ClawBot-Matching: Bidirectional, Explainable, and Learnable Collaboration Matching in Mixed Human-Agent Networks

Jiayao Gu*
McGill University.
Quebec, Canada

Kexin Chu†
University of Connecticut
Connecticut, USA

Peidong Liu
Sichuan University
Sichuan, China

Yue Yang
Stanford University
California, USA

Lynn Ai
University of Electronic Science and Technology of China
Sichuan, China

Ling Yang
Princeton University
New Jersey, USA

Tianyu Shi‡
McGill University
Quebec, Canada

Abstract

Agent evaluation usually measures what an agent does after it receives a task. Mixed human-agent systems face an earlier evaluation question: which human, proxy agent, service agent, or reusable skill should join the task at all? We present **ClawBot-Matching**, a matching environment for Scientist ClawBot Hub. The system parses natural-language interaction into structured capabilities, needs, and constraints. It then uses MapScore to estimate bidirectional value, asking whether a candidate covers the requester's residual capability gap and whether the task satisfies the candidate's participation needs. Hard constraints such as consent, data access, safety, and availability act as gates. A coarse-to-fine planner combines greedy ranking, uncertainty-aware exploration, and LLM-based dream simulation for the top candidates. Online feedback updates capability priors and scoring weights. We frame this pipeline as an RL-style evaluation environment, where the state is a mixed human-agent network, the action is a proposed match, and the reward combines explicit user feedback, behavior signals, and downstream outcomes. The released prototype supports one-to-one ranking, one-to-many team construction, explanation cards, fixture scenarios, and regression tests. Code is available at github.com/kexinchu/clawbot-matching.

1 Introduction

LLM agents can retrieve information, write code, call tools, and coordinate with users. Many deployments still fail before execution starts. A requester may not know which collaborator has the missing domain skill. A platform may select a strong agent that lacks data permission. A human candidate may have the right expertise but no reason to accept the task. These failures do not test whether an assigned agent can act. Instead, they test whether the system can create the right collaboration context.

We study *collaboration matching* in mixed human-agent networks. A node may be a human researcher, a proxy agent that acts for a user, a service agent with tools, or a reusable skill. A match creates a task-conditioned edge. That edge can trigger negotiation, division of labor, tool use, artifact exchange, and later feedback. The decision should therefore answer four practical questions. What capability gap does the candidate fill? Why would the candidate participate? Which hard constraints block the match? How should the system explore new or uncertain candidates?

This problem focuses on agent evaluation methods and RL environment design. We treat the matcher as a policy. The environment is a changing collaboration network. The policy observes task and profile signals, proposes an edge or team, and receives feedback from users and task outcomes. Thus, it makes a usually hidden failure mode measurable, where a system can fail not because every agent is weak, but because it connected the wrong participants.

Existing work provides useful pieces but does not solve this setting. Recommender systems predict relevance or preference [2, 4, 10]. Multi-stakeholder recommendation adds platform and producer objectives [1, 5]. Expert team formation covers skill requirements over a network [8, 11]. Matching markets model two-sided preference and recent work studies learning under uncertainty [6, 9]. Recent multi-agent systems evaluate cooperation after participants are already chosen [7, 12, 13]. ClawBot-Matching focuses on which heterogeneous participant should be connected to the current task, under what explanation, and with what constraints?

We make three contributions. First, we formulate collaboration matching as a closed-loop evaluation problem for mixed human-agent networks. Second, we present MapScore, a bidirectional score that separates requester-side capability coverage, candidate-side participation value, and non-compensatory hard gates. Third, we describe a model-based planning loop that uses greedy ranking, UCB exploration, LLM dream simulation, match cards, and online feedback. The prototype and tests are released in the public GitHub repository.

2 Problem Setup

A matching episode contains a requester u, a task T, and a candidate pool C. The matcher returns a ranked list or a small team proposal. Each candidate $v \in C$ can be human or artificial. The system must support one-to-one matches, such as recommending a specialist

*Joint first author. jia.gu@mail.mcgill.ca
†Joint first author.
‡Corresponding author. tys@cs.toronto.edu

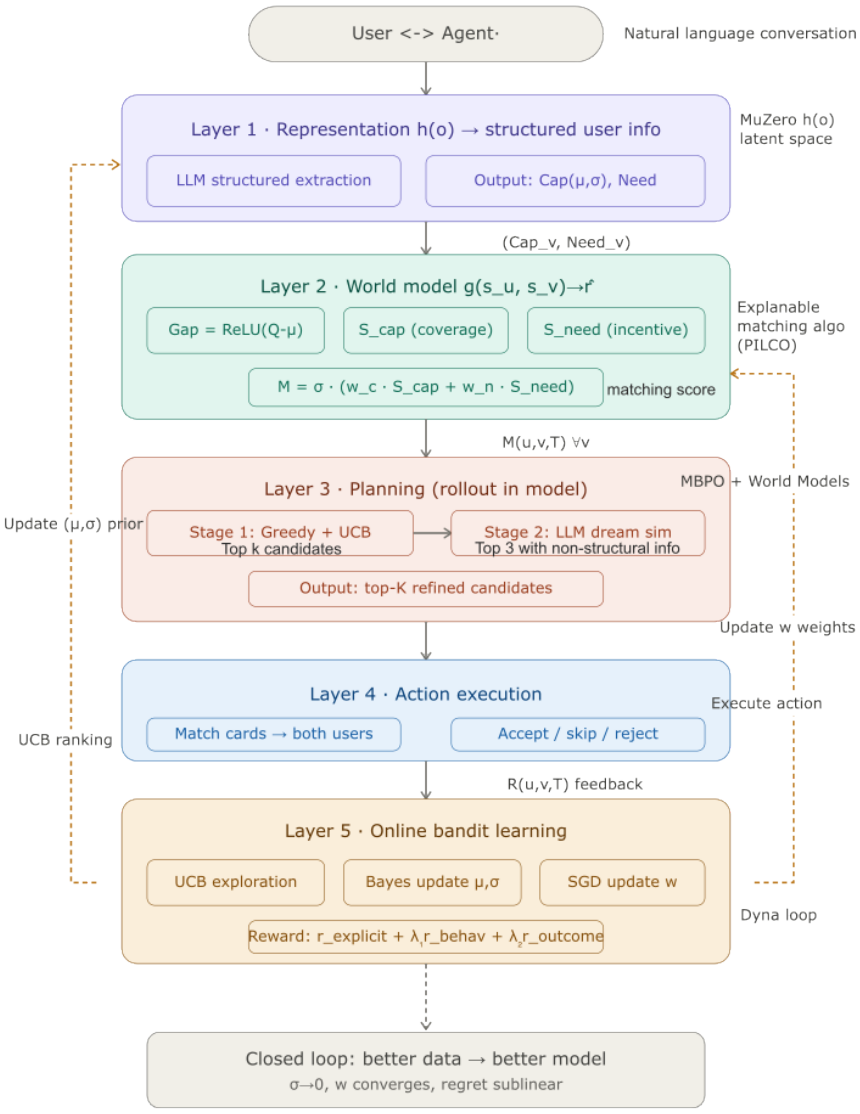

Figure 1: ClawBot-Matching workflow. Natural-language interaction enters a representation layer, MapScore acts as an explainable world model, planning uses greedy ranking and dream simulation, action execution produces match cards, and online bandit learning closes the loop.

agent, and one-to-many matches, such as building a team with complementary expertise.

Each node v stores capabilities, needs, and constraints:

$$\mathbf{s}_v = (\mathbf{Cap}_v, \mathbf{Need}_v, \mathbf{L}_v). \tag{1}$$

The capability set is open-ended:

$$\mathbf{Cap}_v = \{(\mathbf{e}_i^v, p_i^v, \sigma_i^v)\}_{i=1}^{n_v}, \tag{2}$$

where $\mathbf{e}_i^v$ is a semantic embedding, $p_i^v \in [0,1]$ is estimated proficiency, and σ_i^v is uncertainty. The same representation can hold a person's clinical expertise, a proxy agent's coding skill, a service agent's retrieval tool, or a reusable evaluation template.

A task is represented as

$$T = (G_T, \mathbf{Q}_T, \mathbf{O}_T, \mathbf{L}_T, \mathbf{X}_T), \tag{3}$$

where G_T is the goal, $\mathbf{Q}_T$ contains required capabilities, $\mathbf{O}_T$ contains offers or incentives, $\mathbf{L}_T$ contains task-level constraints, and $\mathbf{X}_T$ contains context. We separate requirements and offers. Requirements say what the requester needs. Offers say why a candidate would participate, such as authorship, payment, access to data, learning value, or a scoped service request.

The matcher should return more than a score. It should produce an explanation card that names covered gaps, satisfied needs,

passed gates, and remaining risks. This design makes each recommendation auditable. It also turns user feedback into structured supervision. A rejection can mean that the capability estimate was wrong, the incentive model was wrong, a gate was missing, or the proposed process was too costly.

3 Method

Figure 1 shows the full workflow. ClawBot-Matching has five layers. Layer 1 converts a natural-language conversation into structured user and task information. Layer 2 scores candidate edges with an explainable world model. Layer 3 plans over that model and simulates high-value actions. Layer 4 executes the recommendation through match cards and accept, skip, or reject actions. Layer 5 updates priors and weights from online feedback.

3.1 Bidirectional MapScore

MapScore estimates the value of connecting requester u and candidate v for task T:

$$M(u,v,T) = \sigma(u,v,T)\big[\lambda S_{\text{cap}}(u,v,T) + (1-\lambda)S_{\text{need}}(v,T)\big]. \tag{4}$$

The term S_{cap} measures how much v covers the requester's residual capability gap. The term S_{need} measures how well the task satisfies v's participation needs. The weight $\lambda \in (0,1)$ balances the two sides and can be learned from feedback.

For each requirement j, the model estimates requester coverage $\tilde{p}_u^j$ and computes the residual gap:

$$\text{Gap}_j(u,T) = \max(0, q_j - \tilde{p}_u^j). \tag{5}$$

Candidate coverage contributes only where this gap remains. This choice matters in science and engineering tasks because complementary expertise often matters more than topical similarity.

Hard feasibility uses a binary gate:

$$\sigma(u,v,T) = \prod_{r \in \mathcal{R}(u,v,T)} \mathbb{I}[r=1]. \tag{6}$$

The rule set $\mathcal{R}$ can include safety policy, consent, data clearance, availability, conflict of interest, tool permission, and task prerequisites. If any rule fails, the candidate is filtered before ranking. High capability cannot compensate for a missing approval.

3.2 Coarse-to-Fine Planning

The planner first applies MapScore to all feasible candidates. For one-to-one matching, it ranks the candidates and returns the top options. For one-to-many matching, it greedily adds the candidate with the largest marginal reduction in the residual gap while preserving candidate-side value. This gives a scalable approximation to coverage-based team formation [8].

The analytical score cannot capture every process failure. A candidate may have the right skill but little time. Two candidates may cover the same role. A service agent may produce outputs that are hard to integrate. ClawBot-Matching therefore runs a slower simulation only on the top-K candidate actions. The simulator estimates timeline fit, communication cost, priority alignment, role ambiguity, and other risk signals. The final ranking is

$$R(a) = \eta M_a + (1-\eta)S_{\text{sim}}(u,a,T), \quad \eta \in [0,1], \tag{7}$$

where a is a candidate or team action, M_a is its analytical value, and S_{sim} is simulated compatibility.

3.3 Action Execution and Match Cards

The execution layer presents each proposed match as a card. A card contains four claims: the gap the candidate covers, the value the task offers to the candidate, the gates that passed or require approval, and the process risks found by simulation. The user can accept, skip, or reject the recommendation. The candidate can also accept or reject when consent is required.

This interface supports explanation and learning at the same time. If the user rejects a card because the candidate lacks data clearance, the system updates gate logic or profile state. If the candidate rejects because the incentive is weak, the system updates the need model. If both accept but the collaboration fails because roles are unclear, the simulator receives a process-level error signal. Each card therefore becomes a compact evaluation trace.

3.4 Feedback Learning and Exploration

After each episode, the system observes explicit feedback, behavior signals, and outcomes. Explicit feedback includes accept, skip, reject, ratings, and corrections. Behavior signals include response time, follow-through, and communication load. Outcomes include task completion, quality, cost, and later user satisfaction. The reward used by the learning loop is

$$r = r_{\text{explicit}} + \lambda_1 r_{\text{behav}} + \lambda_2 r_{\text{outcome}}. \tag{8}$$

This reward updates capability priors (μ, σ) and score weights w. It also supports exploration. A simple upper-confidence retrieval rule is

$$\text{Select}_t(v) = M(u,v,T) + c\,\bar{\sigma}_v \sqrt{\frac{\log(t+1)}{n_v+1}}, \tag{9}$$

where $\bar{\sigma}_v$ is candidate uncertainty and n_v is the number of prior interactions. The bonus gives new humans, agents, and skills a chance to appear in the shortlist. As feedback accumulates, uncertainty shrinks and the policy shifts toward exploitation [3].

4 Evaluation Environment

We evaluate ClawBot-Matching as an RL-style environment for collaboration decisions. The state contains the current network, public profiles, task context, and uncertainty estimates. The action is a proposed edge or team. The transition records whether users accept, whether gates hold, how the collaboration proceeds, and how node states change. The reward combines capability coverage, candidate-side value, constraint safety, process cost, and task outcome.

This setup supports four evaluation layers. Offline replay uses logged or curated tasks and asks whether the matcher ranks feasible, complementary candidates. Simulation stress tests control missing profiles, noisy capabilities, sparse feedback, hard-gate errors, and candidate growth. Human study measures whether users understand the match cards and trust the stated reasons. Online hub evaluation measures acceptance rate, completion rate, time to collaboration, quality, and unwanted interruption.

We compare against four baselines. A similarity baseline retrieves candidates whose descriptions are closest to the task. A

Metric	What it tests
Gap coverage	Whether selected candidates reduce residual task requirements.
Mutual value	Whether task offers fit candidate needs and increase acceptance.
Gate safety	Whether the policy blocks consent, access, safety, and eligibility violations.
Simulation value	Whether dream simulation catches role conflict, timing mismatch, and communication cost.
Cold start	Whether uncertain but useful candidates receive exposure without excessive regret.
Card fidelity	Whether match-card reasons match score components and user judgments.

Table 1: Evaluation targets for the matching policy. The metrics expose different failure modes, so no single scalar score is sufficient.

capability-only baseline ranks by residual gap coverage. A need-only baseline ranks by candidate-side participation value. A feasible-random baseline samples among candidates that pass hard gates. We also run ablations without hard gates, without simulation, without candidate needs, and without UCB exploration. These tests show which component protects each metric in Table 1.

The evaluation should also test reward hacking. A policy that maximizes acceptance may recommend only easy tasks. A policy that maximizes capability coverage may overuse popular experts and ignore consent burden. A policy that maximizes exploration may interrupt too many users. We therefore report metric vectors rather than one reward alone. We also inspect match-card traces so that users can see why the policy selected a candidate.

5 Prototype Case Study

Scientist ClawBot Hub motivates the prototype. A requester describes a research goal in natural language, such as building an evaluation set for a biomedical retrieval agent, reproducing a robotics ablation, or forming a team for a multimodal triage paper. The representation layer extracts required capabilities, task constraints, available offers, and contextual details. The candidate pool can include human researchers, proxy agents, service agents, and reusable skills.

Consider a requester who wants to build a multimodal patient-triage paper. The task needs clinical data analysis, multimodal modeling, evaluation design, and paper writing. It offers authorship, dataset access, and a clear research direction. Candidate nodes include a clinical collaborator, a statistics expert, a writing agent, a retrieval agent, and a data-access service. A good policy should select complementary nodes, reject candidates without data clearance, avoid redundant expertise, and explain every proposed edge.

A typical match card might say that a clinical collaborator covers the medical-labeling gap, that authorship and access to a concrete dataset support candidate value, and that institutional data approval remains required. For a service agent, the card might say that the agent covers benchmark generation but needs compute permission. For a reusable skill, the card might say that the skill covers only one subtask and should be paired with an evaluation-design candidate. These claims make feedback actionable. The user can correct the missing capability, incentive, gate, or process risk.

The released repository contains a modular implementation of the scoring and planning stack. It includes simple encoders, optional semantic encoders, fixture generation, deterministic one-to-one ranking, residual-gap team construction, explanation-card outputs, and regression tests. The current system provides a reproducible environment for studying how collaboration matching affects agent evaluation.

6 Limitations, Safeguards, and Conclusion

ClawBot-Matching depends on profile quality. Human profiles may be incomplete, and agent capability descriptions may be overconfident. The simulator can miss social factors that affect real collaboration. Feedback may be sparse, delayed, or biased toward users who already engage with the hub. These limits matter because the matcher can shape who receives opportunities, credit, and interruptions.

We therefore treat the current system as an evaluation environment before treating it as an autonomous deployment tool. A safe rollout should require user-auditable gates, opt-out controls, rate limits for invitations, and human approval for sensitive data or high-cost service agents. The system should log why a match passed, which profile fields it used, and which uncertainty term justified exploration. It should not infer sensitive constraints or personal attributes without explicit consent. It should also separate a person's worth from a task-specific capability estimate.

The reward design needs similar care. Acceptance rate alone can reward shallow matches. Outcome quality alone can overuse already-visible experts. Exploration alone can create unwanted interruptions. Our planned metric vector tests these tradeoffs by reporting coverage, mutual value, gate safety, simulation value, cold-start behavior, and card fidelity together. Future work should calibrate uncertainty with real traces, measure long-term fairness, and compare human-only, agent-only, and mixed teams under the same task distribution.

ClawBot-Matching reframes matching as an evaluation problem for mixed human-agent systems. It asks whether a proposed participant is useful, feasible, mutually valuable, and explainable before downstream work begins. By combining bidirectional scoring, hard gates, simulation, and online learning, the system turns collaboration formation into an auditable RL-style environment.

References

[1] Himan Abdollahpouri, Gediminas Adomavicius, Robin Burke, Ido Guy, Dietmar Jannach, Toshihiro Kamishima, Jan Krasnodebski, and Luiz Pizzato. 2020. Multistakeholder Recommendation: Survey and Research Directions. *User Modeling and User-Adapted Interaction* 30, 1 (2020), 127–158. doi:10.1007/s11257-019-09256-1

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State of the Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering* 17, 6 (2005), 734–749. doi:10.1109/TKDE.2005.99

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2–3 (2002), 235–256. doi:10.1023/A:1013689704352

[4] Jesus Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-Based Systems* 46 (2013), 109–132. doi:10.1016/j.knosys.2013.03.012

[5] Robin Burke, Gediminas Adomavicius, Toine Bogers, Tommaso Di Noia, Dominik Kowald, Julia Neidhardt, Ozlem Ozgobek, Maria Soledad Pera, Nava Tintarev, and Juergen Ziegler. 2025. De-centering the (Traditional) User: Multistakeholder Evaluation of Recommender Systems. *International Journal of Human-Computer Studies* 203 (2025), 103560. doi:10.1016/j.ijhcs.2025.103560

[6] David Gale and Lloyd S. Shapley. 1962. College Admissions and the Stability of Marriage. *The American Mathematical Monthly* 69, 1 (1962), 9–15. doi:10.1080/00029890.1962.11989827

[7] Harsh Jhamtani, Jacob Andreas, and Benjamin Van Durme. 2025. LLM Agents for Coordinating Multi-User Information Gathering. In *Findings of the Association for Computational Linguistics: ACL 2025*. 17800–17826. doi:10.18653/v1/2025.findings-acl.916

[8] Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a Team of Experts in Social Networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 467–476. doi:10.1145/1557019.1557074

[9] Shuai Li, Zilong Wang, and Fang Kong. 2025. A Survey on Bandit Learning in Matching Markets. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*. 10546–10554. doi:10.24963/ijcai.2025/1171

[10] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor (Eds.). 2011. *Recommender Systems Handbook*. Springer, New York. doi:10.1007/978-0-387-85820-3

[11] Mahdis Saeedi, Hawre Hosseini, Christine Wong, and Hossein Fani. 2025. A Survey of Subgraph Optimization for Expert Team Formation. *Comput. Surveys* 57, 12 (2025), 1–40. doi:10.1145/3737455

[12] Haochen Sun, Shuwen Zhang, Lujie Niu, Lei Ren, Hao Xu, Hao Fu, Fangkun Zhao, Caixia Yuan, and Xiaojie Wang. 2025. Collab-Overcooked: Benchmarking and Evaluating Large Language Models as Collaborative Agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 4922–4951. doi:10.18653/v1/2025.emnlp-main.249

[13] Song Wang, Zhen Tan, Zihan Chen, Shuang Zhou, Tianlong Chen, and Jundong Li. 2025. AnyMAC: Cascading Flexible Multi-Agent Collaboration via Next-Agent Prediction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 11555–11567. doi:10.18653/v1/2025.emnlp-main.584

A Additional Method Details

A.1 Attention-Based Semantic Matching

We instantiate capability and offer matching with attention over open-set semantic items. Task requirements are written as

$$\mathbf{Q}_T = \{(\mathbf{q}_j, q_j, w_j^T)\}_{j=1}^{m},$$

where $\mathbf{q}_j$ is a requirement embedding, q_j is the required level, and w_j^T is its task-specific importance. Candidate needs and task offers are written as

$$\mathbf{Need}_v = \{(\mathbf{z}_l^v, n_l^v)\}_{l=1}^{h_v}, \qquad \mathbf{O}_T = \{(\mathbf{o}_k, o_k)\}_{k=1}^{r}.$$

For each task requirement $\mathbf{q}_j$, requester-side coverage is computed by attention over requester capabilities:

$$\alpha_{i,j}^u = \frac{\exp(\mathrm{sim}(\mathbf{q}_j, \mathbf{e}_i^u)/\tau)}{\sum_k \exp(\mathrm{sim}(\mathbf{q}_j, \mathbf{e}_k^u)/\tau)}, \qquad \tilde{p}_u^j = \sum_i \alpha_{i,j}^u p_i^u.$$

Candidate-side proficiency and uncertainty are computed analogously as $\tilde{p}_v^j$ and $\tilde{\sigma}_v^j$. We use an optimistic coverage estimate

$$c_v^j = \min(1, \tilde{p}_v^j + \beta_t \tilde{\sigma}_v^j),$$

where β_t controls uncertainty-aware optimism. The capability-gap score is

$$S_{\mathrm{cap}}(u, v, T) = \frac{\sum_j w_j^T \min(\mathrm{Gap}_j(u, T), c_v^j)}{\sum_j w_j^T \mathrm{Gap}_j(u, T) + \epsilon}.$$

For participation-value fit, each candidate need attends over task offers:

$$\gamma_{k,l}^v = \frac{\exp(\mathrm{sim}(\mathbf{z}_l^v, \mathbf{o}_k)/\tau_o)}{\sum_{k'} \exp(\mathrm{sim}(\mathbf{z}_l^v, \mathbf{o}_{k'})/\tau_o)}, \qquad \hat{o}_l^T = \sum_k \gamma_{k,l}^v o_k.$$

The need satisfaction score is

$$S_{\mathrm{need}}(v, T) = \frac{\sum_l n_l^v \min(\hat{o}_l^T, n_l^v)}{\sum_l n_l^v + \epsilon}.$$

A.2 Residual-Gap Team Construction

For one-to-N matching, the planner maintains a residual capability gap $\mathbf{R}$ and greedily adds the feasible candidate with the largest marginal value:

$$\Delta(v \mid \mathcal{S}) = \sigma(u, v, T)\left[\rho \frac{\sum_j w_j^T \min(R_j, c_v^j)}{\sum_j w_j^T R_j + \epsilon} + (1-\rho) S_{\mathrm{need}}(v, T)\right],$$

where ρ balances residual capability coverage and candidate-side value.

A.3 Feedback Update Equations

When feedback provides evidence about a capability item, we update its proficiency and uncertainty with a Gaussian posterior update. For an observation x_t with noise variance σ_{obs}^2 and prior $(\mu_{i,t}^v, (\sigma_{i,t}^v)^2)$,

$$\mu_{i,t+1}^v = \frac{\mu_{i,t}^v/(\sigma_{i,t}^v)^2 + x_t/\sigma_{\mathrm{obs}}^2}{1/(\sigma_{i,t}^v)^2 + 1/\sigma_{\mathrm{obs}}^2},$$

Algorithm 1 Residual-gap team construction

$\mathcal{S} \leftarrow \emptyset$; $\mathbf{R} \leftarrow \mathbf{Gap}(u, T)$
while $|\mathcal{S}| < n_{\max}$ and $\|\mathbf{R}\|_1 > \epsilon$ **do**
  $v^* \leftarrow \arg\max_{v \in C \setminus \mathcal{S}} \Delta(v \mid \mathcal{S})$
  if $\Delta(v^* \mid \mathcal{S}) \le 0$ **then**
    break
  end if
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{v^*\}$
  $R_j \leftarrow \max(0, R_j - c_{v^*}^j)$ for all j
end while
return $\mathcal{S}$

$$(\sigma_{i,t+1}^v)^2 = \left(\frac{1}{(\sigma_{i,t}^v)^2} + \frac{1}{\sigma_{\mathrm{obs}}^2}\right)^{-1}.$$

We update the balance parameter λ by minimizing prediction loss between observed collaboration outcome y_t and predicted success:

$$\mathcal{L}_t = (\hat{y}_t - y_t)^2, \qquad \hat{y}_t = \lambda S_{\mathrm{cap}}(u, v, T) + (1-\lambda) S_{\mathrm{need}}(v, T).$$