Heterogenous Swarms

Jointly Optimizing Model Roles and Weights in Multi-LLM Systems

Shangbin Feng[1*] Zifeng Wang[2] Palash Goyal[2] Yike Wang[1] Weijia Shi[1] Huang Xia[3]
Hamid Palangi[2] Luke Zettlemoyer[1] Yulia Tsvetkov[1] Chen- Yu Lee[2] Tomas Pfister[2]

[1]University of Washington [2]Google Cloud AI Research [3]Google

NeurIPS 2025

Paper presented by Abhinav Biju and Eric Fackelman
4/21/27

Background

What are multi-LLM systems?

- To handle complex, multi-step tasks, we can employ a group of specialized LLMs that collaborate together

- This approach offers many key **advantages**:

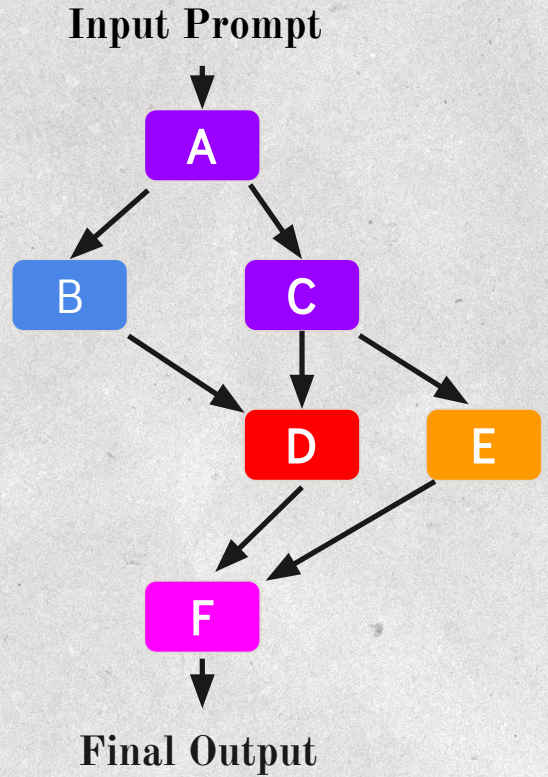
Collaborative Verification

Specialized Expertise

Parallel Processing

Cost Optimization

Multi-LLM Systems as DAGs

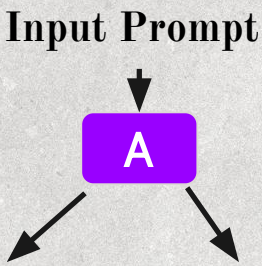


In this paper, multi-LLM systems are represented as **Directed Acyclic Graphs (DAGs).**

Each node is a language model.

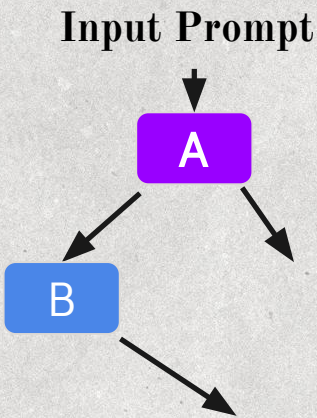
Edges represent the flow of information.

Let’s walk through an example



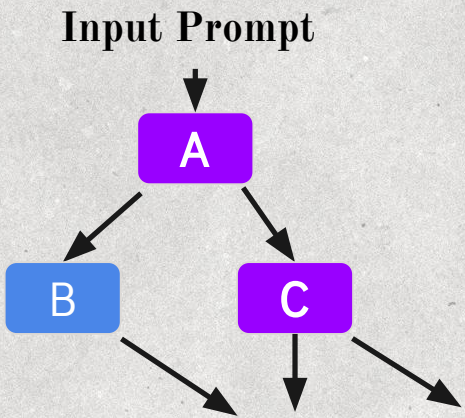
Node A: Task Interpreter

- Identify key concepts
- Break into components
- Define goals



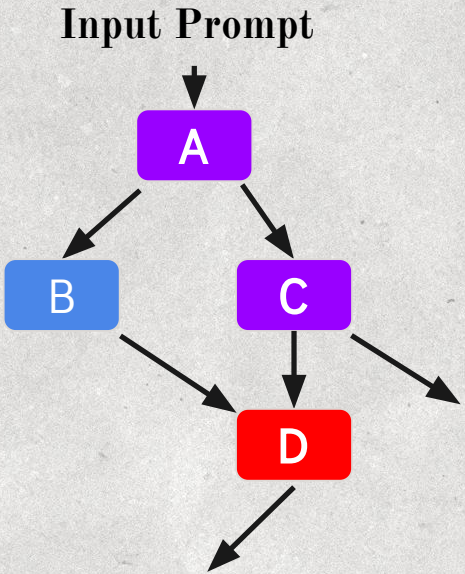
Node B: Factual Specialist

- Definitions
- Known principles
- Relevant formulas / facts



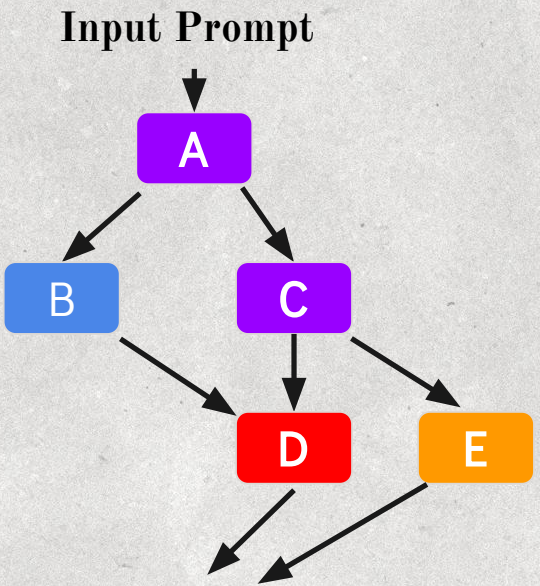
Node C: Reasoning + Exploration

- Step-by-step reasoning
- Hypotheses
- Edge cases



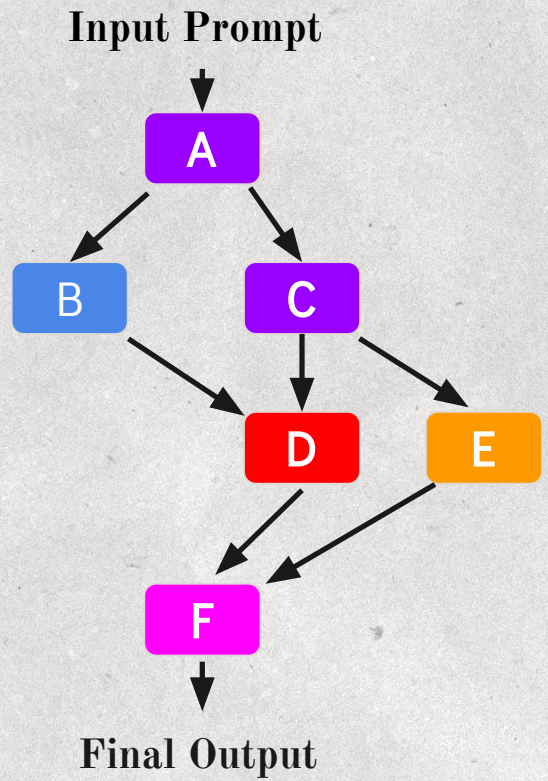
Node D: Synthesizer

- Takes in facts and reasoning
- Outputs:
  - Coherent explanation grounded in facts
  - Resolves inconsistencies



Node E: Alternate Perspective

- Different framing
- Simplified explanation or counterpoint
- Uncontaminated by B



Node F: Final Aggregator

- Final polished answer
- Chooses best framing

Important Caveat: This Diagram is an Abstraction

The roles shown are designed and interpetable.

The actual system learns emergent, implicit roles.

Motivation & Problem Statement

Previous approaches to build multi-LLM systems are often:

Fixed-role

- LLMs are orchestrated through a fixed workflow
- Role is often defined by system prompts

Problem: Building new workflows requires manual prompt engineering, creating a scaling bottleneck.

Fixed-weight

- Employ static LLMs which contextualize roles through textual interaction

Problem: Static models are repeated across roles and contexts, creating a bottleneck to flexibility

Previous approaches to build multi-LLM systems are often:

Fixed-role

- LLMs are orchestrated through a fixed workflow
- Role is often defined by system prompts

Problem: Building new workflows requires manual prompt engineering, creating a scaling bottleneck.

Fixed-weight

- Employ static LLMs which contextualize roles through textual interaction

Problem: Static models are repeated across roles and contexts, creating a bottleneck to flexibility

Previous approaches to build multi-LLM systems are often:

Fixed-role

- LLMs are orchestrated through a fixed workflow
- Role is often defined by system prompts

Problem: Building new workflows requires manual prompt engineering, creating a scaling bottleneck.

Fixed-weight

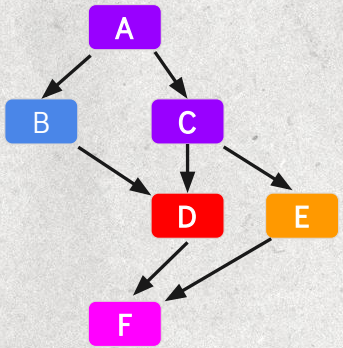
- Employ static LLMs which contextualize roles through textual interaction

Problem: Static models are repeated across roles and contexts, creating a bottleneck to flexibility

Problem Statement

- This paper builds a novel optimization framework for multi-LLM systems that **dynamically** optimizes both **roles** and **weights**
- It builds off of previous literature demonstrating the success of dynamic-weight systems

1. Optimize structure of the graph
2. Optimize models at each node



Core Contribution

Overall Architecture

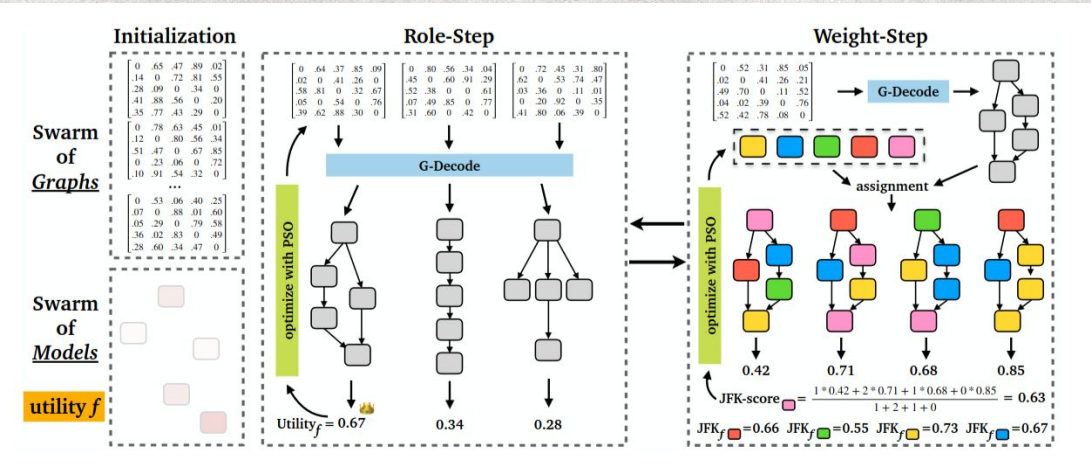


Figure 2. Source: Feng et al. (2025)

- Jointly optimize roles
- Jointly optimize weights
- Alternate between role-step and weight-step

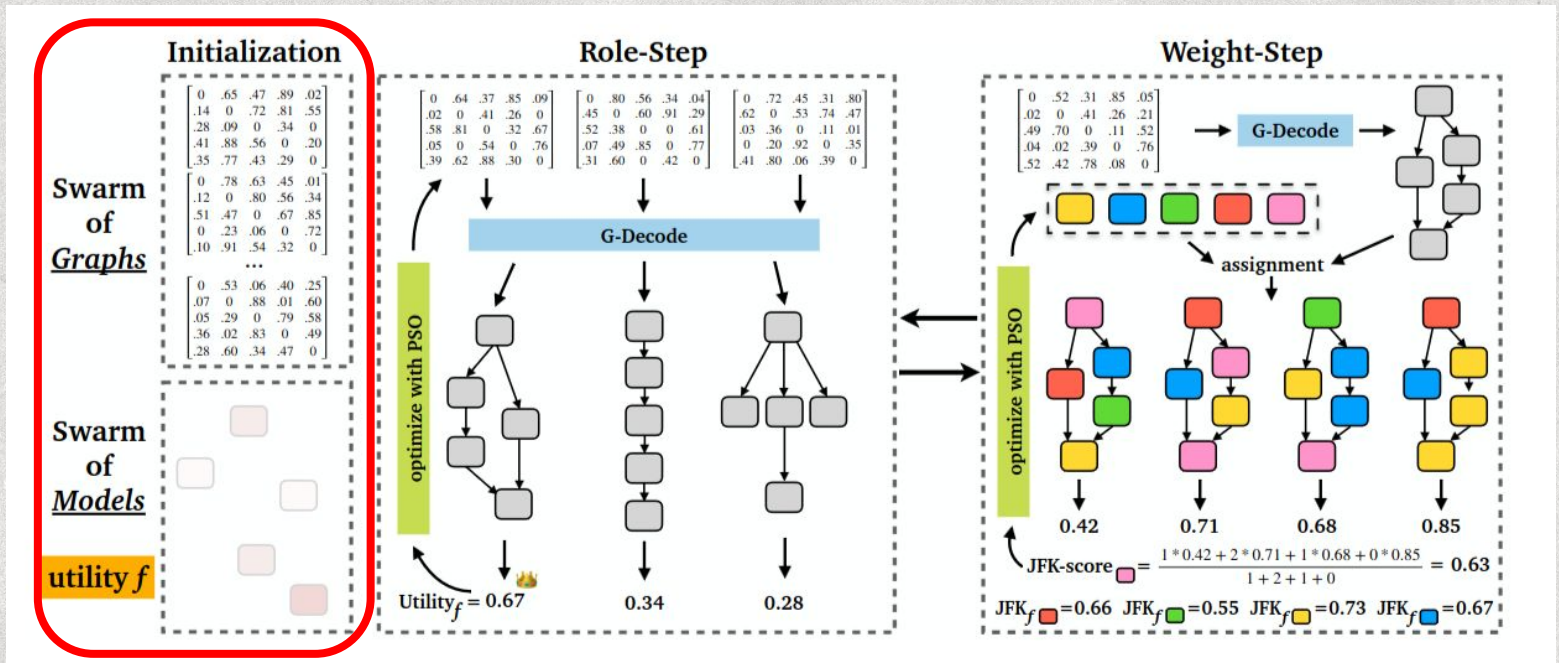


Figure 2. Source: Feng et al. (2025)

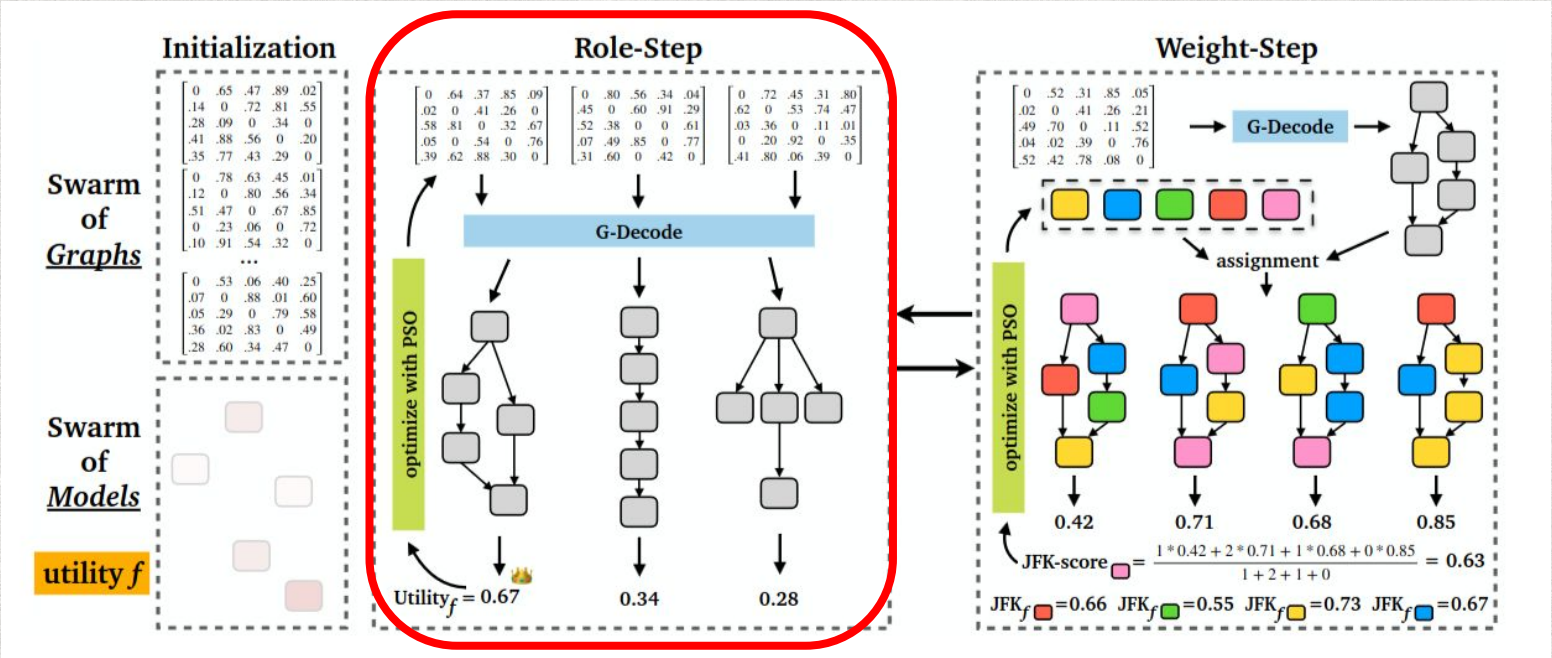


Figure 2. Source: Feng et al. (2025)

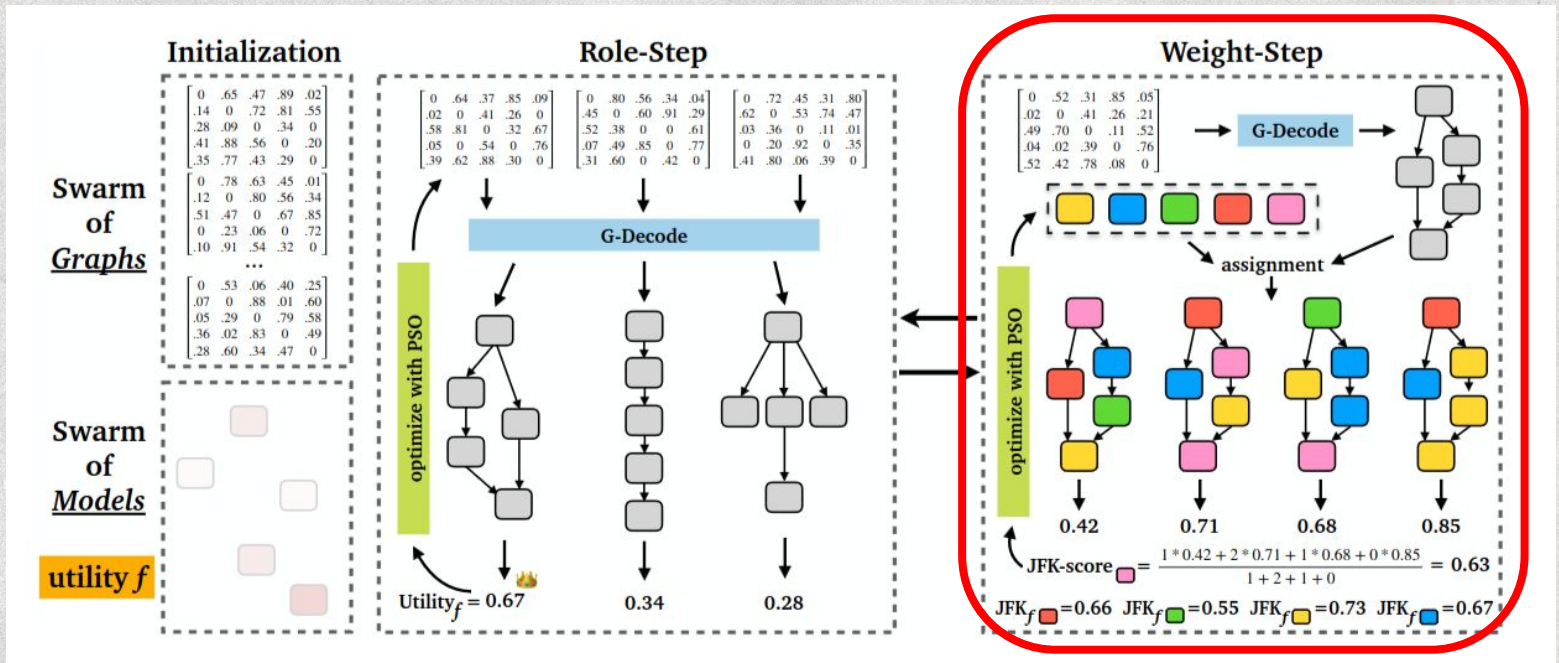


Figure 2. Source: Feng et al. (2025)

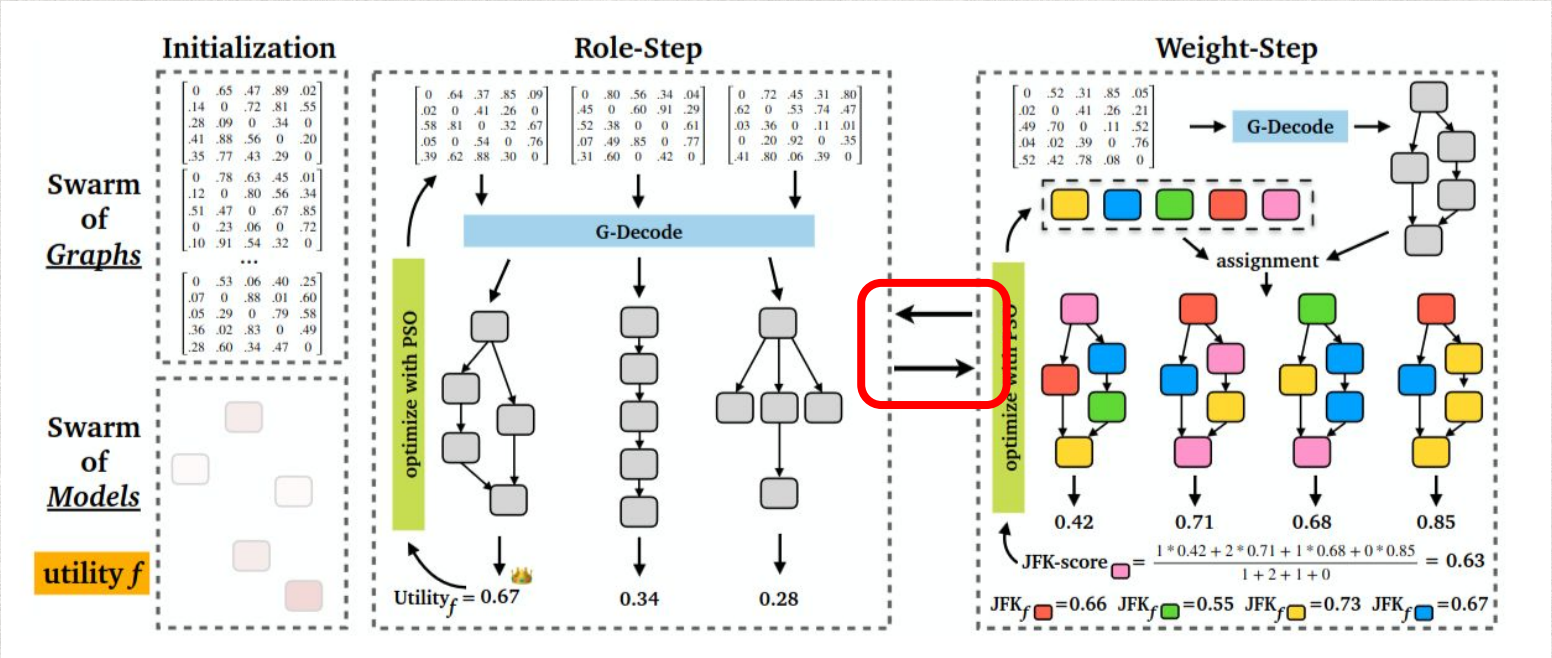


Figure 2. Source: Feng et al. (2025)

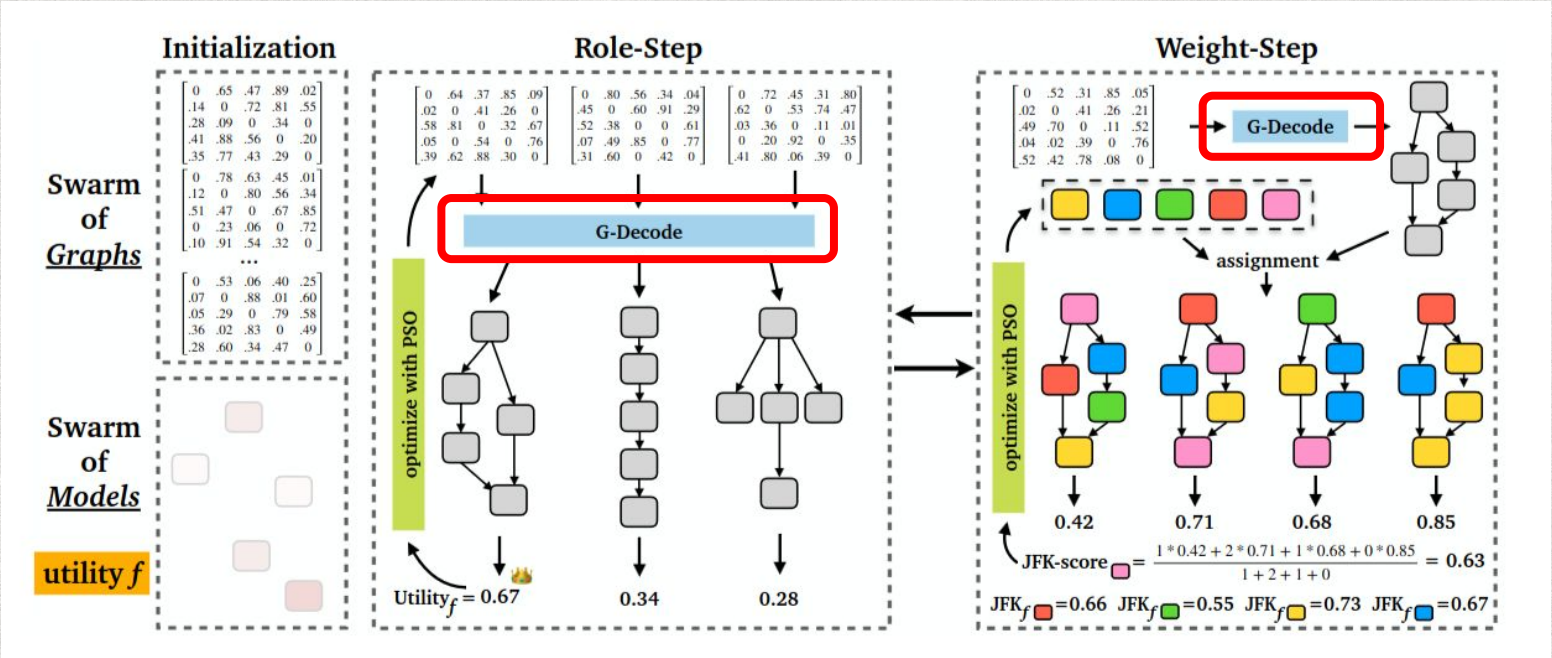
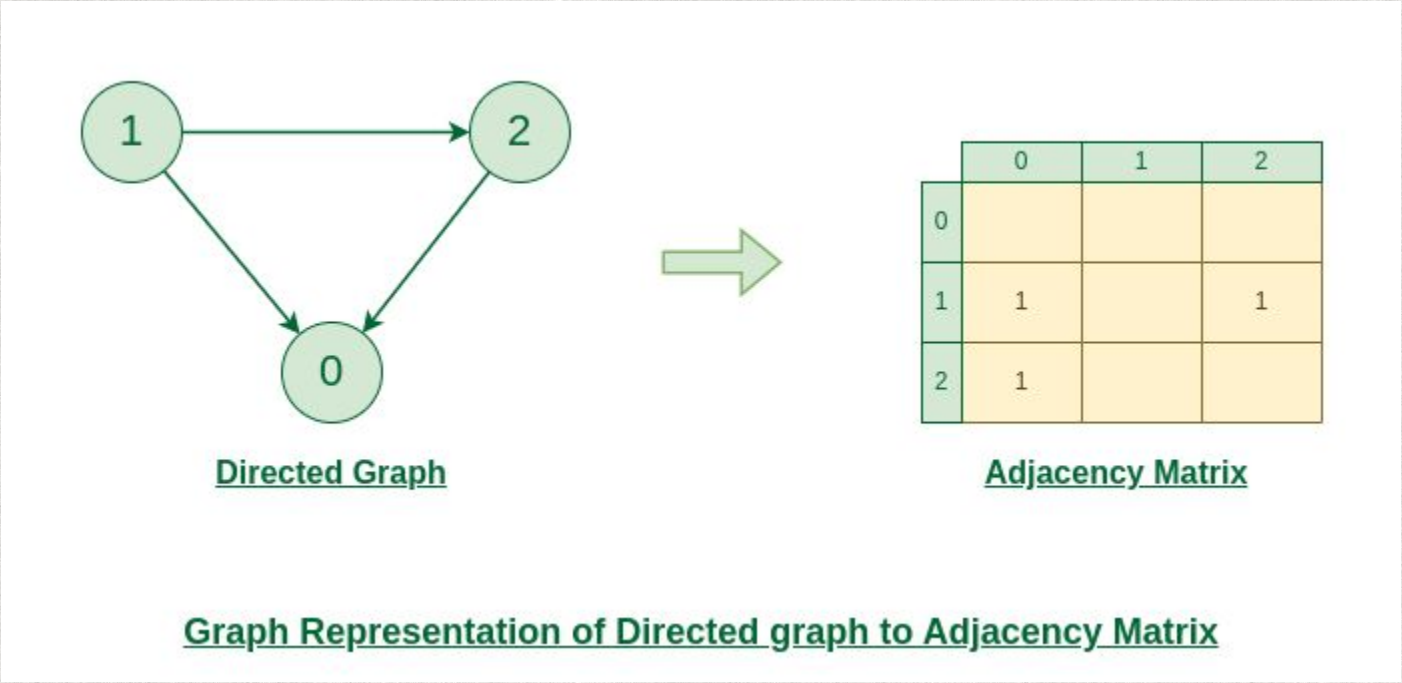


Figure 2. Source: Feng et al. (2025)

Representing graphs with matrices



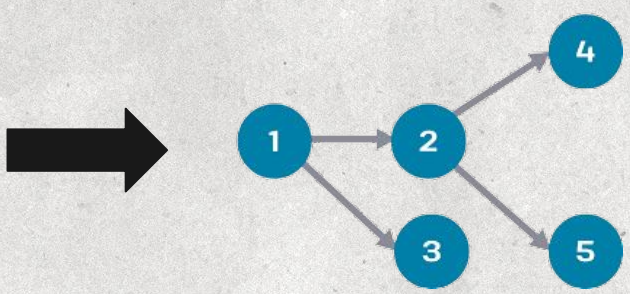
Source: GeeksforGeeks, 2025

G-Decode algorithm

The paper introduces G-Decode, which translates continuous adjacency matrices into discrete DAGs.

Goal:

.47	.12	.33	.11
.72	.12	.11	.03
.06	.41	.23	.18
.51	.32	.06	.05



G-Decode algorithm

The paper introduces G-Decode, which translates continuous adjacency matrices into discrete DAGs.

Start with adjacency matrix A:

.47	.12	.33	.11
.72	.12	.11	.03
.06	.41	.23	.18
.51	.32	.06	.05

G-Decode algorithm

Start with adjacency matrix A:

.47	.12	.33	.11
.72	.12	.11	.03
.06	.41	.23	.18
.51	.32	.06	.05

Step 1:

Find the terminal node k:

$$k = \text{top-p}\left(\left\{\frac{1}{\sum_{j=1}^{n} a_{ij}}\right\}_{i=1}^{n}\right)$$

G-Decode algorithm

Find the out-degrees for each node.

Start with adjacency matrix A:

.47	.12	.33	.11
.72	.12	.11	.03
.06	.41	.23	.18
.51	.32	.06	.05

= 1.03

Find the terminal node k:

$$k = \text{top-p}\left(\left\{\frac{1}{\sum_{j=1}^{n} a_{ij}}\right\}_{i=1}^{n}\right)$$

Out-degrees tell us how outwardly connected a node is.

G-Decode algorithm

Find the out-degrees for each node.

Start with adjacency matrix A:

.47	.12	.33	.11	= 1.03
.72	.12	.11	.03	**= 0.98**
.06	.41	.23	.18	
.51	.32	.06	.05	

Find the terminal node k:

$$k = \text{top-p}\left(\left\{\frac{1}{\sum_{j=1}^{n} a_{ij}}\right\}_{i=1}^{n}\right)$$

G-Decode algorithm

Find the out-degrees for each node.

Start with adjacency matrix A:

.47	.12	.33	.11	= 1.03
.72	.12	.11	.03	= 0.98
.06	.41	.23	.18	**= 0.88**
.51	.32	.06	.05	

Find the terminal node k:

$$k = \text{top-p}\left(\left\{\frac{1}{\sum_{j=1}^{n} a_{ij}}\right\}_{i=1}^{n}\right)$$

G-Decode algorithm

Find the out-degrees for each node.

Start with adjacency matrix A:

.47	.12	.33	.11	= 1.03
.72	.12	.11	.03	= 0.98
.06	.41	.23	.18	= 0.88
.51	.32	.06	.05	**= 0.94**

Find the terminal node k:

$$k = \text{top-p}\left(\left\{\frac{1}{\sum_{j=1}^{n} a_{ij}}\right\}_{i=1}^{n}\right)$$

G-Decode algorithm

Start with adjacency matrix A:

.47	.12	.33	.11	= 1.03
.72	.12	.11	.03	= 0.98
.06	.41	.23	.18	= 0.88
.51	.32	.06	.05	= 0.94

Take inverse:

= 0.97

= 1.02

= 1.13

= 1.06

Find the terminal node k:

$$k = \text{top-p}\left(\left\{\frac{1}{\sum_{j=1}^{n} a_{ij}}\right\}_{i=1}^{n}\right)$$

G-Decode algorithm

Find the terminal node k:

$$k = \text{top-p}\left(\left\{\frac{1}{\sum_{j=1}^{n} a_{ij}}\right\}_{i=1}^{n}\right)$$

Start with adjacency matrix A:

.47	.12	.33	.11	= 1.03
.72	.12	.11	.03	= 0.98
.06	.41	.23	.18	= 0.88
.51	.32	.06	.05	= 0.94

Take inverse:

= 0.97

= 1.02

= 1.13

= 1.06

Choose a row using top-p sampling

Here, we sample x_3

G-Decode algorithm

.47	.12	.33	.11
.72	.12	.11	.03
.06	.41	.23	.18
.51	.32	.06	.05

Terminal node $k = x_3$

DAG diagram

G-Decode algorithm

Next, iteratively select a remaining node u based on out-degrees.

.47	.12	.33	.11
.72	.12	.11	.03
.06	.41	.23	.18
.51	.32	.06	.05

Existing nodes v: {3, }

DAG diagram

G-Decode algorithm

Next, iteratively select a remaining node u based on out-degrees.

.47	.12	.33	.11
.72	.12	.11	.03
.06	.41	.23	.18
.51	.32	.06	.05

Select x_1

= 1.03

= 0.98

= 0.94

Existing nodes k: {3, }

DAG diagram

G-Decode algorithm

Next, iteratively select a remaining node u based on out-degrees.

.47	.12	.33	.11	= 1.03
.72	.12	.11	.03	= 0.98
.06	.41	.23	.18	
.51	.32	.06	.05	= 0.94

Then add an edge between u and any existing node v with probability:

$$\frac{\exp(a_{uv})}{\sum_{i \in \varepsilon} \exp(a_{ui})}$$

Existing nodes k: {3, }

DAG diagram

G-Decode algorithm

Calculate softmax for each existing node

x_1 \ x_3				
.47	.12	.33	.11	= 1.03
.72	.12	.11	.03	= 0.98
.06	.41	.23	.18	
.51	.32	.06	.05	= 0.94

Then add an edge between u and any existing node v with probability:

$$\frac{\exp(a_{uv})}{\sum_{i \in \varepsilon} \exp(a_{ui})}$$

$P(x_1 \mid x_3) = 1.0$

Add an edge

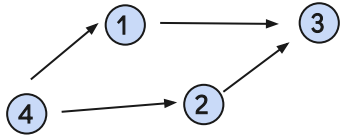
Existing nodes k: {3, }

DAG diagram

G-Decode algorithm

After repeating this process:

DAG diagram



Now we can evaluate our system according to utility function f

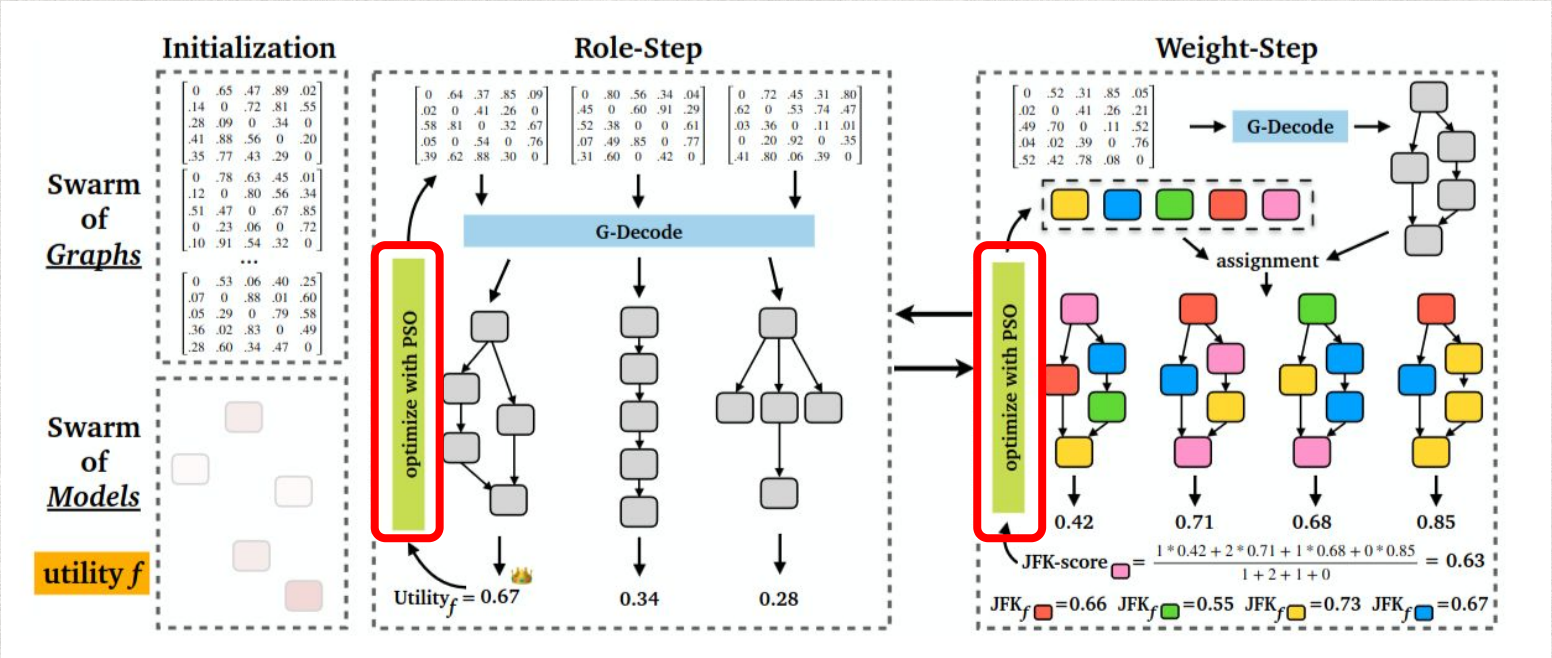


Figure 2. Source: Feng et al. (2025)

What is “Particle Swarm Optimization”?

- PSO is an optimization algorithm inspired by social behavior, introduced in 1995
- It finds optimal solutions by having a population of candidate solutions (particles) move through a multi-dimensional search space

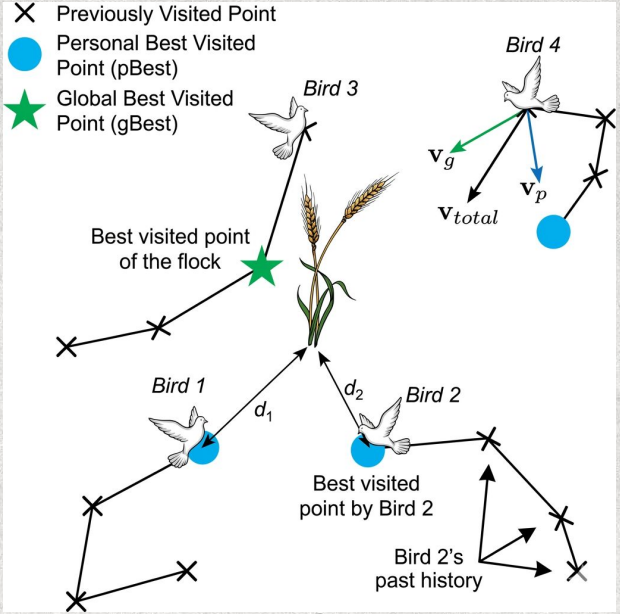


Figure adapted from Kovitz 2026.

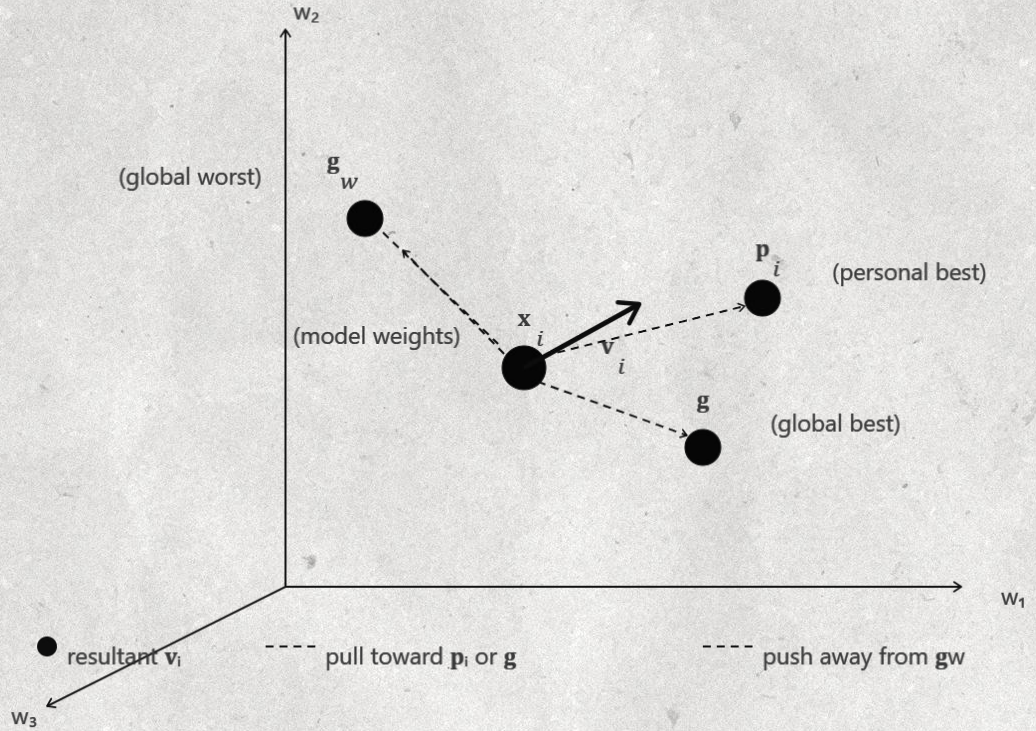
Particle Swarm Optimization (PSO)

We will use it to optimize two things:

1. The graph structure that connects models (role-step)
2. The models themselves (via internal weights) (weight-step)

We will use $\mathbf{x}_i$ to represent either the adj. matrices or model weights.

Particle Swarm Optimization (PSO)



Particle Swarm Optimization (PSO)

Update x_i:

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \lambda \mathbf{v}_i$$

x_i : continuous vector
λ : step length hyperparameter
v_i: velocity vector

Particle Swarm Optimization (PSO)

First, calculate v_i

$$\mathbf{v}_i \leftarrow \frac{1}{\mathcal{C}}\left[r_v\phi_v\mathbf{v}_i + r_p\phi_p(\mathbf{p}_i - \mathbf{x}_i) + r_g\phi_g(\mathbf{g} - \mathbf{x}_i) - r_w\phi_w(\mathbf{g}_w - \mathbf{x}_i)\right]$$

Particle Swarm Optimization (PSO)

Hyperparameters

$$\phi_v, \phi_p, \phi_g, \phi_w$$

Randomness factors

$$r_v, r_p, r_g, r_w \sim \mathcal{U}(0, 1)$$

$$\mathbf{v}_i \leftarrow \frac{1}{\mathcal{C}} \left[r_v \phi_v \mathbf{v}_i + r_p \phi_p (\mathbf{p}_i - \mathbf{x}_i) + r_g \phi_g (\mathbf{g} - \mathbf{x}_i) - r_w \phi_w (\mathbf{g}_w - \mathbf{x}_i) \right]$$

Inertia | Personal best | Global best | Global worst

Normalization term: $\mathcal{C} = r_v \phi_v + r_p \phi_p + r_g \phi_g + r_w \phi_w$

Particle Swarm Optimization (PSO)

Hyperparameters

$$\phi_v, \phi_p, \phi_g, \phi_w$$

Randomness factors

$$r_v, r_p, r_g, r_w \sim \mathcal{U}(0, 1)$$

$$\mathbf{v}_i \leftarrow \frac{1}{\mathcal{C}}\left[r_v\phi_v\mathbf{v}_i + r_p\phi_p(\mathbf{p}_i - \mathbf{x}_i) + r_g\phi_g(\mathbf{g} - \mathbf{x}_i) - r_w\phi_w(\mathbf{g}_w - \mathbf{x}_i)\right]$$

Inertia | Personal best | Global best | Global worst

The x_i keeps part of its velocity, preventing it from moving erratically.

Particle Swarm Optimization (PSO)

Hyperparameters

$$\phi_v, \phi_p, \phi_g, \phi_w$$

Randomness factors

$$r_v, r_p, r_g, r_w \sim \mathcal{U}(0, 1)$$

$$\mathbf{v}_i \leftarrow \frac{1}{\mathcal{C}} \left[r_v \phi_v \mathbf{v}_i + r_p \phi_p (\mathbf{p}_i - \mathbf{x}_i) + r_g \phi_g (\mathbf{g} - \mathbf{x}_i) - r_w \phi_w (\mathbf{g}_w - \mathbf{x}_i) \right]$$

Inertia | **Personal best** | Global best | Global worst

Draw the x_i toward its personal best, promoting local exploration.

Particle Swarm Optimization (PSO)

Hyperparameters

$$\phi_v, \phi_p, \phi_g, \phi_w$$

Randomness factors

$$r_v, r_p, r_g, r_w \sim \mathcal{U}(0, 1)$$

$$\mathbf{v}_i \leftarrow \frac{1}{\mathcal{C}} \left[r_v \phi_v \mathbf{v}_i + r_p \phi_p (\mathbf{p}_i - \mathbf{x}_i) + r_g \phi_g (\mathbf{g} - \mathbf{x}_i) - r_w \phi_w (\mathbf{g}_w - \mathbf{x}_i) \right]$$

Inertia | Personal best | **Global best** | **Global worst**

Draw the x_i toward the global best, and away from global worst, enabling collective intelligence among all x_i's

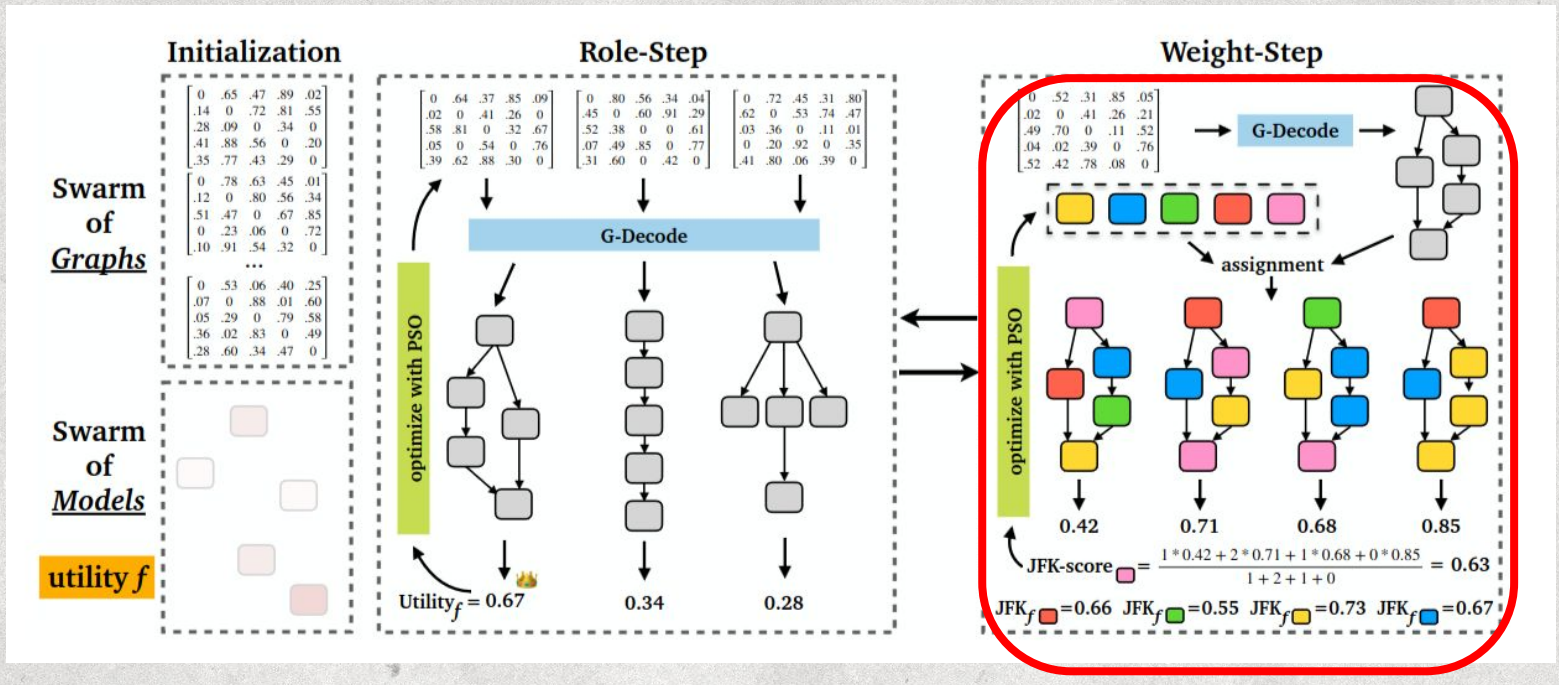
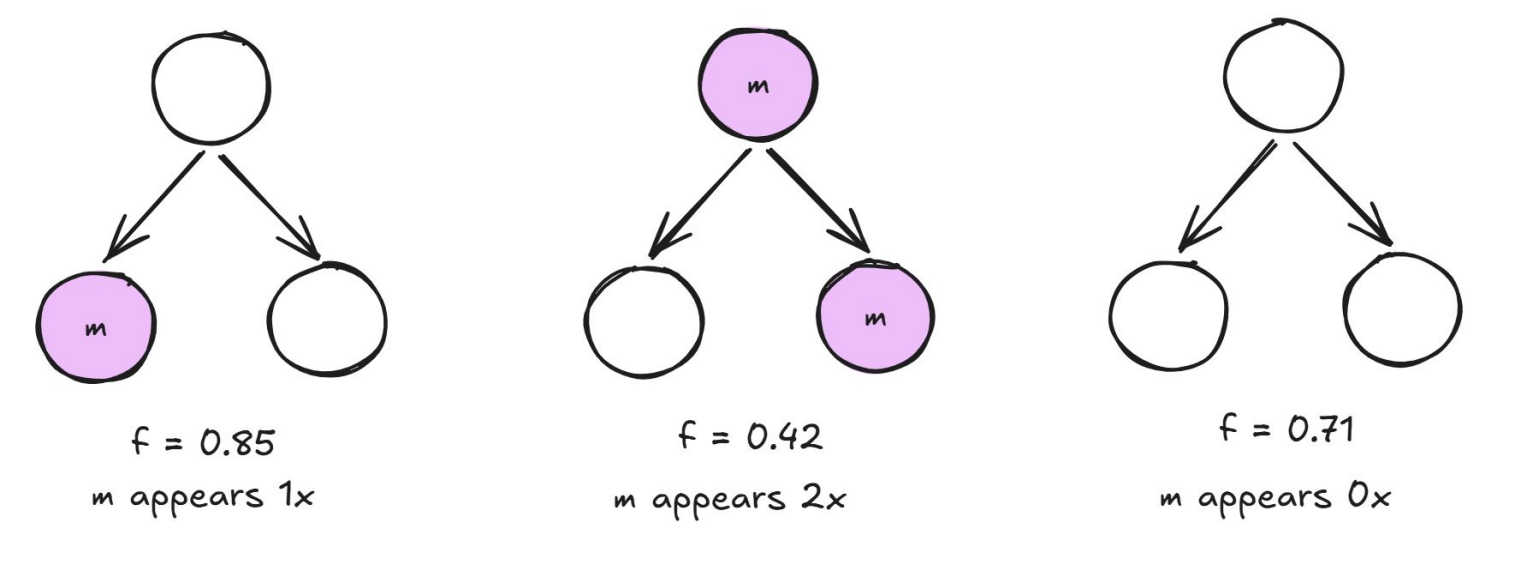


Figure 2. Source: Feng et al. (2025)

Why JFK-Score?

- Update quality metric from utility function *f* to focus on improving *each* model's weights
- Account for frequency of model

Why JFK-Score?



Naive Average = (0.85 + 0.42 + 0.71) / 3 = 0.66

JFK - score(frequency weighted) = (1*0.85 + 2*0.42 + 0*0.71) / (1 + 2 + 0) = 0.56

→ A model shouldn't get credit for graphs it wasn't assigned to

JFK-Score

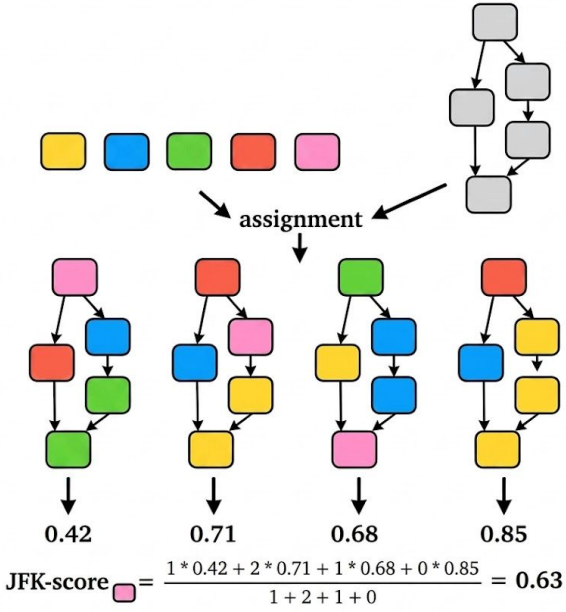


Figure 2. Source: Feng et al. (2025)

Models can:

- appear multiple times
- Play different roles
- Different models in each graph

- Randomly assign the n models into M copies of A_{best} so each copy has a different assignment of models to roles

JFK-Score

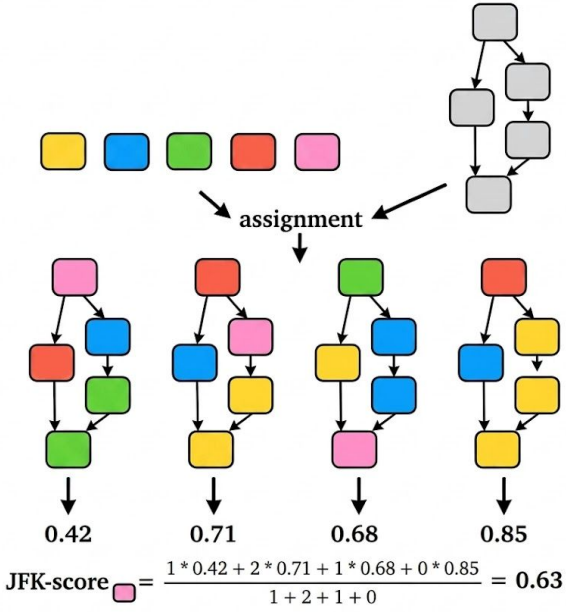
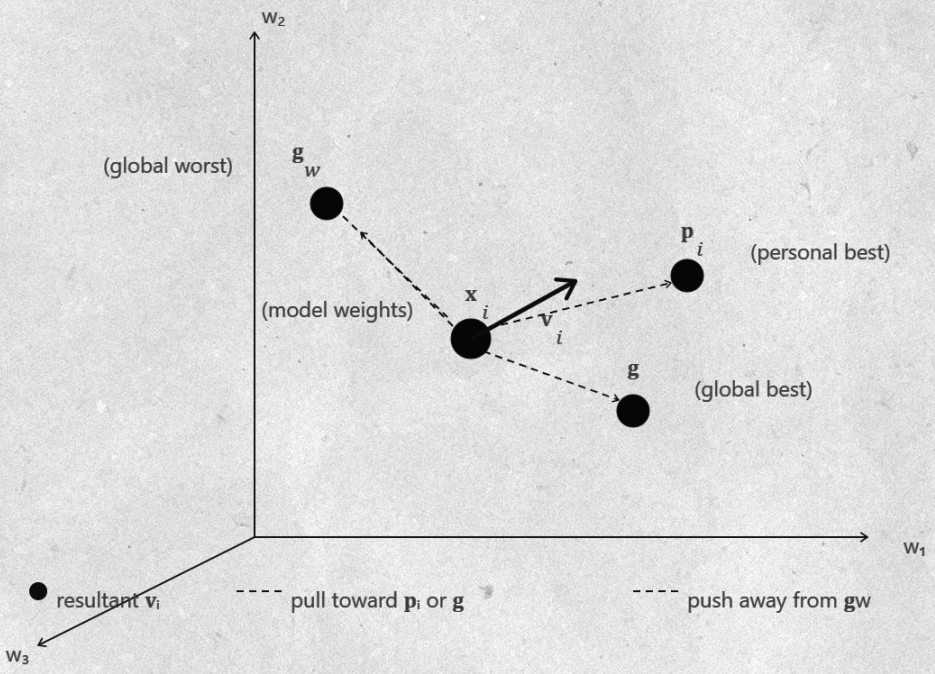


Figure 2. Source: Feng et al. (2025)

$$\text{JFK-score}(\mathbf{x}_i) = \frac{\sum_{j=1}^{M} \text{cnt}_{i,j} \times f(\mathcal{X}^i)}{\sum_{j=1}^{M} \text{cnt}_{i,j}}$$

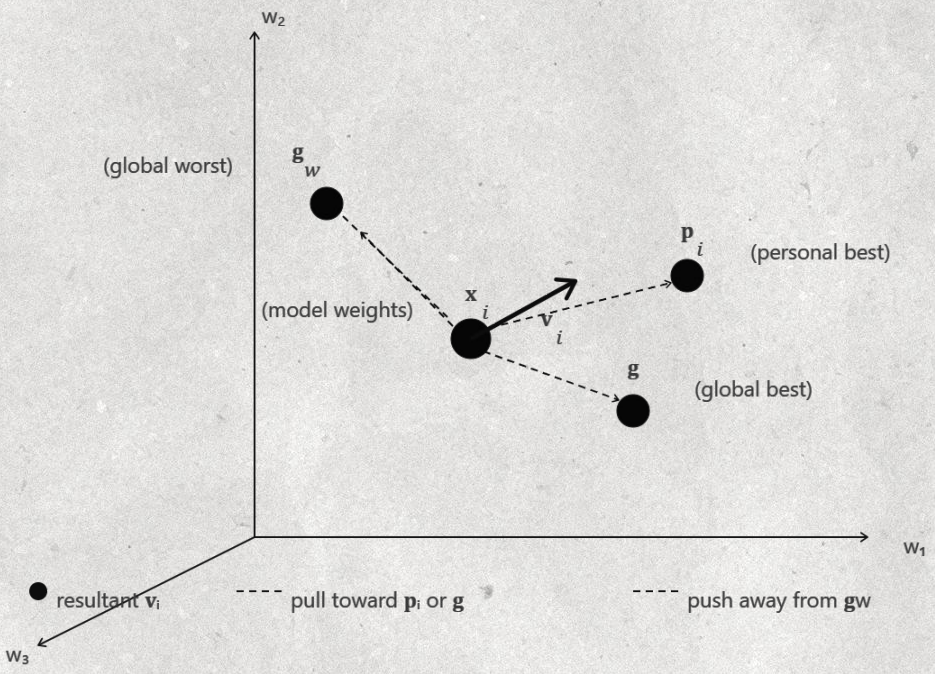
$\mathbf{x}_i$	: Model i (out of n total models in the pool)
$\mathcal{X}^i$	: The i-th random assignment of models to DAG positions
$\text{cnt}_{i,j}$	: Number of times model $\mathbf{x}_i$ appears in assignment $\mathcal{X}^i$
$f(\mathcal{X}^i)$	: Utility score of the full system under assignment $\mathcal{X}^i$
M	: Total number of random assignment repetitions

Particle Swarm Optimization (PSO)



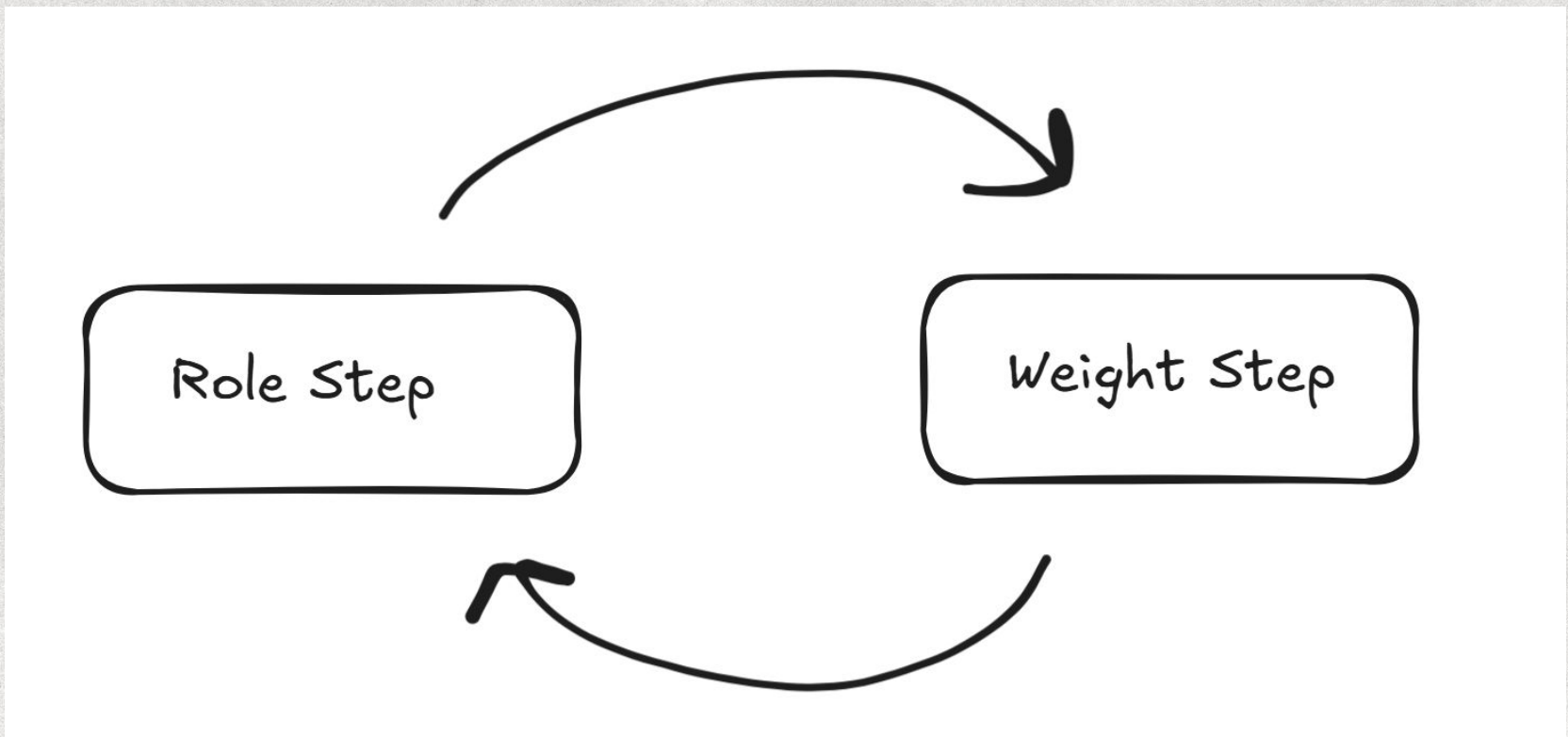
- Each model x_i is a particle, it's position a weight vector
- Instead of utility F used for fitness, it's JFK score(x_i)

Particle Swarm Optimization (PSO)



- PSO updates each model's weights to move towards higher JFK model weight configs
- After weight step, the improved models **feed back into the next role step**
- This continues until convergence

Bringing it together...



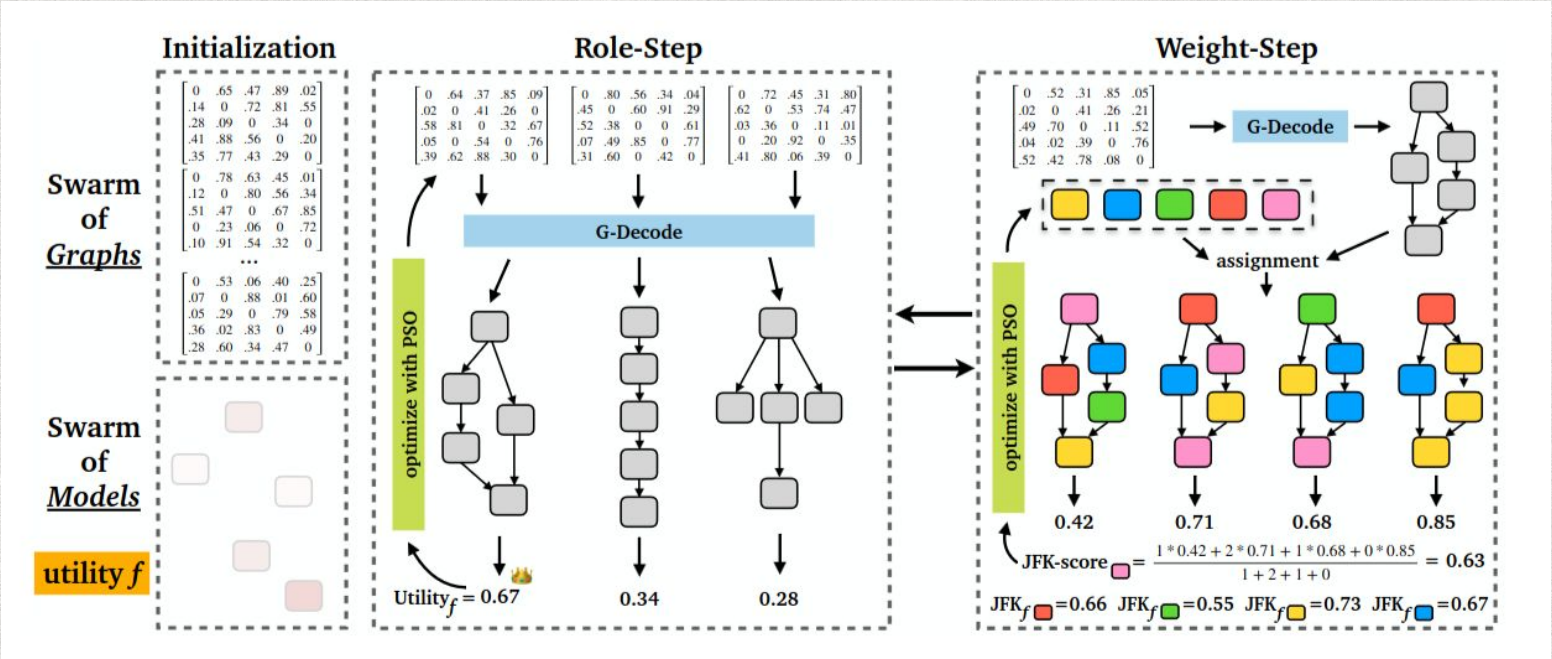


Figure 2. Source: Feng et al. (2025)

Experiments

Datasets Used

- **Knowledge**
  - MMLU-pro
  - K-Cross
  - COM2
- **Reasoning**
  - GSM8k
  - NLGraph
  - Normad
-
- **Agent**
  - GAIA-text
  - AB-kg
    - Knowledge graph
  - AB-ltp
    - Lateral thinking puzzle subtasks
- **Misc**
  - Qasper
    - Long context
  - AbstainQA
    - reliability
  - WoW
    - LLM-as-a-judge

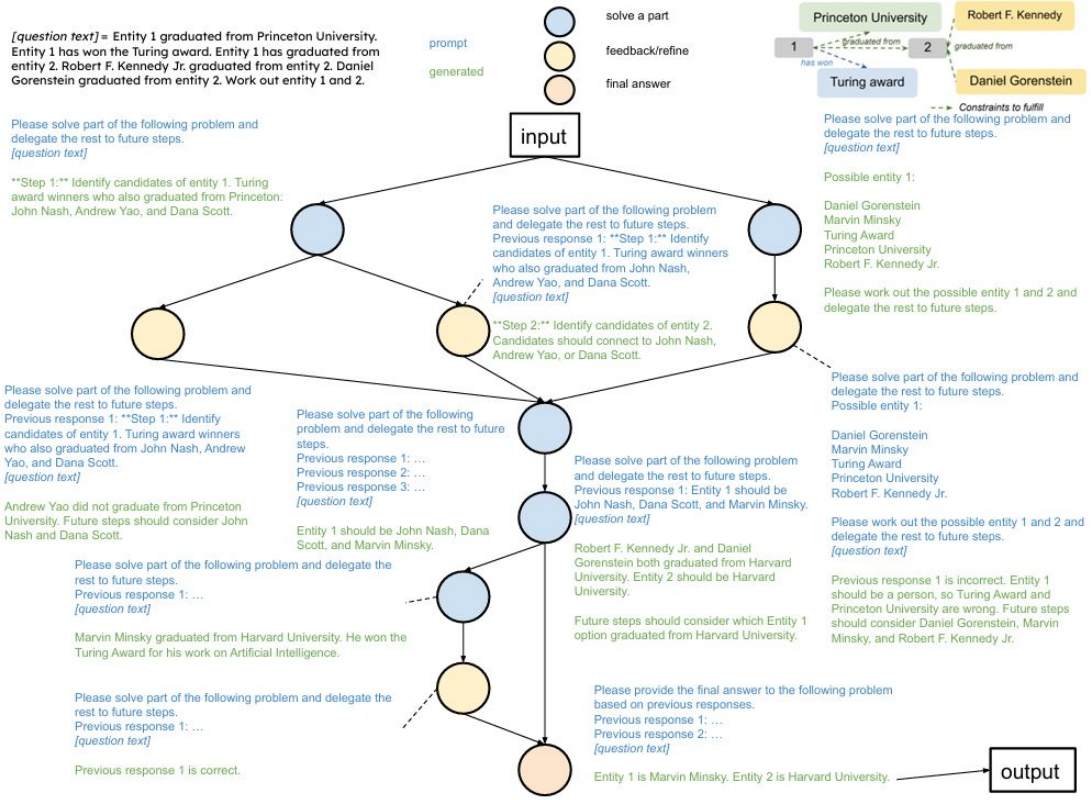


Figure 13: Working example one of HETEROGENEOUS SWARMS.

Experiment 1: Main Comparison

Design: 10 Gemma-7B models, each fine tuned on different domain from Tulu-v2 dataset

Eval: Task accuracy for each dataset

Key Insight:

- H-Swarm best on 11/12 datasets, +18.5% over second-best on average.
- Joint optimization of both roles and weights outperforms any approach that optimizes only one

	Knowledge			Reasoning			Agent			Miscellaneous		
	MMLU-pro	K-Cross	COM2	GSM8k	NLGraph	Normad	GAIA-text	AB-kg	AB-ltp	Qasper	AbstainQA	WoW
BEST SINGLE	0.231	0.346	0.488	0.237	0.535	0.545	0.107	0.383	0.120	0.174	0.065	0.415
PRED. MERGE	0.173	0.309	0.391	0.074	0.502	0.481	0.036	0.225	/	/	/	0.471
DATA MERGE	0.176	0.370	0.377	0.143	0.423	0.415	0.071	0.242	0.112	0.147	-0.025	0.461
UNIFORM SOUP	0.206	0.295	0.519	0.352	0.500	0.430	0.036	0.392	0.105	0.166	0.003	0.455
DARE-TIES	0.230	0.372	0.476	0.307	0.544	0.427	0.071	0.300	0.108	0.137	0.140	0.515
GREEDY SOUP	0.219	0.355	0.539	0.330	0.530	0.543	0.071	0.333	0.114	0.184	0.014	0.565
PACK OF LLMS	0.235	0.352	0.512	0.327	0.532	0.543	0.143	0.392	0.106	0.157	0.095	0.545
LORAHUB	0.231	0.291	0.502	0.354	0.568	0.548	0.071	0.375	0.106	0.169	0.064	0.530
MODEL SWARMS	0.254	0.428	0.505	0.459	**0.672**	0.554	0.107	0.358	0.135	0.225	0.175	0.540
CHAIN	0.216	0.310	0.495	0.295	0.462	0.489	0.143	0.325	0.148	0.218	0.014	0.493
STAR	0.250	0.342	0.508	0.333	0.545	0.518	0.036	0.283	0.130	0.216	0.125	0.499
GPT-SWARM	0.216	0.320	0.460	0.334	0.611	0.510	0.143	0.333	0.134	0.216	0.023	0.492
META-AGENT	0.212	0.276	0.477	0.433	0.515	0.369	0.071	0.325	0.112	0.167	0.016	0.472
AGENT-PRUNE	0.214	0.321	0.497	0.180	0.460	0.467	0.107	0.333	0.122	0.211	-0.005	0.470
GNNS	0.201	0.339	0.479	0.364	0.593	0.530	0.071	0.308	0.148	0.203	0.076	0.503
AGENTVERSE	0.239	0.309	0.501	0.403	0.633	0.513	0.107	0.367	0.136	0.195	0.103	0.489
MACNET	0.252	0.323	0.517	0.409	0.617	0.537	0.143	0.383	0.138	0.207	0.127	0.512
H-SWARMS	**0.312**	**0.450**	**0.579**	**0.481**	0.660	**0.588**	**0.250**	**0.425**	**0.215**	**0.266**	**0.220**	**0.590**

Table 1: Performance on the 12 datasets, best in **bold** and second-best in underline. HETEROGENEOUS SWARMS outperforms TRIVIAL, STATIC WEIGHT, DYNAMIC WEIGHT, STATIC ROLE, and DYNAMIC ROLE approaches by 18.5% on average across tasks.

Table 1. Source: Feng et al. (2025)

Experiment 1: Main Comparison

Results

- Finds effective multi-agent systems for several task types
  - H-Swarms achieves best performance on 11/12 datasets
- Roles and weights provide different advantages for different tasks
  - Knowledge tasks -> weight baselines perform better by 4.3%
  - Agent tasks -> role baselines perform by 9.2%
- Dynamic tends to perform better than static approaches

Experiment 2: Ablation Study

Design: Ablation study that compares performance of only weight step, only role step, and the full system

Key Insight:

- Weight matters more on knowledge tasks, role matters more on agent tasks, pattern seen in 10/12 datasets
- Role and weight have task-dependent importance, which supports the joint optimization approach

	Knowledge			Reasoning			Agent			Miscellaneous		
	MMLU-pro	K-Cross	COM2	GSM8k	NLGraph	Normad	GAIA-text	AB-kg	AB-ltp	Qasper	AbstainQA	WoW
Role Baselines	0.218	0.318	0.486	0.323	0.531	0.480	0.095	0.318	0.132	0.205	0.042	0.488
Weight Baselines	0.222	0.352	0.490	0.325	0.538	0.494	0.082	0.342	0.112	0.169	0.067	0.516
Ours w/o Role	0.242	0.352	0.515	0.392	0.530	0.564	0.107	0.317	0.140	0.222	0.133	0.539
Ours w/o Weight	0.237	0.342	0.492	0.363	0.588	0.557	0.143	0.325	0.164	0.241	0.119	0.510
Ours Full	0.312	0.450	0.579	0.481	0.660	0.588	0.250	0.425	0.215	0.266	0.220	0.590
Consistent?	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE

Table 2: Ablation study of removing the role-step or weight-step in HETEROGENEOUS SWARMS, comparing whether the importance of role/weight is consistent between baselines and our approach. The pattern is consistent in 10 out of 12 datasets, confirming that model roles and weights could have different levels of importance for varying tasks.

Table 2. Source: Feng et al. (2025)

Experiment 2: Ablation Study

Results

- Role and weight are crucial for optimal performance
  - Role-only and Weight-only H-Swarms demonstrate degraded performance across all datasets
- H-Swarm ablation performance is largely consistent with overall baseline trends
  - Consistency checks if the trends of baseline follows(does role baseline perform better alongside H-Swarms role only test)
  - Ex: H-Swarm's role step only iteration performing better on a dataset may be a feature of the dataset rather than a insight into H-Swarm performance

Experiment 3: Collab Gains

Design: Four datasets, group problems into buckets of how many of the 10 LLMs can solve them. Compare to the multi-LLM systems accuracy on each bucket.

Eval: Collaborative gain

- Weighted sum of system accuracy - expected accuracy across all buckets weighted by bucket size

Key Insight:

- Collaborative Gain is positive across all four datasets, even when no singular model can solve the problem, H-Swarms gets 18.1% correct
- System indicates emergent capabilities, the value of the components is greater than the sum of its parts

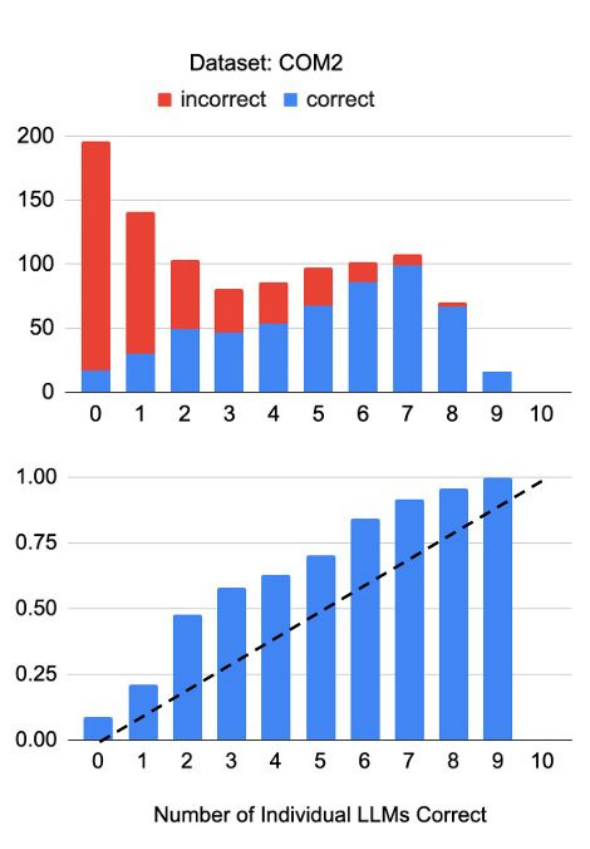


Figure 3. Source: Feng et al. (2025)

Results

- C-gain demonstrates collaborative capabilities
  - Does the system exceed the expected accuracy of it's components(dotted line)
  - Tested positive over all four datasets
  - Ex: 0.184
- Emergent capabilities
  - Bucket B_0 achieves 18.1% on average
    - Represents the problem set that no LLM was able to solve individually

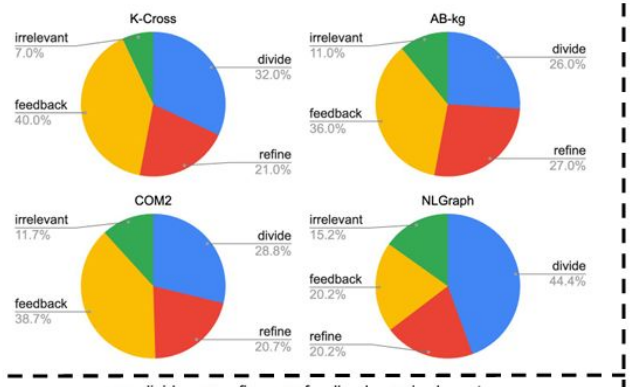
Experiment 4: Role Analysis

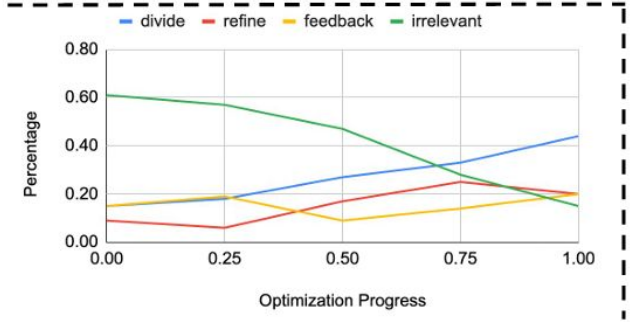
Design: First analyzes role distribution per dataset(how much of each role type appears across different tasks) then role distribution by DAG position(compares roles of early nodes verse later nodes)

Eval: Gemini-1.5-Flash used as judge to classify behavior of each LLM in the systems into four roles: divide, refine, feedback, and irrelevant.

Key Insight:

- Roles are task specific and position specific
  - Divide-graph reasoning, feedback-knowledge tasks
  - Early nodes divide, late nodes refine/feedback
- H-Swarm finds task-based structures with distinct roles without explicitly defining them





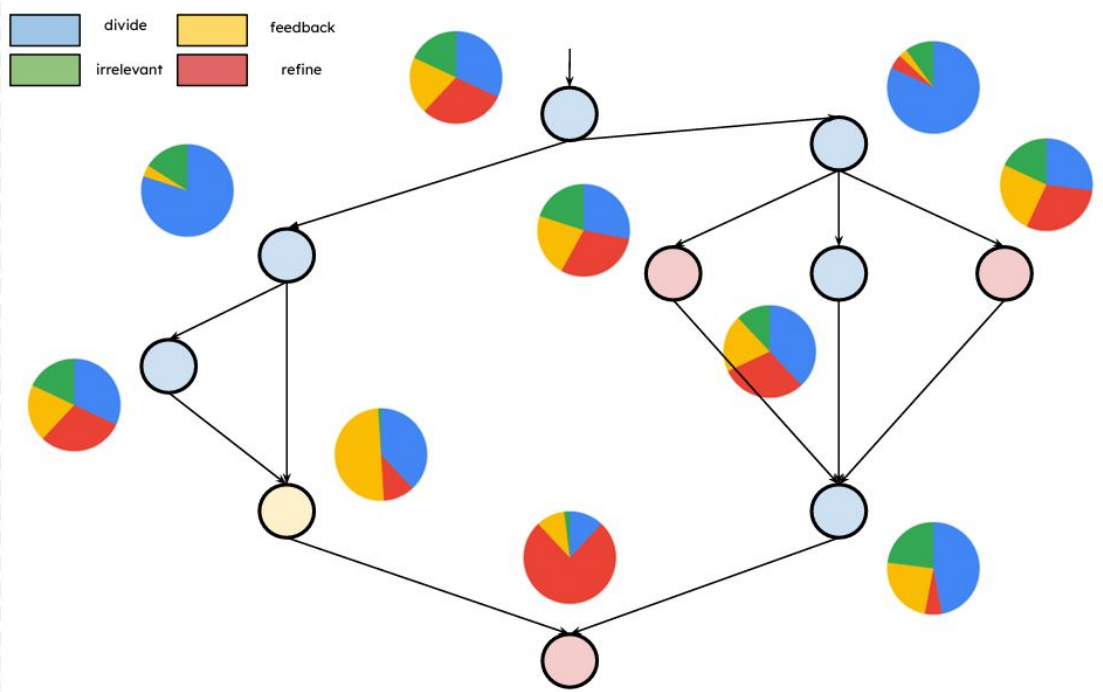


Figure 4: Analyzing the roles in Multi-LLM systems. Top left: the percentage of LLM roles aggregated per dataset. Bottom left: the change of LLM roles in the optimization process for NLGraph. Right: Per-LLM role distribution in the best-found multi-LLM system for NLGraph. Together these figures demonstrate the heterogeneous roles in the multi-LLM systems by HETEROGENEOUS SWARMS.

Figure 4. Source: Feng et al. (2025)

Experiment 4: Role Analysis

Results

- Role distributions are task specific
  - The divide role emerges for graph reasoning tasks(44.4% on NLGraph)
  - Feedback role emerges for knowledge tasks(40% on K-Crosswords)
- Roles are position specific
  - Earlier node typically take on divide roles
  - Late node typically take on refine/feedback roles
- H-Swarms finds optimal roles for tasks through iteration
  - The ‘irrelevant’ role outputs decrease over iterations while functional roles increase

Experiment 5: Sparsity Study

Design: Tests threshold pruning and L1 regularization to sparsify wasteful dense DAGs.

Eval: Measures latency and accuracy for each run

Key Insight:

- L1 regularization finds meaningful inference speedup with minimal accuracy loss
- Threshold pruning is more aggressive but hurts performance more
- It's possible to trade small accuracy loss for inference efficiency, sparsity into optimization(L1) is more effective than tradeoffs afterwards

Experiment 5: Sparsity

Results

- Sparsity should be built into optimization for computational efficiency tradeoff
  - L1 regularization during optimization 12.8% inference speedup with minimal accuracy loss
  - Post-hoc threshold pruning gets larger speedups at substantial cost

Experiment 6: Diversity Study

Design: With fixed model pool size of 10, vary the number of distinct models to understand if model diversity in the pool matters.

Key Insight:

- Accuracy improves consistently with diversity, with around 89% relative gain from least to most diverse
- Heterogeneity of the model pool indicates diverse expertise representing the value of collaboration

Experiment 6: Diversity

Results

- More diverse models increase accuracy
  - Fixing pool size at 10 but varying number of distinct models leads to 89% average relative accuracy gain from least to most diverse config(1x10, 2x5, etc...)

Discussion

Strengths:

1. Well supported joint optimization: neither role nor weights alone gives significant performance advantage
2. Strong collaborative gains from multiple models working together
3. Role specialization emerges from training steps organized topologically

Discussion

Limitations:

1. No ablation against a ‘homogenous’ swarm
2. True architectural model diversity untested
3. Evaluation bias from utility function

Discussion

- Interpretability vs capability tradeoff
- Scalability and optimization cost
- Utility function overfitting

Conclusion

- H-Swarms jointly optimizes what models do(role) and how they're tuned(weights), done by no other prior work, without using gradient updates
- Joint optimization performs well across task types
- System produces emergent collaborative capability
- Heterogeneity and dynamic adaption to task are important for performance
- Role and weight have task-dependent importance, which is why joint optimization is necessary

QA

Citations

Feng, Shangbin, et al. *Heterogeneous Swarms: Jointly Optimizing Model Roles and Weights for Multi-LLM Systems*. 2025. arXiv, https://arxiv.org/abs/2502.04510.

Feng, Shangbin, et al. *Model Swarms: Collaborative Search to Adapt LLM Experts via Swarm Intelligence*. 2025. arXiv, https://arxiv.org/abs/2410.11163.

“Adjacency Matrix Meaning and Definition in DSA.” *GeeksforGeeks*, 23 July 2025, https://www.geeksforgeeks.org/dsa/adjacency-matrix-meaning-and-definition-in-dsa/

Kennedy, J., and Russell Eberhart. “Particle Swarm Optimization.” *Proceedings of the International Conference on Neural Networks 1995*, 1995, pp. 1942–1948. IEEE, https://doi.org/10.1109/ICNN.1995.488968.

Kovitz, Joshua M. “Particle Swarm Optimization Illustration.” *Optimization in Electromagnetics*, 2026, personal website, https://joshuakovitz.com.