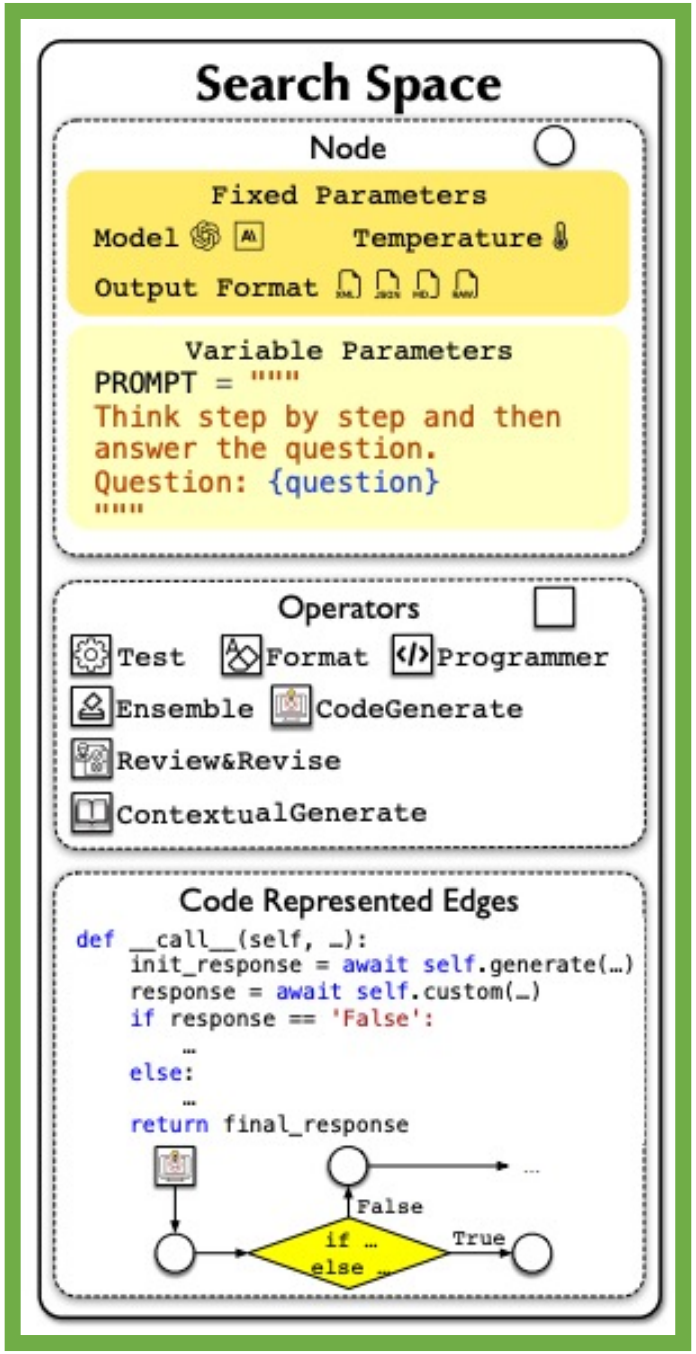
# AFlow: Agentic Workflow

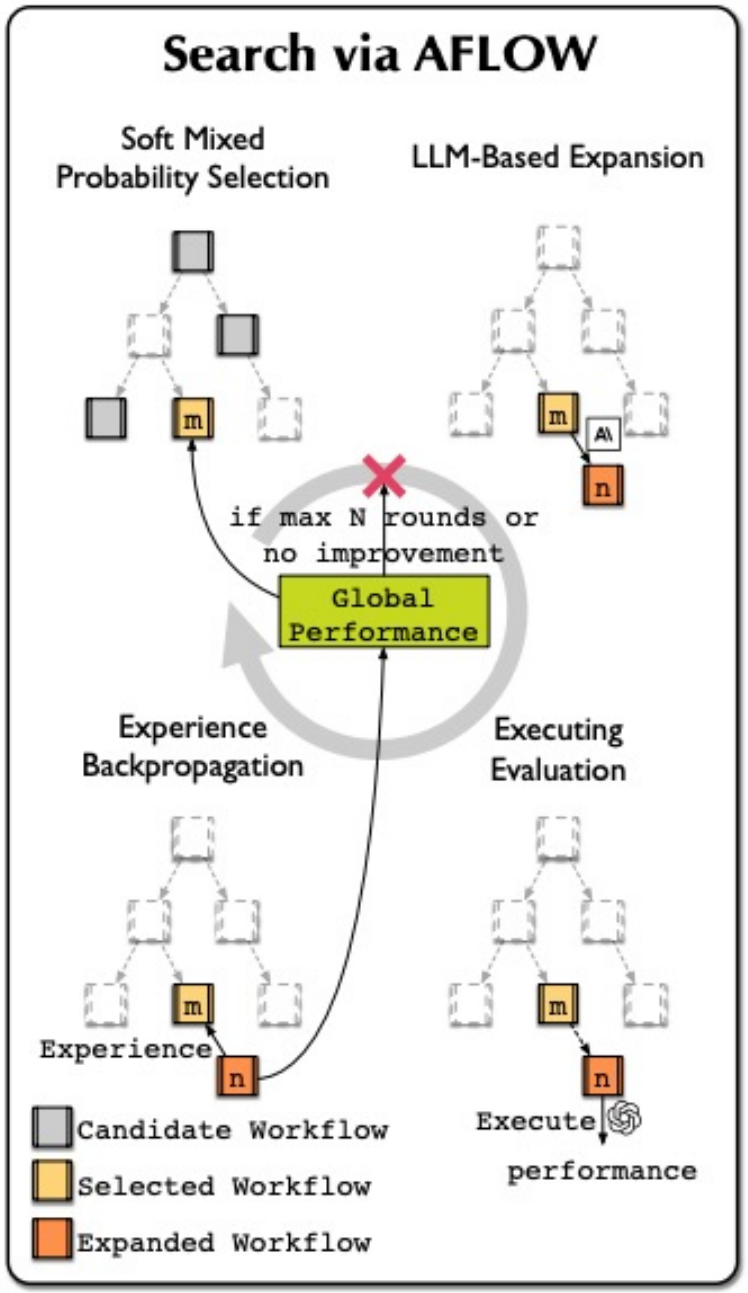


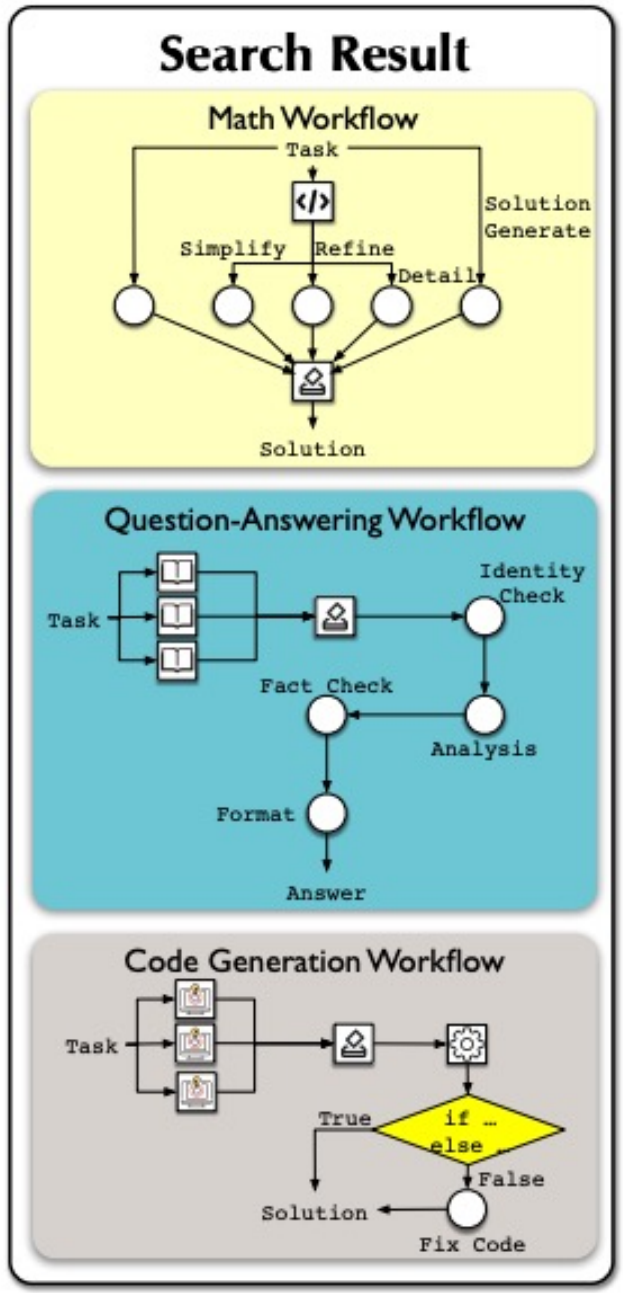

**Agentic Workflow** We define an agentic workflow W as a series of LLM-invoking nodes connected by edges to define the exection orders, denoted as $\mathcal{N} = \{N_1, N_2, \ldots, N_i \ldots\}$. Each node $N_i$ represents a specific operation performed by an LLM and is characterized by the following parameters.

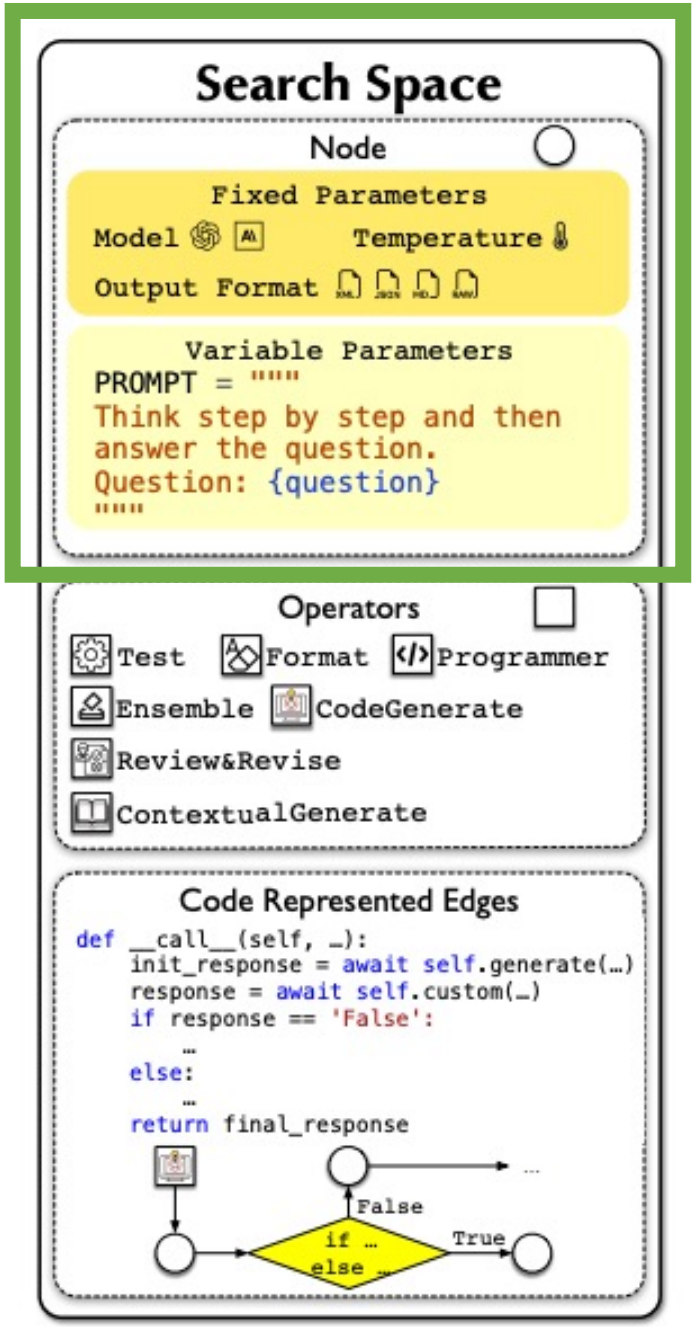
- **Model** $M$: The specific language model invoked at node $N_i$.
- **Prompt** $P$: The input or task description provided to the model at each node.
- **Temperature** $\tau$: A parameter controlling the randomness of the LLM's output at node $N_i$.
- **Output format** $F$: The format in which the model's output is structured (*e.g.*, xml, json, markdown, raw). The node in workflow should provide different output formats, inspired by the Tam et al. (2024).

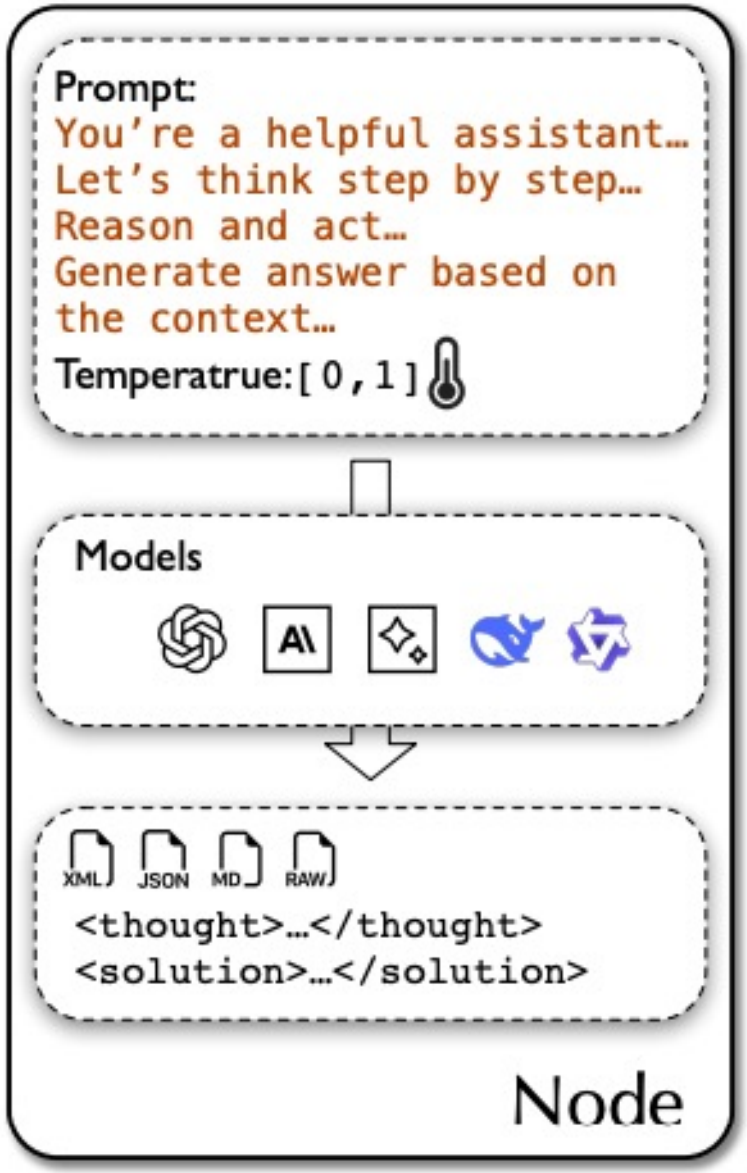
Monte Carlo Tree Search

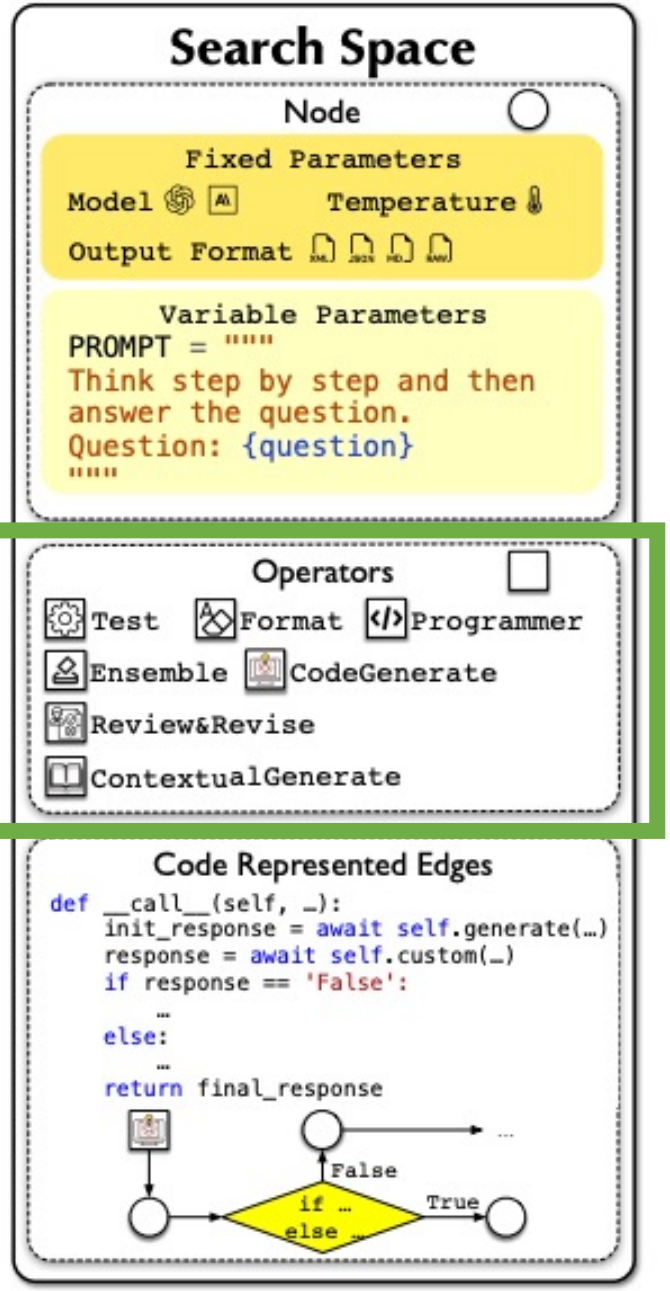


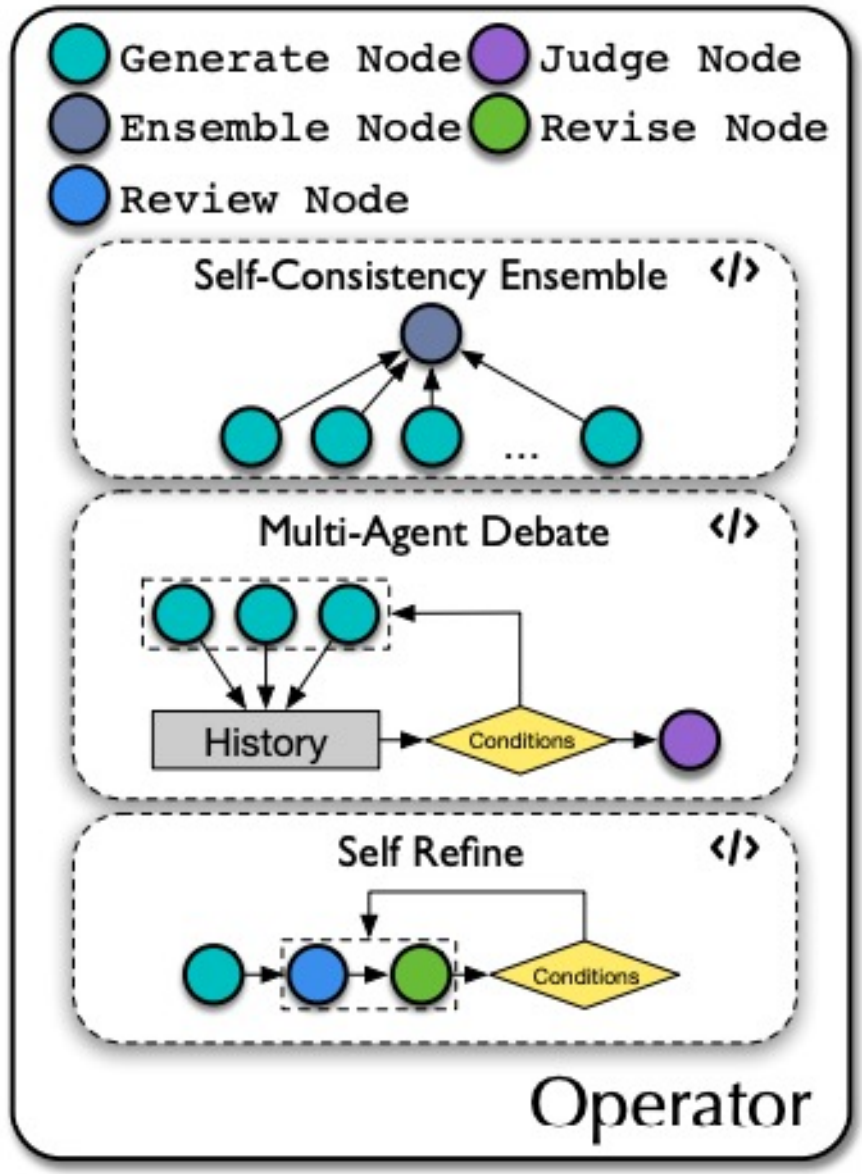



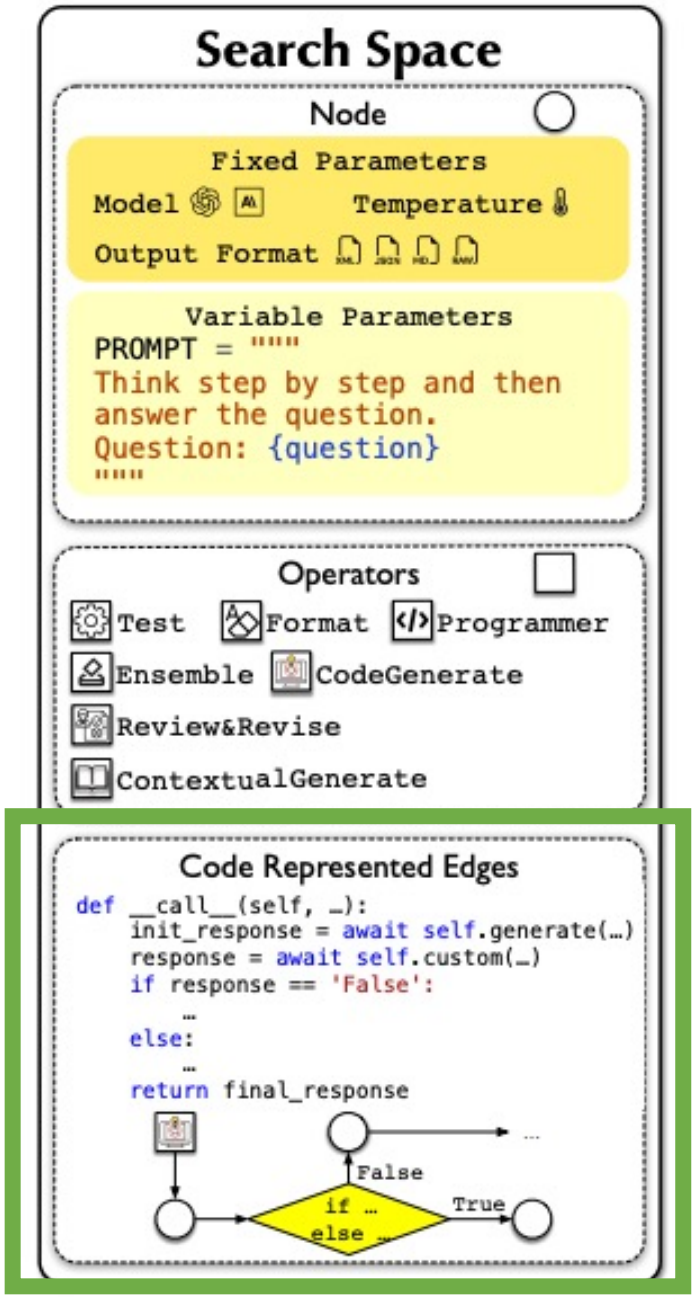

Search Space
Node
Fixed Parameters
Model
Temperature
Output Format
Variable Parameters
PROMPT = """
Think step by step and then
answer the question.
Question: {question}
"""
Operators
Test
Format
Programmer
Ensemble
CodeGenerate
Review&Revise
ContextualGenerate
Code Represented Edges
def __call__(self, …):
init_response = await self.generate(…)
response = await self.custom(…)
if response == 'False':
…
else:
…
return final_response
False
if …
else …
True

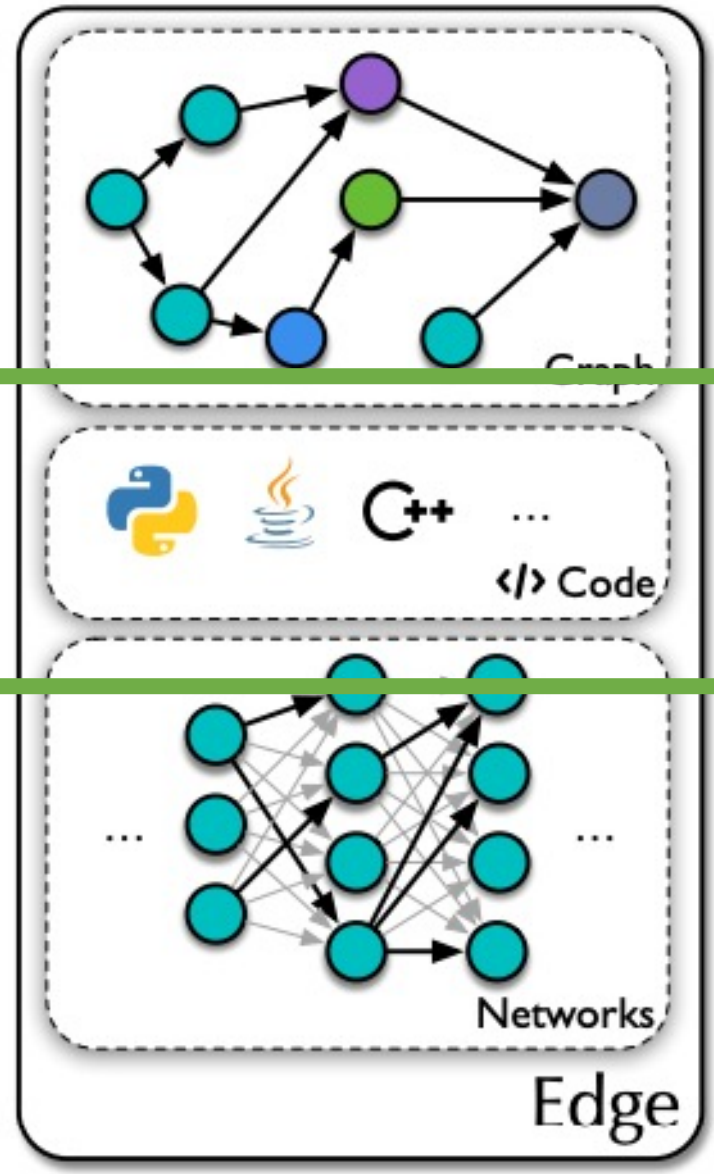

Prompt:
You're a helpful assistant…
Let's think step by step…
Reason and act…
Generate answer based on
the context…
Temperatrue:[0,1]
Models
XML
JSON
MD
RAW
<thought>…</thought>
<solution>…</solution>
Node




Providing **predefined operators** can effectively enhance the search efficiency of AFLOW. We implement six common operator structures, including: Generate (Contextual, Code), Format, Review & Revise, Ensemble, Test, and Programmer.




A flexible structure representing hierarchical, sequential, or parallel relationships between nodes

Express linear sequences, conditional logic, loops, and incorporate graph or network structures, offering the most precise control

Represent complex, non-linear relationships between nodes, allowing for adaptive and learnable workflows

# Search Space

Node

Fixed Parameters

Model  Temperature

Output Format

Variable Parameters

```
PROMPT = """
Think step by step and then
answer the question.
Question: {question}
"""
```

Operators

Test  Format  Programmer

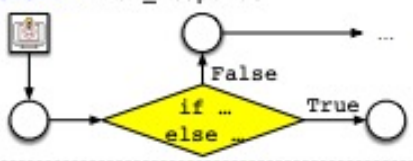
Ensemble  CodeGenerate

Review&Revise

ContextualGenerate

Code Represented Edges

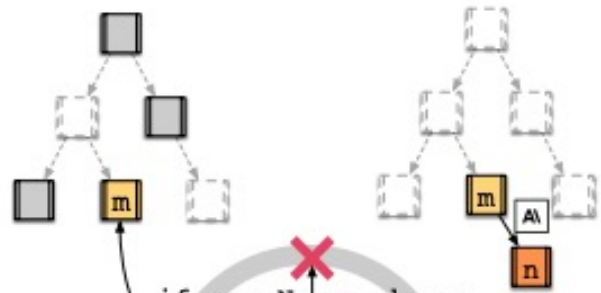
```
def __call__(self, …):
    init_response = await self.generate(…)
    response = await self.custom(…)
    if response == 'False':
        …
    else:
        …
    return final_response
```




# Search via AFLOW

Soft Mixed Probability Selection

LLM-Based Expansion

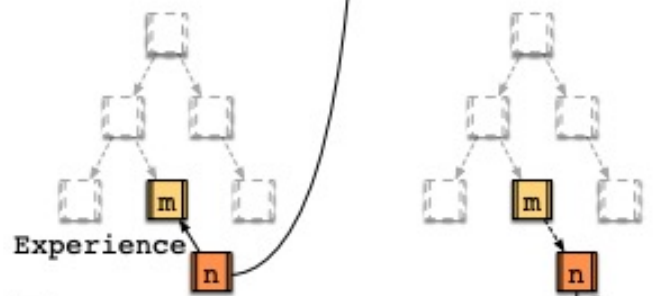



if max N rounds or no improvement

Global Performance

Experience Backpropagation

Executing Evaluation




performance

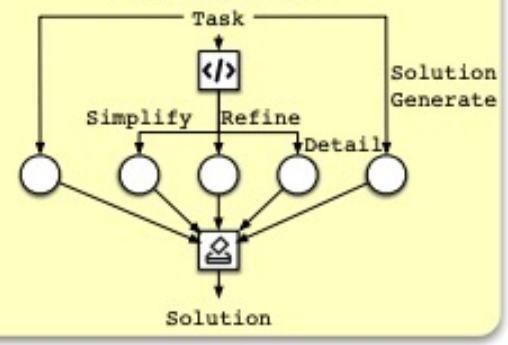
Candidate Workflow

Selected Workflow

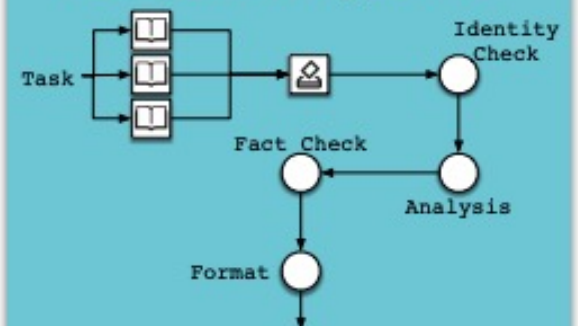
Expanded Workflow

# Search Result

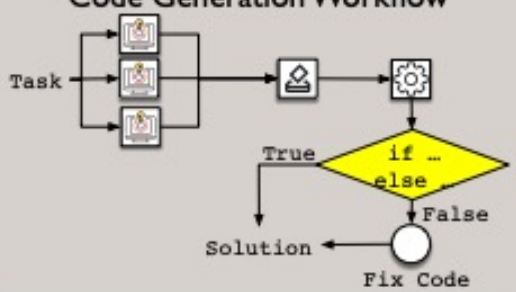
Math Workflow

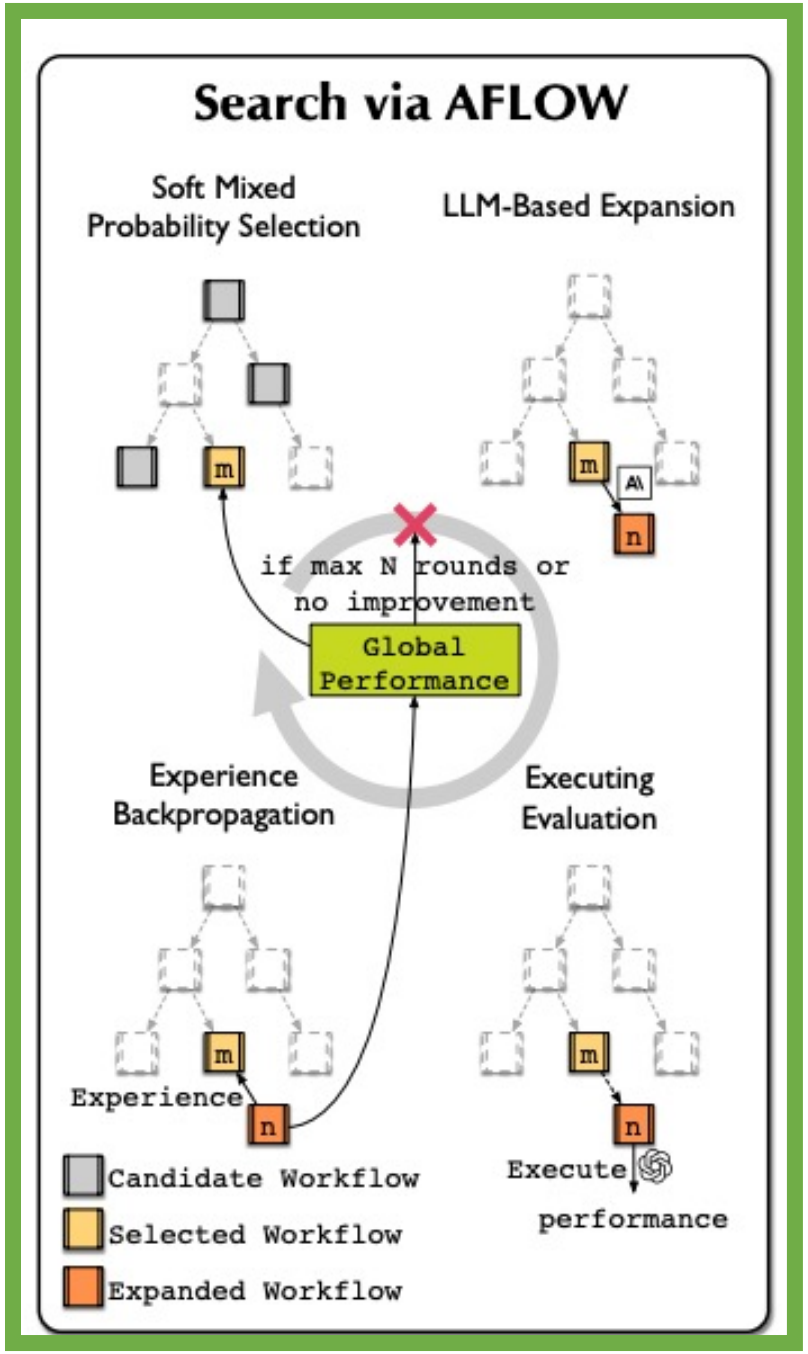



Question-Answering Workflow




Code Generation Workflow



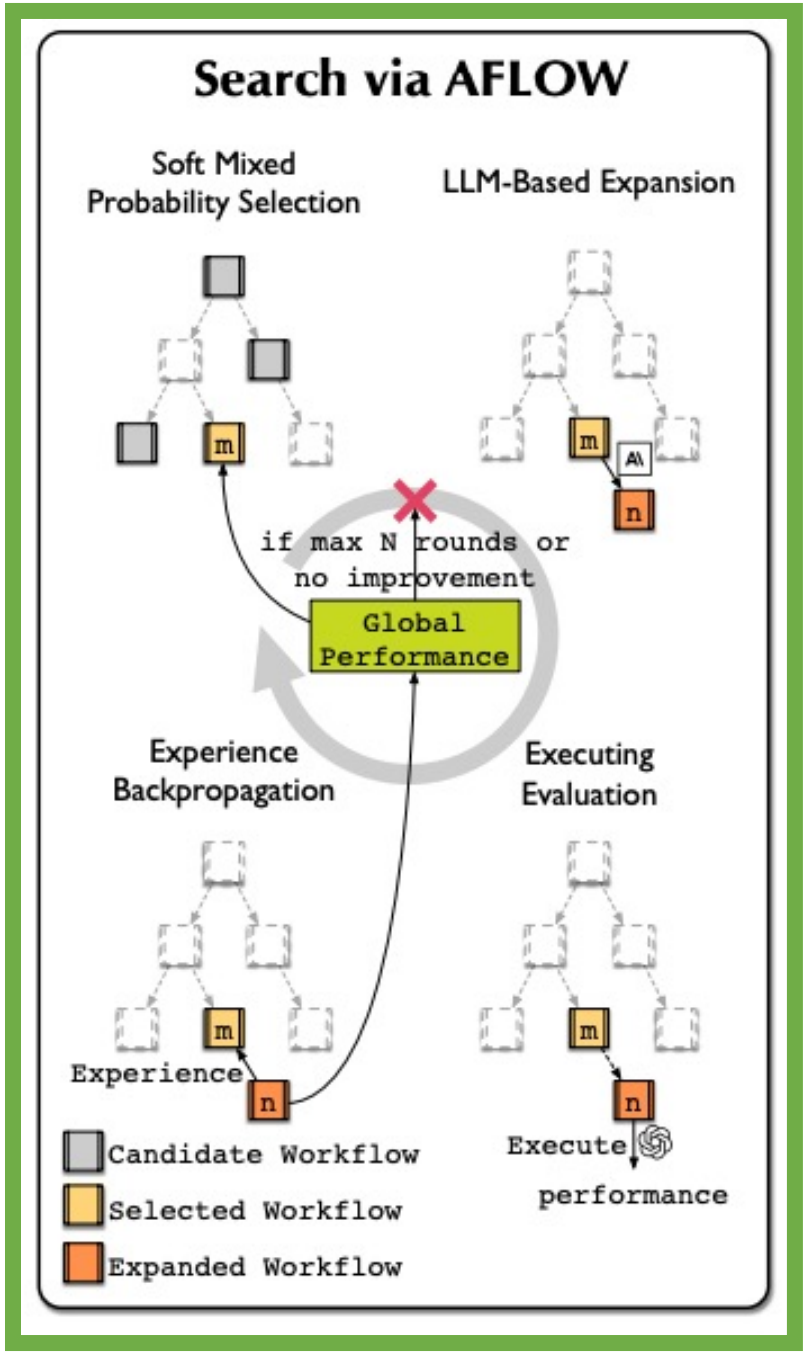



# Soft Mixed Probability Selection

$$P_{\text{mixed}}(i) = \underbrace{\lambda \cdot \frac{1}{n}}_{\text{Exploration}} + \underbrace{(1-\lambda) \cdot \frac{\exp(\alpha \cdot (s_i - s_{\max}))}{\sum_{j=1}^{n} \exp(\alpha \cdot (s_j - s_{\max}))}}_{\text{Exploitation}},$$

- **Uniform Component**:
  $\lambda \cdot \frac{1}{n}$ means every workflow gets at least a small chance (exploration).
- **Softmax Component**:
  $(1-\lambda) \cdot \text{Softmax}(s_i - s_{\max})$ heavily favors higher-performing workflows (exploitation).
  Subtracting $s_{\max}$ ensures numerical stability.

- $n$: number of candidate workflows.
- $s_i$: score (performance) of workflow $i$.
- $s_{\max}$: the highest score among all candidates.
- $\alpha$: controls how strongly scores affect probabilities (default = 0.4).
- $\lambda$: controls how much randomness to inject (default = 0.2).



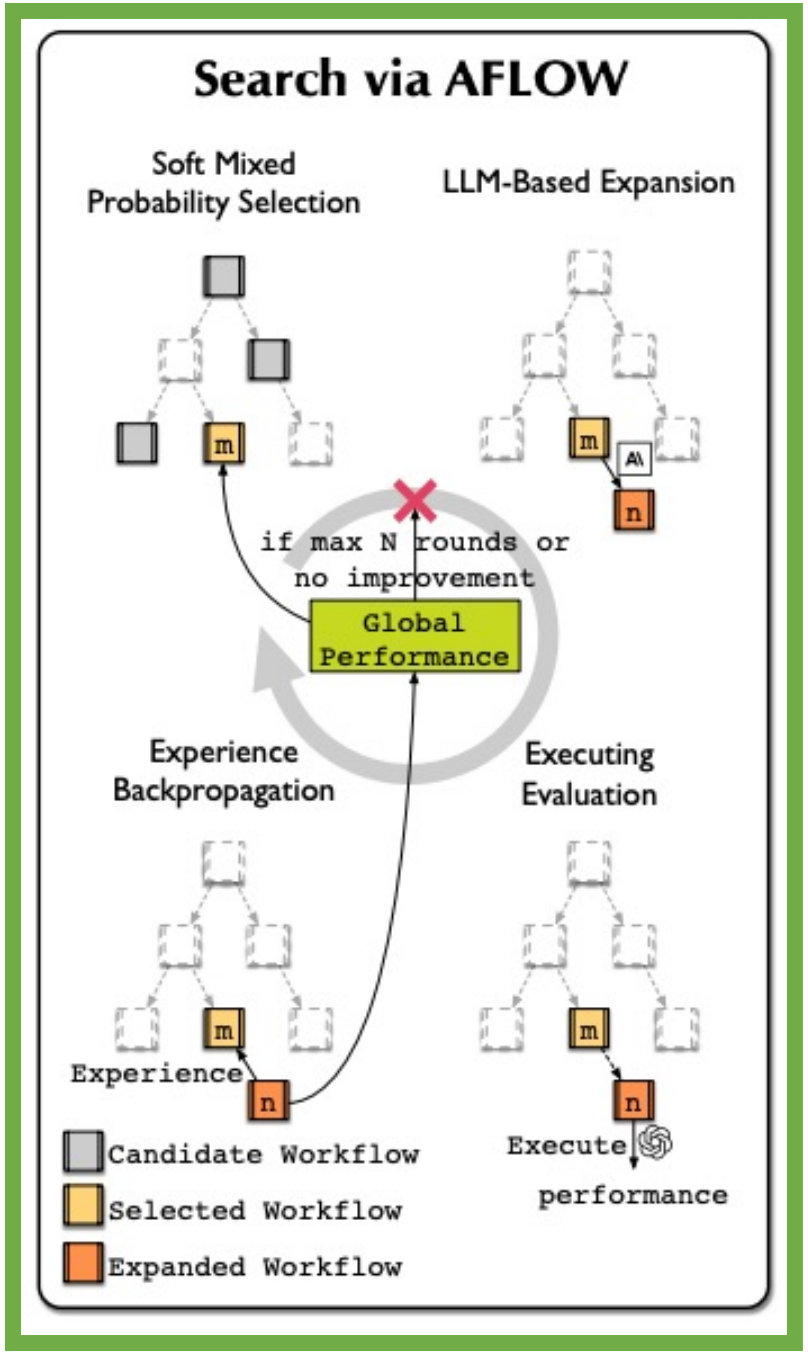

# LLM-based Expansion

Search space:
Node – prompts
Operator selection
Edge – codes

## A.1 LLM Based Expansion: Prompt for LLM Optimizer

**Workflow optimize prompt**

```
PROMPT = """You are building a Graph and corresponding Prompt to jointly solve {type}
↪  problems. Referring to the given graph and prompt, which forms a basic example of
↪  a {type} solution approach, please reconstruct and optimize them. You can add,
↪  modify, or delete nodes, parameters, or prompts. Include your single modification
↪  in XML tags in your reply. Ensure they are complete and correct to avoid runtime
↪  failures. When optimizing, you can incorporate critical thinking methods like
↪  review, revise, ensemble (generating multiple answers through different/similar
↪  prompts, then voting/integrating/checking the majority to obtain a final answer),
↪  selfAsk, etc. Consider Python's loops (for, while, list comprehensions),
↪  conditional statements (if-elif-else, ternary operators), or machine learning
↪  techniques (e.g., linear regression, decision trees, neural networks,
↪  clustering). The graph complexity should not exceed 10. Use logical and control
↪  flow (IF-ELSE, loops) for a more enhanced graphical representation.Ensure that
↪  all the prompts required by the current graph from prompt_custom are
↪  included.Exclude any other prompts. Output the modified graph and all the
↪  necessary Prompts in prompt_custom (if needed).The prompt you need to generate is
↪  only the one used in `prompt_custom.XXX` within Custom. Other methods already
↪  have built-in prompts and are prohibited from being generated. Only generate
↪  those needed for use in `prompt_custom`; please remove any unused prompts in
↪  prompt_custom. the generated prompt must not contain any placeholders.
↪  Considering information loss, complex graphs may yield better results, but
↪  insufficient information transmission can omit the solution. It's crucial to
↪  include necessary context during the process."""
```



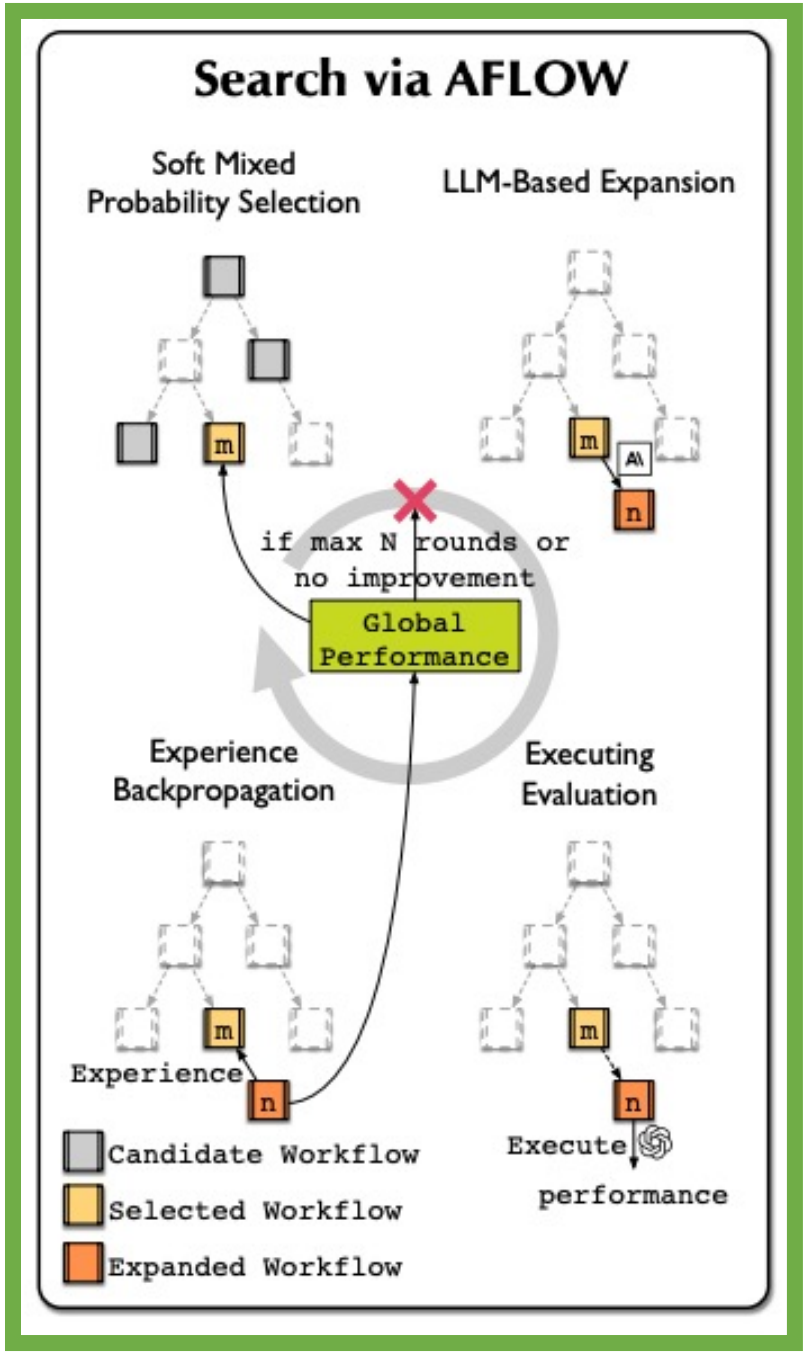

# LLM-based Expansion

Search space:
Node – prompts
Operator selection
Edge – codes

## A.1 LLM Based Expansion: Prompt for LLM Optimizer

**Workflow optimize prompt**

| 🔧 Goal | 💬 Prompt Instruction |
|---|---|
| Improve performance | “reconstruct and optimize…” |
| Be safe | “avoid runtime failures” |
| Use proven strategies | “consider… ensemble, revise, review…” |
| Stay minimal | “include your single modification…” |
| Be precise | “only generate those needed in prompt_custom” |
| Enable learning | “include your change in XML tags” |



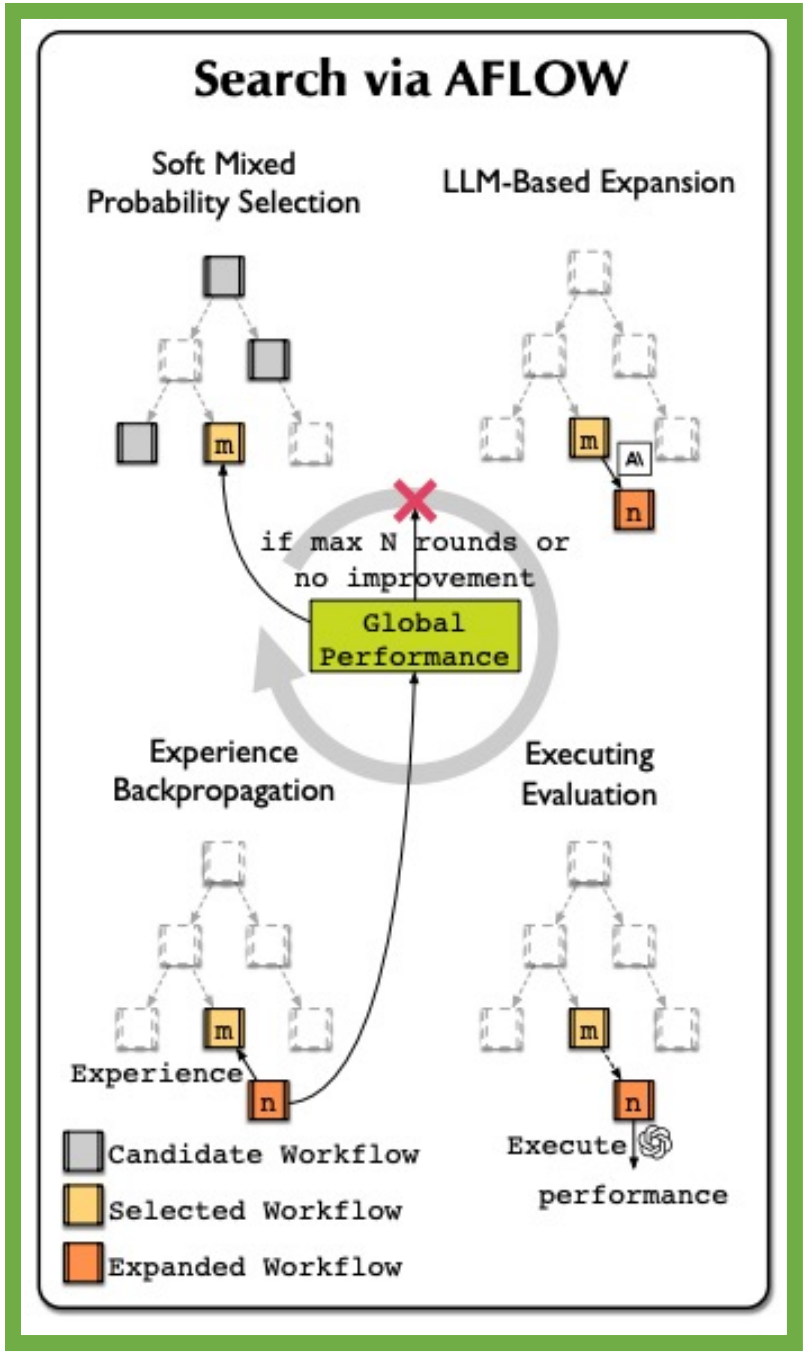

# LLM-based Expansion

Search space:
Node – prompts
Operator selection
Edge – codes

Here's a simplified version of what the LLM gets as a prompt (from Appendix A.1):

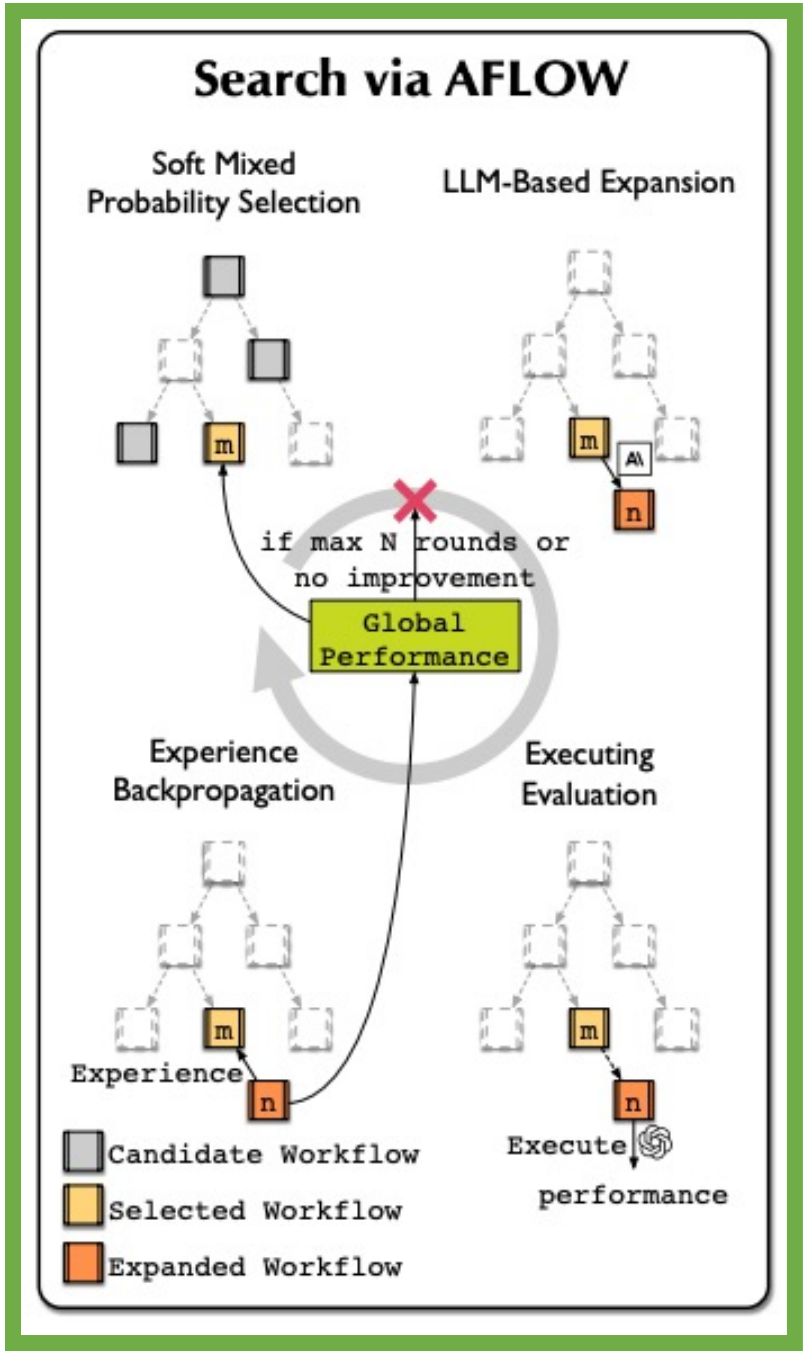
A.

```
text

You're optimizing a workflow for math reasoning.

Base Workflow:
<code block here>

Recent experiences:
- Added Ensemble → score +0.04
- Added Programmer → score +0.05
- Modified prompt for step-by-step → score +0.01
- Changed output format to markdown → score -0.02

Now: propose ONE new change that improves performance.

Rules:
- Use agents like Review, Revise, Ensemble, etc.
- Keep graph ≤ 10 nodes
- Only output valid code and updated prompts
```




# Executing Evaluation

AFLOW directly executes workflows to get feedback due to explicit evaluation functions in reasoning tasks. We test each generated workflow 5 times on the validation set, computing mean and standard deviation.




# Experience Backpropagation

After execution, we record: (1) the workflow's performance, (2) the optimizer's modification of its parent workflow, and (3) optimization success relative to its parent.

This information is stored in experience and propagated back to the parent workflow for LLM-based expansion, while the performance score is added to the global record for selection.

```
Recent experiences:
- Added Ensemble → score +0.04
- Added Programmer → score +0.05
- Modified prompt for step-by-step → score +0.01
- Changed output format to markdown → score -0.02
```

# Search via AFlow

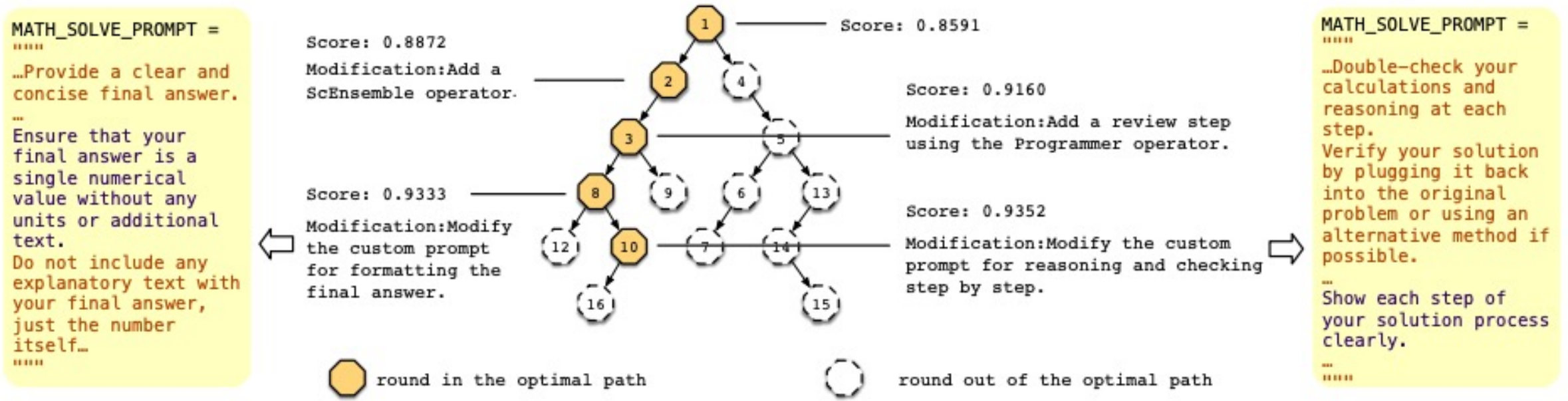
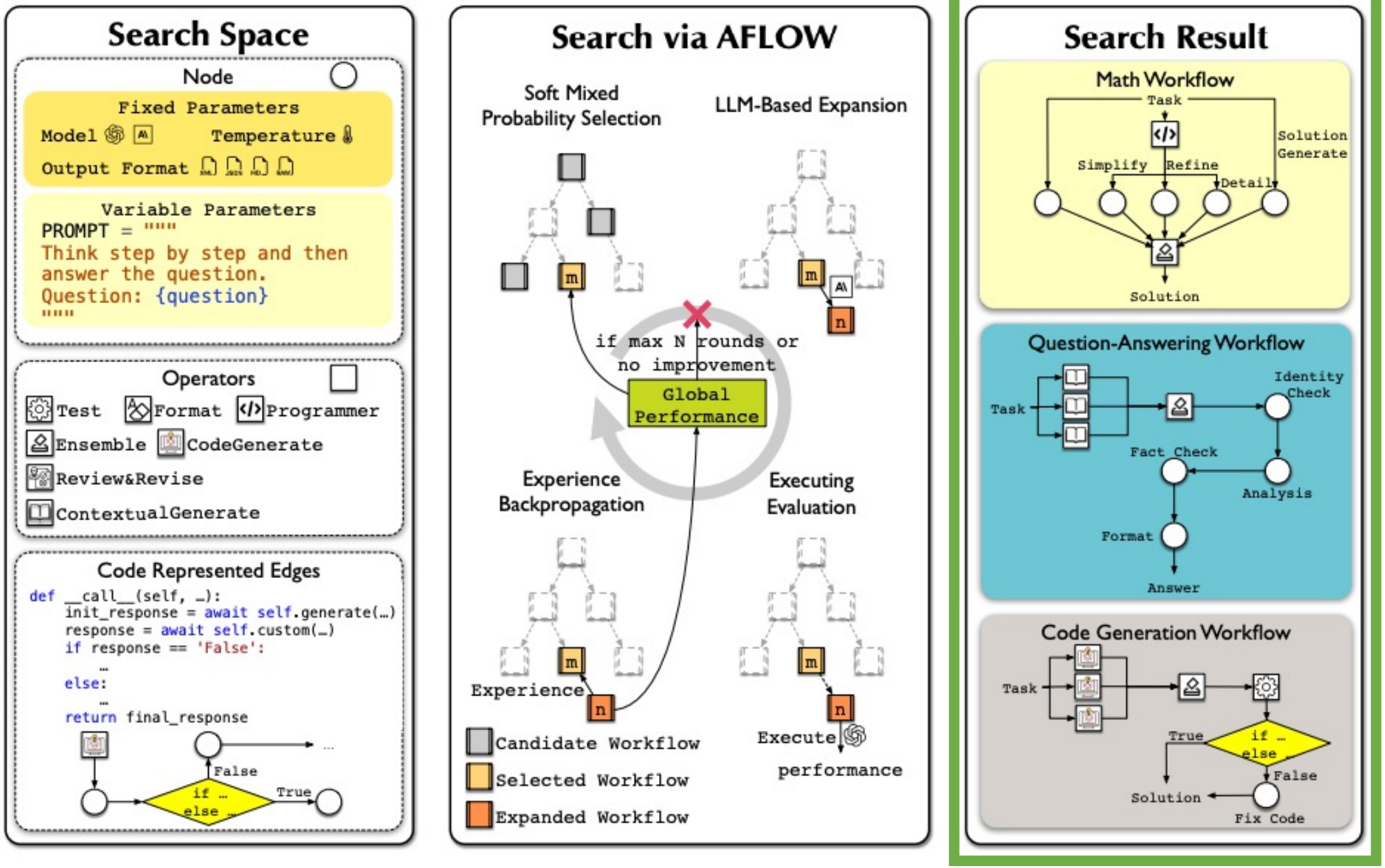
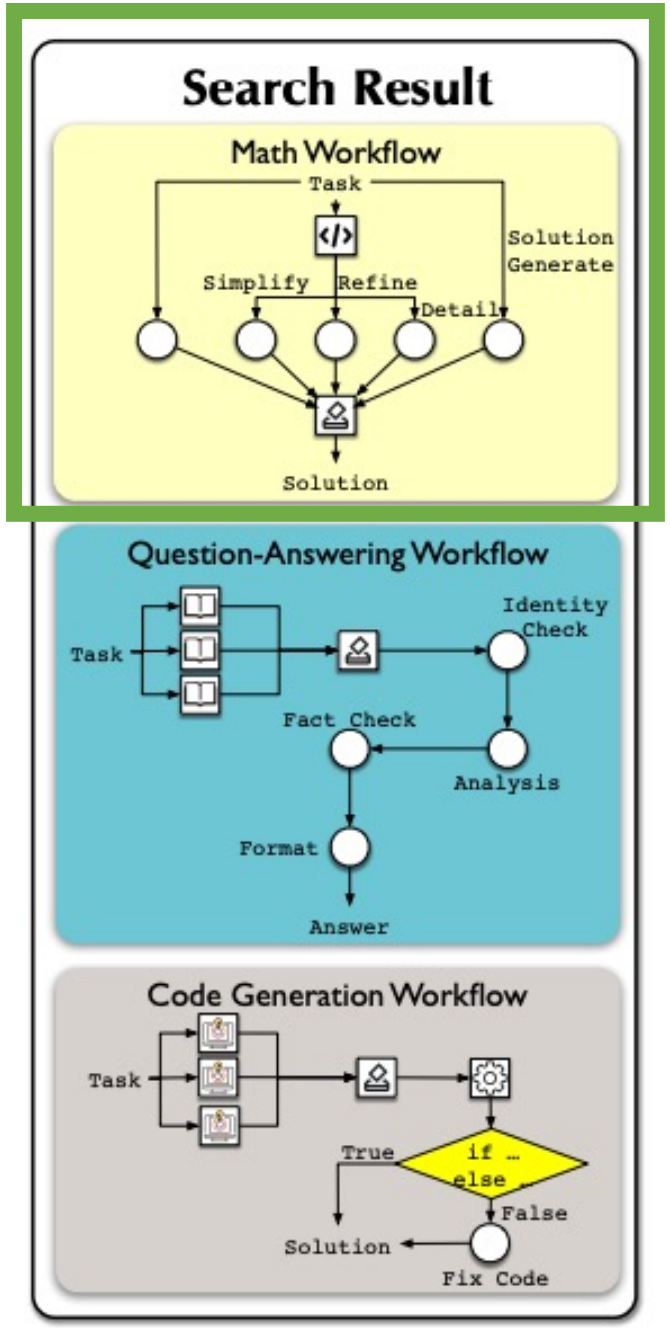
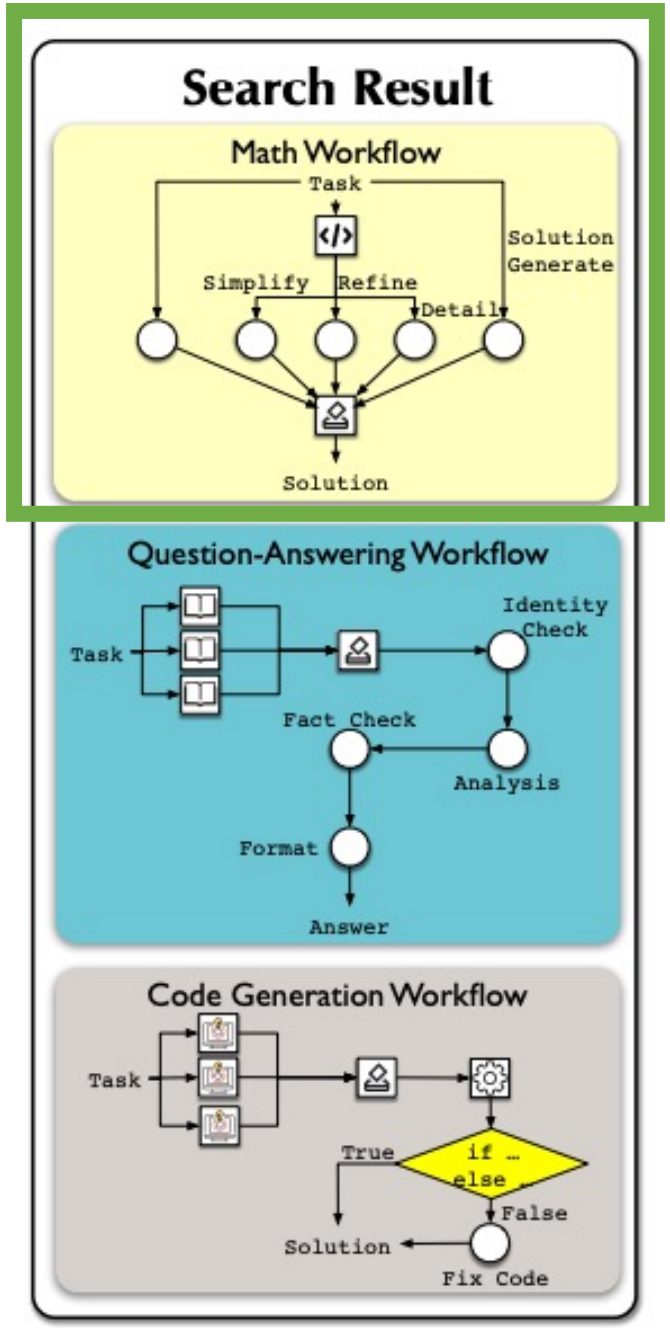



Figure 6: Tree-structured iteration process of AFLOW on GSM8K: We highlight the path from the initial round (round 1) to the best-performing workflow, reporting the score for each node and its modification from the previous node. The purple sections in the prompts on both sides represent the main prompt modifications in this iteration.


Search Space
Node
Fixed Parameters
Model
Temperature
Output Format
Variable Parameters
PROMPT = """
Think step by step and then
answer the question.
Question: {question}
"""
Operators
Test
Format
Programmer
Ensemble
CodeGenerate
Review&Revise
ContextualGenerate
Code Represented Edges
def __call__(self, …):
init_response = await self.generate(…)
response = await self.custom(…)
if response == 'False':
…
else:
…
return final_response
False
True
if …
else …
Search via AFLOW
Soft Mixed
Probability Selection
LLM-Based Expansion
if max N rounds or
no improvement
Global
Performance
Experience
Backpropagation
Executing
Evaluation
Experience
Execute
performance
Candidate Workflow
Selected Workflow
Expanded Workflow
Search Result
Math Workflow
Task
Simplify
Refine
Detail
Solution
Generate
Solution
Question-Answering Workflow
Task
Identity
Check
Fact Check
Analysis
Format
Answer
Code Generation Workflow
Task
True
if …
else …
False
Solution
Fix Code




```
REFINE_ANSWER_PROMPT = """
Given the mathematical problem and the output from the code execution, please provide
↪  a well-formatted and detailed solution. Follow these guidelines:
1. Begin with a clear statement of the problem.
2. Explain the approach and any formulas or concepts used.
3. Show step-by-step calculations, using LaTeX notation for mathematical expressions.
4. Interpret the code output and incorporate it into your explanation.
5. Provide a final answer, enclosed in \boxed{} LaTeX notation.
6. Ensure all mathematical notation is in LaTeX format.
Your response should be comprehensive, mathematically rigorous, and easy to follow.
"""
GENERATE_SOLUTION_PROMPT = """
Please solve the given mathematical problem step by step. Follow these guidelines:
1. State the problem clearly.
2. Outline the approach and any relevant formulas or concepts.
3. Provide detailed calculations, using LaTeX notation for mathematical expressions.
4. Explain each step of your reasoning.
5. Present the final answer enclosed in \boxed{} LaTeX notation.
6. Ensure all mathematical notation is in LaTeX format.
Your solution should be thorough, mathematically sound, and easy to understand.
"""
```




```python
async def __call__(self, problem: str):
    """
    Implementation of the graph
    """
    # Use Programmer to generate and execute Python code
    code_solution = await self.programmer(problem=problem)
    # Use Custom to refine and format the answer
    refined_solution = await self.custom(input=problem + f"\nCode output:
    ↪  {code_solution['output']}", instruction=prompt_custom.REFINE_ANSWER_PROMPT)
    # Generate a detailed step-by-step solution using Custom
    detailed_solution = await self.custom(input=problem,
    ↪  instruction=prompt_custom.DETAILED_SOLUTION_PROMPT)
    # Generate multiple solutions using Custom
    solutions = [
        refined_solution['response'],
        detailed_solution['response']
    ]
    for _ in range(2):
        solution = await self.custom(input=problem,
        ↪  instruction=prompt_custom.GENERATE_SOLUTION_PROMPT)
        solutions.append(solution['response'])
    # Use ScEnsemble to select the best solution
    final_solution = await self.sc_ensemble(solutions=solutions, problem=problem)
    return final_solution['response'], self.llm.cost_manager.total_cost
```

# Results: Baselines (other methods)

- IO (direct LLM invocation, Input->Output)
- Chain-of-Thought
- Self Consistency CoT (5 answers)
- MultiPersona Debate
- Self-Refine
- MedPrompt.
- ADAS: automated workflow optimization method

# Results: Implementation Details

- employ Claude-3.5-sonnet (Anthropic, 2024) as the optimizer
- use models: DeepSeekV2.5 (Deepseek, 2024), GPT-4o-mini-0718 (OpenAI, 2024b), Claude-3.5-sonnet-0620 (Anthropic, 2024), GPT-4o-0513 (OpenAI, 2024a)) as executors.
- All models are accessed via APIs.
- We set the temperature to 1 for DeepSeek-V2.5 (for exploration) and to 0 for the other models.
- We divide the data into validation and test sets using a 1:4 ratio
- We set iteration rounds to 20 for AFLOW.

# Results: Benchmark Datasets

| Dataset | Domain | Task Type | Description & Purpose |
|---|---|---|---|
| **MATH** | Math | Complex symbolic reasoning | Competition-level math problems that require multi-step logic, verification, and structured output (e.g., symbolic math or algebraic proofs). |
| **GSM8K** | Math | Grade-school math | Natural language math problems requiring step-by-step numerical reasoning (e.g., "Bob has 3 apples..."). Ideal for testing chain-of-thought (CoT) strategies. |
| **HumanEval** | Code | Code generation | Code synthesis problems where the model must write Python functions that pass hidden unit tests. Evaluates logical structure and correctness. |
| **MBPP** | Code | Natural language to code | Simpler, beginner-level programming tasks with natural language instructions. Often used to evaluate grounding and basic coding logic. |
| **HotpotQA** | Open-domain QA | Multi-hop question answering | Requires reasoning over **two or more Wikipedia paragraphs** to answer questions. Good testbed for **multi-agent retrieval and synthesis workflows**. |
| **DROP** | Reading comprehension | Discrete reasoning over paragraphs | Complex questions over paragraphs that require arithmetic, counting, or logical reasoning — not just span extraction. Used to evaluate workflows that include **programmatic reasoning or symbolic tools**. |

# Results

| Method | Benchmarks | | | | | | Avg. |
|---|---|---|---|---|---|---|---|
| | HotpotQA | DROP | HumanEval | MBPP | GSM8K | MATH | |
| IO (GPT-4o-mini) | 68.1 | 68.3 | 87.0 | 71.8 | 92.7 | 48.6 | 72.8 |
| CoT (Wei et al., 2022) | 67.9 | 78.5 | 88.6 | 71.8 | 92.4 | 48.8 | 74.7 |
| CoT SC (5-shot) (Wang et al., 2022) | 68.9 | 78.8 | 91.6 | 73.6 | 92.7 | 50.4 | 76.0 |
| MedPrompt (Nori et al., 2023) | 68.3 | 78.0 | 91.6 | 73.6 | 90.0 | 50.0 | 75.3 |
| MultiPersona (Wang et al., 2024a) | 69.2 | 74.4 | 89.3 | 73.6 | 92.8 | 50.8 | 75.1 |
| Self Refine (Madaan et al., 2023) | 60.8 | 70.2 | 87.8 | 69.8 | 89.6 | 46.1 | 70.7 |
| ADAS (Hu et al., 2024) | 64.5 | 76.6 | 82.4 | 53.4 | 90.8 | 35.4 | 67.2 |
| Ours | **73.5** | **80.6** | **94.7** | **83.4** | **93.5** | **56.2** | **80.3** |

All methods are executed with GPT-4o-mini on divided test set

# Results

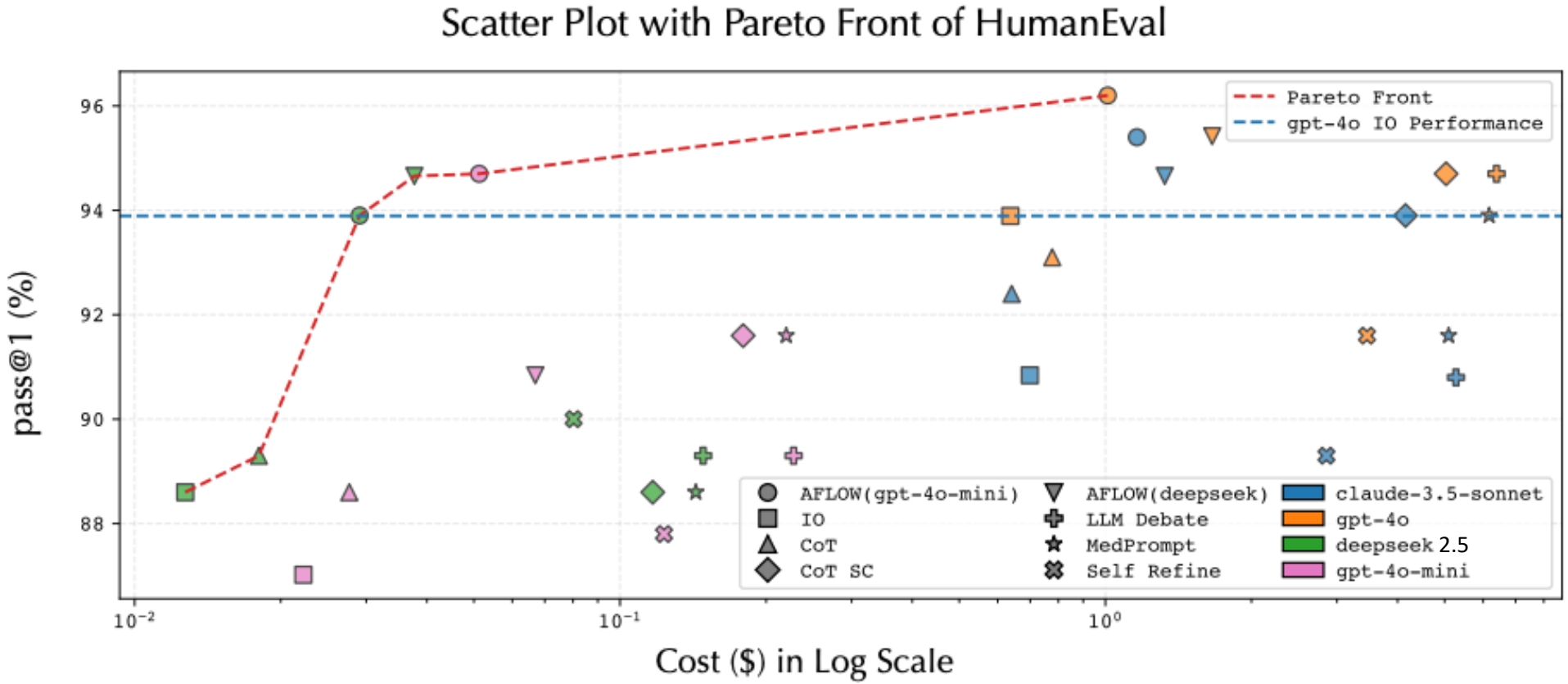
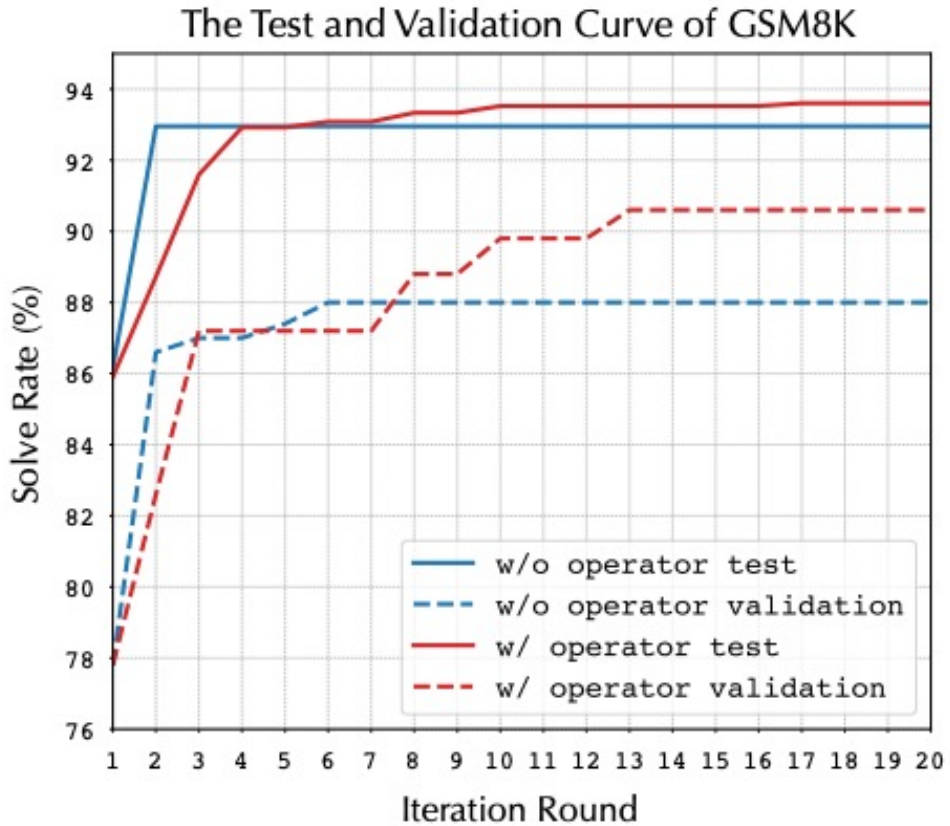



Figure 4: The cost refers to the total expense of executing the divided HumanEval test set. AFLOW (execution model) refers to workflows found by AFLOW using the execution model to obtain feedback. The colors in the legend represent the LLM used to execute each workflow in test dataset. The




(A)

The Code Represented Workflow of GSM8K

```python
async def __call__(self, problem: str):
"""
Implementation of the graph
"""
    solutions = []
    for _ in range(5): # Generate 5 solutions
        solution = await self.custom(
            input=problem, instruction=MATH_SOLVE_PROMPT
        )
        solutions.append(solution['response'])

    final_solution = await self.sc_ensemble(
        solutions=solutions, problem=problem
    )
    # Add a verification step using Programmer operator
    verification = await self.programmer(
        problem=problem,
        analysis=final_solution['response']
    )

    if verification['output']:
        return verification['output'], self.total_cost
    else:
        return final_solution['response'], self.total_cost
```

(B)

Figure 5: (A) Comparison of highest performance curves on GSM8K for both validation and test sets generated by AFLOW with and without operators. Compared to other datasets, GSM8K has a larger data volume, meaning that the same percentage improvement represents a greater increase in correctly solved samples, avoiding fluctuations in improvement due to small data size that could affect comparisons; (B): The code for the best-performing workflow discovered by AFLOW on the GSM8K dataset.

# Conclusion

- AFLOW has leveraged Monte Carlo Tree Search and code-represented workflows to navigate the vast search space of possible workflows efficiently (prompts, operators, code represented edges).
- AFLOW outperformed manually designed methods and existing automated optimization approaches.
- Importantly, AFLOW has enabled weaker models to outperform stronger ones on the Pareto front of cost-effectiveness.
- These results have highlighted AFLOW’s potential for enhancing LLMs’ problem-solving capabilities while optimizing computational costs.

# Upcoming meetups – Open to proposals and guest speakers!

- Multi-agent real-world use case
- Agent orchestrations
- Open-source frameworks
- Hands-on
- Panel discussions

Slides posted at:
https://github.com/YanXuHappygela/LLM-reading-group

Recordings posted at:

**YanAITalk**

@yanaitalk · 2.87K subscribers · 65 videos

Make machine learning easy to understand! ...more