NETWORK WORLD MODELS AS ENVIRONMENTS FOR ALGORITHM DESIGN ON COMPLEX SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

World models, which simulate an environment and predict how it changes under actions, are increasingly used in real-world applications such as robotics. Complex systems call for the same tool because the effect of an action is not immediate. Seeding nodes for a campaign, or immunizing nodes against an epidemic, changes little on its own; what matters is the outcome that unfolds over the steps that follow. Designing an algorithm that selects such actions to maximize expected performance on a task is inherently iterative, and every candidate must be scored by the outcome it produces. Obtaining that outcome has relied on simulation, whose cost becomes a bottleneck when candidates are evaluated over many sampled trajectories. We propose an action-conditioned Network World Model that learns a network's diffusion dynamics under interventions over time, applies each action to the network, and predicts the outcome that follows. It serves as a fast evaluator inside an algorithm design loop in which a coding agent designs and refines executable algorithms using feedback from full rollouts, action-level credit, and counterfactual probes over alternative interventions. Across eight network tasks and five diffusion models, the designed algorithms match or exceed the strongest reported baseline in 118 of 121 settings while enabling up to 14.5 times faster rollouts than Monte Carlo simulation. Our project's full code is available at `https://anonymous.4open.science/r/NetworkWorldModel`.

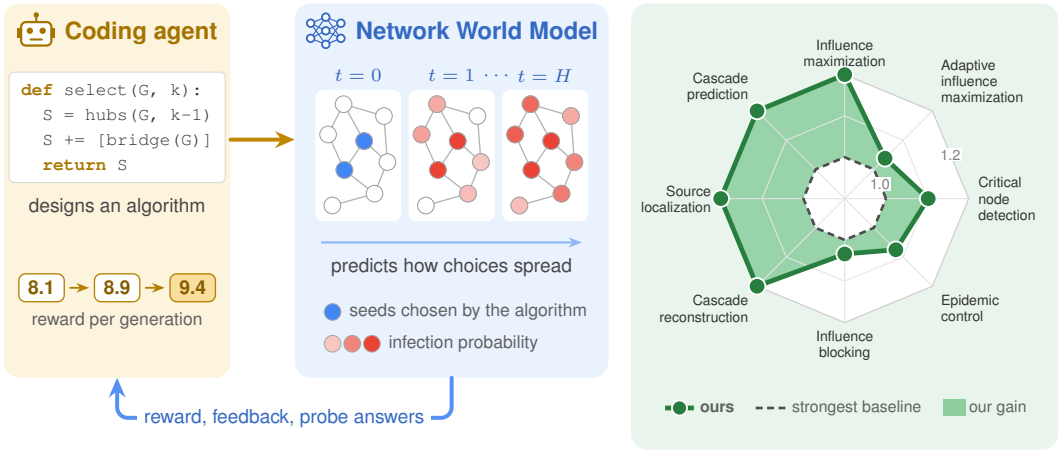


Figure 1: A coding agent writes algorithms that choose interventions on a network, and a Network World Model rolls out how each choice unfolds, returning a reward, feedback, and probe answers the agent uses to improve the algorithm (left). Across eight tasks on complex networks, the agent-designed algorithms outperform the strongest baselines, averaged over budgets (right).

1 INTRODUCTION

Complex systems such as social, contact and information networks are dynamic in two ways: their state evolves on its own, and it also changes under actions taken on the system (Pastor-Satorras et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2015). Choosing which actions to take to maximize task performance is a combinatorial problem that appears across domains, from selecting the users who will carry adoption farthest (Kempe et al., 2003) to choosing who to vaccinate before an outbreak spreads (Pastor-Satorras & Vespignani, 2002; Cohen et al., 2003) or seeding a correction ahead of a rumor (Budak et al., 2011). One of the major roadblocks in designing algorithms for these choices lies in evaluation: how do we know what an action will lead to before taking it? Classically, the answer is simulation, called inside the search once per candidate (Kempe et al., 2003; Tang et al., 2014; Han et al., 2018), since computing the expected outcome is #P-hard (Chen et al., 2010a). Learning-based methods handle this problem by fitting a mapping from a network and a chosen set to the quantity of interest (Xia et al., 2021; Ko et al., 2020), or by predicting the solution itself (Fan et al., 2020; Ling et al., 2023), but each is fit to a particular task and objective. This creates a trade-off between generality and cost: simulation supports arbitrary candidate interventions but requires repeated process execution, whereas learned mappings are inexpensive but typically specialized to a particular task and objective.

Designing effective algorithms for these problems often requires substantial heuristic design and empirical tuning. LLM-based program search can automate part of this process by generating, evaluating, and refining algorithmic code (Romera-Paredes et al., 2024; Mankowitz et al., 2023; Novikov et al., 2025; Liu et al., 2024; Ye et al., 2024; Van Stein & Bäck, 2024; Zheng et al., 2025). Such a search process needs an environment in which the consequences of candidate actions can be evaluated. In other domains, world models serve this role by learning environment dynamics in which an agent can act (Ha & Schmidhuber, 2018; Hafner et al., 2025; Schrittwieser et al., 2020; Bruce et al., 2024). Existing world models largely operate on images, video, point clouds, or low-dimensional control states. Yet network systems require modeling both the relational structure through which interactions propagate and how interventions modify the system and its subsequent dynamics. This raises the central question of this work: can we learn such an action-conditioned world model and use it as a reliable and efficient environment for algorithm design?

Complex systems pose three challenges for world modeling and algorithm design. First, actions may modify node states, edges, or contact weights, changing both the immediate system configuration and the subsequent diffusion dynamics. Conventional world models evolve states within fixed observation spaces (Hafner et al., 2025; Schrittwieser et al., 2020), while learned diffusion surrogates often fix the intervention type, such as a seed set, and predict a specific objective (Xia et al., 2021; Ko et al., 2020; Ling et al., 2023). Second, long-horizon evaluation introduces compounding transition error, so accurate next-state prediction does not necessarily preserve the relative performance of candidate algorithms over repeated rollouts. Third, evaluating candidate algorithms with Monte Carlo simulation is costly, while aggregate task scores provide limited guidance for revision, and evaluating alternatives, such as swapping a seed or blocking a different edge, incurs simulation cost.

To address these challenges, we propose a Network World Model and an LLM-based algorithm design framework (Figure 1). The model is a graph neural simulator that applies interventions exactly to node states, edges, and contact weights, then predicts subsequent diffusion through a transition head constrained by the process's known rules. We develop a coding agent search loop that uses fast model rollouts to score candidate algorithms, estimate the contribution of individual actions through ablation, and evaluate alternatives such as node swaps, providing targeted feedback for algorithm revision. We further provide a theoretical analysis that bounds algorithm selection regret in terms of transition error, rollout horizon, and search suboptimality, and shows how exact intervention handling tightens this bound. Across eight tasks and five diffusion models, the framework matches or exceeds the strongest reported baseline in 118 of 121 settings while enabling up to 14.5$\times$ faster rollouts than Monte Carlo simulation. **Our main contributions are: (1)** an action-conditioned Network World Model for complex network dynamics; **(2)** an LLM-based framework for network algorithm design using the learned model; and **(3)** theoretical guarantees for world-model-based algorithm selection together with extensive evaluation across eight tasks and five diffusion models.

2 Related Work

Algorithms for complex network dynamics. Classical methods for influence, containment, and inverse diffusion problems typically rely on repeated simulation or task specific structural objectives (Kempe et al., 2003; Pastor-Satorras & Vespignani, 2002; Budak et al., 2011; Shah & Zaman, 2011). Learned alternatives reduce this cost by approximating the forward process, directly predict-

ing solutions, or learning generative priors over diffusion trajectories, but are generally specialized to a particular task, objective, or intervention form (Xia et al., 2021; Fan et al., 2020; Ling et al., 2022; Yan et al., 2023; Memon et al., 2025).

World models. World models learn a transition function and plan or learn a policy against it in place of the environment (Ha & Schmidhuber, 2018), either by compressing observations and rolling them forward in a latent space (Hafner et al., 2019b;a) or by training the model to support search (Schrittwieser et al., 2020). Predominantly, they are learned over video frames or low-dimensional control states, where the observation keeps a fixed shape, however the agent acts. On graphs, the pieces exist separately: learned solvers predict the next state by message passing but take no action (Battaglia et al., 2016; Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020); the estimators above take an action but return only a final number; and graph world models act on multimodal agent states, where an action names a task and not a change to the network (Feng et al., 2025). A complex system requires both a latent representation structured by the graph and dynamics that respond to an action which changes that structure (Memon et al., 2026).

Algorithm discovery with language models. Language models can write a program, score it, and revise it, and this work has converged on an evolutionary loop over a population of candidates. FunSearch pairs a model with a program database and selects on the exact objective (Romera-Paredes et al., 2024); EoH, ReEvo, LLaMEA and MCTS-AHD organize the same loop as an evolution over heuristics, adding reflection, tree search, or mutation operators (Liu et al., 2024; Ye et al., 2024; Van Stein & Bäck, 2024; Zheng et al., 2025); AlphaEvolve scales it to whole programs (Novikov et al., 2025). Reinforcement learning is the other learned route on graphs, training a policy that emits a solution instead of a procedure that produces one (Khalil et al., 2017; Manchanda et al., 2020). Across these systems, the evaluator is taken as given, since the objective is cheap and the loop's cost is the model's. Graph diffusion breaks that assumption: scoring one candidate means thousands of simulations. We supply the evaluator and make it return diagnostics and counterfactual answers.

3 METHODOLOGY

3.1 PROBLEM FORMULATION

Let $\mathcal{G}$ be a complex network with node set $\mathbb{V}$, such as a social network or a power grid, carrying a state $\boldsymbol{s}_t$ at discrete time t, and let a_t be an action applied to $\mathcal{G}$ at time t. The network evolves as

$$\mathbf{s}_{t+1} \sim P(\cdot \mid \boldsymbol{s}_t, a_t), \tag{1}$$

where P gives the distribution of the next state given the current state and the action applied to it. For the processes we study, the distribution of the outcome of the transition P after H steps is not simple: computing expected spread is #P-hard under IC (Chen et al., 2010a). The score of a candidate must therefore be estimated from many sampled trajectories, which is what makes it expensive. Actions are produced by an algorithm π, and let H be the horizon of the task, the number of timesteps after which its outcome is read. Applying the actions returned by π to $\mathcal{G}$ and advancing Eq. (1) for H steps yields the outcome of the plan $\boldsymbol{a}$, which receives a score $R(\boldsymbol{a})$, the metric the task defines for that outcome. The actions must satisfy constraints $\mathbb{F}$, for example how many actions may be applied, which operations are allowed, and which elements of the network must be left intact. A task instance is formally represented by $\mathcal{T} = (\mathcal{G}, P, \mathbb{F}, H, \boldsymbol{s}_0)$ with initial state $\boldsymbol{s}_0$, and the plan π produces is $\boldsymbol{a} = (a_0, \ldots, a_{H-1})$.

The goal is to produce an algorithm that returns actions satisfying constraints and attaining the highest expected score at the horizon. The score is obtained by executing actions on the complex network and letting it evolve, which is required for every candidate algorithm and for queries about actions not yet applied. The problem therefore splits into two parts: learning a world model of the network dynamics (Section 3.2) and searching for algorithms against that world model (Section 3.3). We train a world model $f_{\boldsymbol{\theta}}$ on transitions $(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1})$ observed on the network by minimizing

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}\big[-\log f_{\boldsymbol{\theta}}(\boldsymbol{s}_{t+1} \mid \mathcal{G}_t, \boldsymbol{s}_t, a_t)\big]. \tag{2}$$

Algorithm design solves

$$\pi^\star \in \arg\max_{\pi} J_P(\pi), \qquad J_P(\pi) = \mathbb{E}\big[R(\pi(\mathcal{T}))\big], \qquad \text{subject to} \qquad \boldsymbol{a} = \pi(\mathcal{T}) \in \mathbb{F}, \tag{3}$$

where J_P is the expected score under the trusted transition kernel P, and the expectation is over the transition randomness in Eq. (1). This requires $f_{\boldsymbol{\theta}}$ to capture the effect of changing actions, remain accurate over the horizon H, and generalize across task instances sharing the same dynamics.

3.2 Network World Model of Complex Networks

The Network World Model $f_{\boldsymbol{\theta}}$ predicts the state of the network one step ahead, $\boldsymbol{s}_{t+1}$, from the current state $\boldsymbol{s}_t$ and the action a_t applied to it. An action changes the network in two ways: directly, through an immediate effect that the action determines and that is known exactly, and indirectly, through the diffusion that follows. The functional form of the one-step diffusion is given by the process, but its parameters are in general not observed and must be inferred from recorded transitions. The direct effect is applied, and the dynamics are learned,

$$(\mathcal{G}_t^+, \boldsymbol{s}_t^+) = T_{\text{exo}}(\mathcal{G}_t, \boldsymbol{s}_t, a_t), \qquad \hat{\boldsymbol{s}}_{t+1} = f_{\boldsymbol{\theta}}(\mathcal{G}_t^+, \boldsymbol{s}_t^+, \boldsymbol{\eta}(a_t)), \tag{4}$$

which is the factorized form of the model in Eq. (2): T_{exo} applies the action by modifying the network, its parameters, or the state as the action specifies, $(\mathcal{G}_t^+, \boldsymbol{s}_t^+)$ is the resulting post-intervention network and state, before the diffusion takes its step, and $\boldsymbol{\eta}(a_t)$ are features describing the operations the action applies. Theoretical analysis in Appendices A.4 and A.5 shows that applying the intervention exactly removes intervention semantics error from the rollout bound and consequently tightens the downstream algorithm selection guarantee.

Action conditioning. The action reaches the model through two channels: the post intervention network and state, into which T_{exo} has already written it, and a set of features $\boldsymbol{\eta}(a_t)$ carried alongside it. The second channel is needed because the state records what changed but not which operation caused that change, and distinct operations can produce similar post intervention states. The features record, for each modified element, the operation type and, when parameterized, its magnitude. They decompose into node level features $\boldsymbol{\eta}_v(a_t)$ and edge level features $\boldsymbol{\eta}_{uv}(a_t)$, concatenated to the node input of the message passing network and its edge map,

$$\boldsymbol{h}_v^{(0)} = \phi\big([\, \boldsymbol{s}_{t,v}^+,\ \boldsymbol{\eta}_v(a_t)\,]\big), \qquad e_{uv} = \psi\big([\, \boldsymbol{h}_u,\ \boldsymbol{h}_v,\ w_{uv},\ \boldsymbol{\eta}_{uv}(a_t)\,]\big), \tag{5}$$

Here, $\boldsymbol{s}_{t,v}^+$ is the post intervention state of node v, $\boldsymbol{h}_v^{(0)}$ its initial representation, $\boldsymbol{h}_u$ and $\boldsymbol{h}_v$ the representations returned by message passing, w_{uv} the edge weight, and ϕ and ψ the node and edge maps. For an edited edge, $\boldsymbol{\eta}_{uv}(a_t)$ records whether the edge was modified and its new weight; node operations are encoded analogously. Message passing runs on $\mathcal{G}_t^+$, so structural edits alter aggregation: removed edges carry no messages, while downweighted edges contribute proportionally less.

Transition head. The model of Eq. (4) runs in two stages. Message passing advances the system one step from the post-intervention network and state, so $\boldsymbol{h}_v$ represents the state of node v at $t+1$, and a readout g reports that state. The transition constrains g in two ways. First, T_{exo} has already set the nodes affected by the action, so g is applied only to the remainder. Writing $\mathbb{U}_t$ for that set,

$$\hat{\boldsymbol{s}}_{t+1,v} = g(\boldsymbol{h}_v) \ \text{ for } v \in \mathbb{U}_t, \qquad \hat{\boldsymbol{s}}_{t+1,v} = \boldsymbol{s}_{t,v}^+ \ \text{ for } v \notin \mathbb{U}_t. \tag{6}$$

Second, when the process form is known, g is constrained to that form, leaving only its parameters to be learned. For a cascade process, for example, $g(\boldsymbol{h}_v) = 1 - \prod_{u \to v}(1 - q_{uv} z_u)$, where $q_{uv} = e_{uv}$ is the learned transmission probability and z_u indicates whether u lies on the post intervention frontier. Training applies Eq. (2) one step at a time and node by node, using cross-entropy between $g(\boldsymbol{h}_v)$ and the conditional next state target over $\mathbb{U}_t$.

Rollout. A candidate plan $\boldsymbol{a}$ is scored by running the model forward to the horizon. A trajectory starts at $\boldsymbol{s}_0$ and repeats the same three operations at each step: the action scheduled for that step is applied by T_{exo}, the model returns a probability for each node in $\mathbb{U}_t$, and a state is drawn from those probabilities and carried into the next step,

$$(\mathcal{G}_t^+, \hat{\boldsymbol{s}}_t^{+(j)}) = T_{\text{exo}}(\mathcal{G}_t, \hat{\boldsymbol{s}}_t^{(j)}, a_t), \qquad \hat{\boldsymbol{s}}_{t+1}^{(j)} \sim f_{\boldsymbol{\theta}}(\mathcal{G}_t^+, \hat{\boldsymbol{s}}_t^{+(j)}, \boldsymbol{\eta}(a_t)), \qquad t = 0, \ldots, H-1. \tag{7}$$

The score of trajectory j is read from $\hat{\boldsymbol{s}}_H^{(j)}$. Drawing a state instead of carrying the probabilities forward keeps each rollout consistent with the underlying process: a vector of marginals holds every

node partially active at once and discards the dependence between them, which is what the score at the horizon is built from. The candidate and current best algorithms are advanced under common random numbers so their score difference is not dominated by process variance. This is what the search calls in place of a Monte Carlo simulator. The features $\boldsymbol{\eta}$, the network, and the readout depend on the dynamic process alone, so one $\boldsymbol{\theta}$ can be applied across tasks with the same dynamic process. An evaluator $\mathcal{E}$ scores a plan $\boldsymbol{a}$ by rolling it out on n sampled trajectories, under a seed ξ that fixes their randomness, and averaging the per-trajectory scores,

$$\hat{R}_{\mathcal{E}}(\boldsymbol{a};\xi) = \frac{1}{n}\sum_{i=1}^{n} R_i(\boldsymbol{a};\xi), \qquad \hat{R}_{\mathcal{E}}(\pi;\xi) = \hat{R}_{\mathcal{E}}(\pi(\mathcal{T});\xi), \tag{8}$$

where $R_i(\boldsymbol{a};\xi)$ is the task's score read at the horizon from trajectory i, that is, from $\hat{\boldsymbol{s}}_H^{(i)}$ under the world model. An algorithm is scored through the plan it produces, so $\hat{R}_{\mathcal{E}}(\pi;\xi)$ is the evaluator's estimate of $J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi)$, the counterpart under the learned dynamics of the objective $J_P(\pi)$ in Eq. (3). To simplify notation, we write $R_i(\pi;\xi) = R_i(\pi(\mathcal{T});\xi)$.

3.3 Network Algorithm Design Agent

We develop an LLM-based network algorithm design framework around the Network World Model, in which a coding agent searches over executable algorithms for the objective in Eq. (3). Section 3.3.1 describes each round of this search. Section 3.3.2 describes the feedback the world model returns beyond the task score, which tells the agent how its candidate algorithm should be revised. Section 3.3.3 describes how a candidate algorithm is compared with the best under sampling noise.

3.3.1 Algorithm Design Loop

In each round of the algorithm design loop, the coding agent generates an executable candidate algorithm π. The agent is provided with a library of classical algorithms for the task as part of its context (Appendix C), which it may reuse, combine, or extend when generating the candidate. The candidate produces the actions $\boldsymbol{a}$ for the task instance, and the Network World Model rolls these actions forward over n sampled trajectories, scoring the candidate and the current best algorithm $\hat{\pi}$ on the same trajectories under $\hat{R}_{\mathcal{E}}$. The candidate replaces $\hat{\pi}$ when its gain exceeds the variability of the rollout estimates, and the algorithm returned at the end of the search is the estimate of $\pi^\star$ in Eq. (3). Together with the task score, the Network World Model returns the feedback described in Section 3.3.2, which is provided to the agent before the next candidate is generated.

Candidate generation is governed by a set of search operators divided between exploration and exploitation. Exploration is favored earlier in the search, while later iterations increasingly refine the current best algorithm using the evidence accumulated during previous evaluations. The full operator definitions and scheduling rules are provided in Appendix B.3. Each candidate is validated against the task constraints before evaluation.

3.3.2 Rollout Feedback from the Network World Model

The task score states how well a candidate performs but not how to revise it, and a revision to a plan can take three forms: retiming its actions, removing or keeping actions in the plan, and adding or swapping in actions outside it. The Network World Model returns one view for each: rollout diagnostics describe how the process evolved under the plan, action-level credit measures what each action contributed, and counterfactual probes evaluate actions the plan did not take. Diagnostics and credit are computed after every evaluation from the rollouts that produce the score, and the agent poses probes before writing the next candidate, targeting the actions the other two single out.

Rollout diagnostics. The coding agent is given a trace of how the actions produced by the candidate algorithm unfold over the sampled trajectories. For spreading tasks, these diagnostics include the estimated task score and its standard error, the cumulative number of nodes reached after each step, the number reached during that step, and the step at which the process stops spreading. Task-specific diagnostics are used for the inverse and forecasting tasks (Appendix E.1). These diagnostics come from the same rollouts used to evaluate the actions produced by the candidate algorithm and

therefore incur no additional rollout cost. They provide intermediate information about how the network state evolves under those actions, as revealed by the rollout trajectories.

Action-level credit. Credit attributes the task score to the individual actions of the plan. Let $a_1, \ldots, a_k$ denote the k attributable actions contained in the plan $\boldsymbol{a}$ produced by the candidate algorithm, and let $\boldsymbol{a}_{-j}$ denote the same plan with action a_j removed. The credit assigned to a_j is the change in the evaluator score when that action is omitted under the same rollout seed ξ,

$$c_j = \hat{R}_{\mathcal{E}}(\boldsymbol{a}; \xi) - \hat{R}_{\mathcal{E}}(\boldsymbol{a}_{-j}; \xi). \tag{9}$$

A larger c_j indicates a greater marginal contribution of a_j to the evaluated outcome, whereas a value near zero indicates that removing the action has little effect on the task score. Computing credit for all k actions requires $k + 1$ plan evaluations, which are batched through the Network World Model over the same H-step rollout horizon.

Counterfactual probes. Probes evaluate actions outside the current plan. Before generating the next candidate algorithm, the coding agent is given a probe turn in which it can query the Network World Model about counterfactual modifications to the current action set. Each query is evaluated through a world model rollout. For example, one of the probes uses gradients from a mean-field rollout to identify unselected actions with high marginal potential and selected actions with low marginal value. More generally, the probes evaluate alternative actions and interrogate how those alternatives change the resulting network dynamics. The full set of probe operations is provided in Table 4 of Appendix B.1. Together, these feedback signals inform the generation of the next candidate, whose performance is then compared with the current best algorithm.

3.3.3 Search and Candidate Selection

Candidate algorithm selection. The task scores of the candidate algorithm π and the current best algorithm $\hat{\pi}$ are estimated from sampled trajectories and are therefore subject to rollout variability. To reduce the effect of sampling noise on the comparison, a fresh seed ξ is drawn in each round, and both algorithms are evaluated on the same n sampled trajectories. Using shared rollout realizations makes the paired score difference reflect differences between the two algorithms, with variation from independently sampled trajectories removed from the comparison. For trajectory i, the paired difference, the mean improvement, and its standard error are

$$\Delta_i = R_i(\pi; \xi) - R_i(\hat{\pi}; \xi), \qquad \Delta = \frac{1}{n}\sum_{i=1}^{n} \Delta_i, \qquad b = \frac{\mathrm{sd}(\Delta_1, \ldots, \Delta_n)}{\sqrt{n}}. \tag{10}$$

The candidate replaces $\hat{\pi}$ when $\Delta > b$ and is otherwise rejected, so only improvements larger than the estimated variability of the paired rollout differences are accepted. The current best algorithm is re-evaluated under the fresh seed in every round, preventing its score from being carried forward from a favorable rollout realization.

3.4 Reliability of World-Model-Based Algorithm Selection

The Network World Model approximates the transition kernel P, so errors in the learned kernel $\widehat{P}_{\boldsymbol{\theta}}$ may affect which algorithm is selected. We quantify this through algorithm selection regret. Let Π be the set of algorithms considered by the search, with expected scores $J_P(\pi)$ and $J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi)$ under the trusted and learned dynamics. For $\pi^\star \in \arg\max_{\pi \in \Pi} J_P(\pi)$ and the returned algorithm $\widehat{\pi}$,

$$\mathrm{Reg}_P(\widehat{\pi}; \Pi) = J_P(\pi^\star) - J_P(\widehat{\pi}). \tag{11}$$

Because the search runs for finitely many rounds, let $\eta_{\mathrm{search}} \geq 0$ denote its learned objective suboptimality, such that $J_{\widehat{P}_{\boldsymbol{\theta}}}(\widehat{\pi}) \geq \sup_{\pi \in \Pi} J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi) - \eta_{\mathrm{search}}$.

Theorem 1 (World-model error implies bounded algorithm-selection regret)**.** *Suppose the one-step transition error of the Network World Model is at most ϵ over the state-action support reachable by algorithms in Π within horizon H, the task score range is bounded by B_R, and the returned algorithm is η_{search} suboptimal under the learned evaluator. Then*

$$\mathrm{Reg}_P(\widehat{\pi}; \Pi) \leq 2 B_R H \epsilon + \eta_{\mathrm{search}}. \tag{12}$$

Table 1: Performance of designed algorithms on four tasks and two datasets per task. Each cell reports results under two diffusion models: IC / LT, except for influence blocking, which reports IC / CLT. A dash indicates an OOM error or timeout.

Influence maximization: spread (% of nodes activated) (↑)

	Network Science (1,589 nodes)				Digg (116,893 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%
IMM	8.7 / 10.9	24.8 / 30.2	38.0 / 45.1	59.2 / 68.5	25.9 / 47.0	36.0 / 60.9	44.2 / 69.7	55.7 / 79.6
OPIM	8.8 / 10.8	24.2 / 29.5	37.7 / 45.3	58.3 / 67.4	27.4 / 50.5	41.6 / 68.4	51.8 / 77.6	62.7 / 85.5
SubSIM	8.7 / 10.7	24.3 / 29.1	37.3 / 45.0	57.8 / 68.5	27.3 / 50.1	41.7 / 68.4	51.8 / 77.7	62.7 / 85.4
DeepIM	4.8 / 5.3	15.3 / 20.0	27.7 / 32.9	45.8 / 53.0	– / –	– / –	– / –	– / –
DegreeDiscount	8.3 / 10.7	23.7 / 29.7	36.3 / 44.5	55.0 / 65.5	26.0 / 49.6	37.0 / 62.7	45.8 / 70.0	60.0 / 81.6
Network World Model	**8.9 / 11.3**	**25.3 / 30.9**	**39.1 / 46.8**	**60.1 / 69.3**	**30.4 / 56.8**	**45.8 / 73.2**	**56.5 / 83.1**	**69.4 / 93.5**

Adaptive influence maximization: spread (% of nodes activated) (↑)

	Network Science (1,589 nodes)				Digg (116,893 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%
EPIC	8.4 / 10.8	23.1 / 28.7	34.6 / 41.5	52.9 / 62.3	27.5 / 48.4	40.0 / 65.8	49.0 / 74.0	60.7 / 82.4
Adaptive DegreeDiscount	8.2 / 10.8	23.6 / 30.2	36.5 / 45.8	56.9 / 69.3	29.0 / 54.2	43.5 / 68.9	54.8 / 79.4	69.3 / 93.6
IMM	8.7 / 10.9	24.8 / 30.2	38.0 / 45.1	59.2 / 68.5	25.9 / 47.0	36.0 / 60.9	44.2 / 69.7	55.7 / 79.6
Static-Split	8.3 / 10.7	24.1 / 30.3	37.5 / 45.9	58.1 / 69.3	26.4 / 49.6	38.7 / 64.5	49.0 / 73.8	65.1 / 87.5
Network World Model	**8.9 / 11.3**	**25.2 / 30.9**	**39.1 / 46.8**	**60.0 / 69.4**	**30.4 / 56.8**	**45.6 / 73.2**	**56.4 / 83.0**	**71.9 / 94.0**

Critical node detection: remaining spread (% of nodes infected) (↓)

	Power Grid (4,941 nodes)				PGP (10,680 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%
HDA	26.5 / 28.9	24.0 / 27.1	21.8 / 25.4	18.6 / 22.3	26.4 / 31.9	23.3 / 28.5	20.8 / 25.6	18.0 / 21.6
BPD+R	26.7 / 29.1	24.1 / 27.5	21.9 / 25.8	18.5 / 22.4	26.4 / 32.0	– / –	21.2 / 25.8	18.0 / 21.6
CI+R	26.7 / 29.0	24.1 / 27.3	21.9 / 25.7	18.5 / 22.3	26.6 / 32.0	23.4 / 28.5	21.2 / 25.9	18.1 / 21.6
EI	26.8 / 29.2	24.5 / 27.9	22.1 / 26.0	19.1 / 23.2	26.5 / 32.0	23.5 / 28.8	21.1 / 25.9	18.2 / 22.2
Frontier	26.3 / 28.6	22.7 / 25.0	18.7 / 20.4	10.7 / **10.7**	26.7 / 32.0	23.4 / 27.9	20.2 / 23.6	14.2 / 14.5
Network World Model	**26.2 / 28.2**	**22.0 / 24.1**	**17.3 / 19.6**	**10.2 / 10.7**	**25.8 / 31.1**	**21.3 / 26.7**	**16.5 / 21.5**	**11.0 / 13.0**

Influence blocking: rumor cascade size (nodes) (↓)

	Email-EU (1,005 nodes)				Gnutella24 (26,518 nodes)			
Method	10	20	40	50	10	20	40	50
RPS	60.2 / **70.9**	47.4 / 56.7	35.0 / 42.8	31.9 / 39.1	1520.6 / 1625.3	1311.6 / 1380.6	1029.6 / 1076.5	939.8 / 980.4
Reverse	61.0 / 73.1	49.2 / 57.9	37.2 / 44.8	34.5 / 42.1	1817.6 / 1961.8	1713.1 / 1856.4	1462.0 / 1587.7	1413.1 / 1535.7
Proximity	63.1 / 74.9	52.4 / 61.2	39.6 / 47.7	36.3 / 44.2	2338.9 / 2531.6	2289.9 / 2481.1	2196.5 / 2379.0	2141.1 / 2319.7
GreedyReplace	92.2 / 162.3	77.2 / 163.5	60.1 / 157.3	54.8 / 154.9	1551.8 / 1640.5	1364.9 / 1449.2	1088.2 / 1211.8	990.4 / 1072.4
SandIMIN	92.5 / 160.0	77.4 / 158.7	63.7 / 157.8	54.4 / 150.9	1564.9 / 1633.3	1391.2 / 1456.3	1106.4 / 1173.9	1017.8 / 1096.9
Network World Model	**59.4** / 71.7	**46.3 / 55.4**	**34.5 / 42.2**	**31.0 / 38.7**	**1503.3 / 1582.3**	**1275.0 / 1333.8**	**985.6 / 1023.0**	**896.0 / 924.7**

The theorem shows that exact reproduction of every rollout is not required for reliable algorithm selection: the downstream loss is controlled by the accumulated transition error and the search error. The full assumptions and the proof are provided in Appendix A.1. Further consequences for pairwise ranking preservation and exact recovery of the optimal algorithm under a sufficient selection margin are provided in Appendices A.2 and A.3.

4 Experiments

4.1 Setup

Complex network tasks. We evaluate eight tasks spanning three problem families: five intervention tasks, *influence maximization*, *adaptive influence maximization*, *critical node detection*, *influence blocking*, and *epidemic control*; two inverse tasks, *source localization* and *cascade reconstruction*; and one forecasting task, *cascade prediction*. Appendix E.2 provides detailed task definitions.

Evaluation protocol. Each task is evaluated on networks ranging from 10^3 to more than 10^5 nodes (Appendix E.3) under the appropriate diffusion models: IC, LT, CLT, SIR, and SIS. We compare against five task specific baselines, including classical algorithms and neural methods, listed in Appendix E.4. The design loop runs for $G = 10$ rounds with $n = 200$ sampled rollouts per evaluation, horizon $H = 10$, and six probes per round. GPT-5.6 Sol (OpenAI, 2026) is used as the coding agent for the main experiments, with the coding model comparison reported in Figure 2. Final performance of both the designed algorithms and the baselines is evaluated independently using the trusted Monte Carlo simulator with 200 samples (Appendix E.1). Seeding, containment,

and epidemic tasks use budgets of 1%, 5%, 10%, and 20% of N, while influence blocking uses $k \in \{10, 20, 30, 40, 50\}$; recovery and forecasting tasks have no intervention budget. Performance is reported using the task specific metric for each designed algorithm, while the Network World Model is evaluated using one step ΔF1, Brier score, and rollout count bias. Appendix E.2 and Appendix I provide the remaining implementation and experimental details.

4.2 Performance of Designed Algorithms

The designed algorithms are evaluated with the trusted Monte Carlo simulator, independently of the Network World Model used during search, to assess their performance under the true environment dynamics. Across the eight tasks summarized in Figure 1, the designed algorithms improve on the strongest existing method across seven tasks and match it on epidemic control. Table 1 reports representative results, while complete results across all eight tasks and evaluated datasets are provided in Appendix H. Across the 64 configurations in Table 1, the designed algorithms beat or tie the strongest listed baseline in 63, except in one influence-blocking setting. The gains are observed under both IC and LT dynamics and hold across intervention budgets (Figure 5, Appendix H). We additionally compare the framework with four published LLM algorithm discovery systems using the same coding model. The proposed framework performs best at every evaluated budget on both influence maximization and critical node detection; full results are reported in Table 12 of Appendix G.

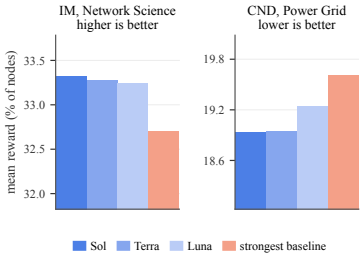


Figure 2: Task score of the returned algorithm under three coding models, averaged over budgets.

Choice of coding model. We show that the gains do not depend on which LLM writes the code. Figure 2 repeats the search on Network Science and Power Grid under three models, GPT-5.6 Sol, Terra, and Luna. Averaged over the four budgets, the three finish within 0.1 points of one another on Network Science and within 0.3 points on Power Grid, and each beats the strongest baseline. The similar performance across coding models suggests that the gains are not specific to the choice of coding model. We provide the full numbers across all three models in Appendix F.

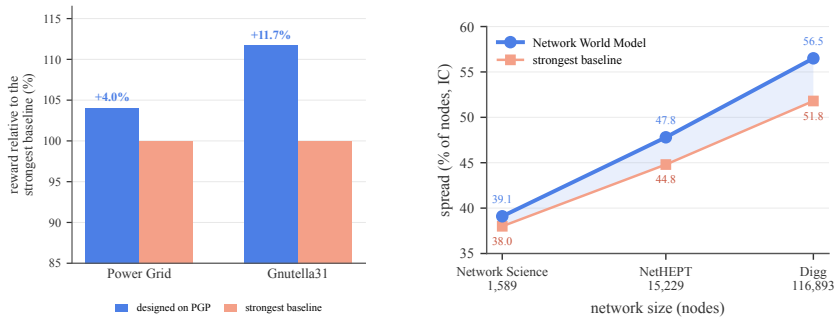
4.3 Cross-Network Generalization and Scaling

We test whether an algorithm designed on one network transfers to other networks of the same task without modification. Figure 3a replays the critical node detection algorithm designed on PGP on Power Grid and Gnutella31, each against the strongest baseline of the target network. The transferred algorithms beat the strongest baseline on every target by 1.8 to 5.1% for influence maximization and 4.0 to 11.7% for critical node detection. Figure 3b complements this transfer result by showing influence maximization performance across increasingly large networks, where the designed algorithm remains ahead of the strongest baseline from Network Science through Digg, with the improvement reaching 7 to 12% on Digg compared with 1 to 4% on Network Science. Across the completed transfers, performance remains within 0.1 percentage points of algorithms designed directly on the target network at every budget (Table 14, Appendix H.1).

4.4 Network World Model Accuracy and Efficiency

We evaluate whether the Network World Model provides the three properties algorithm design requires: accurate rollouts, reliable candidate selection, and efficient evaluation. Table 2 reports prediction quality for five diffusion models. One-step ΔF1 ranges from 0.87 to 0.99 for IC, LT, and CLT and from 0.85 to 0.86 for SIR and SIS, with Brier scores ≤ 0.0006. The model thus accurately identifies which nodes change state with well-calibrated transition probabilities. Free-running rollouts remain within 2.2% of the trusted simulator at the horizon except under LT, where the model undercounts by 6.0%. Rollout bias also remains stable over the horizon (Appendix I).

For algorithm search, the core requirement is preserving the ordering of candidate algorithms. Table 3 compares the same candidate pools scored by the Network World Model and the trusted simulator using preference accuracy and selection regret. On held-out SBM and BA networks, the



(a) Critical node detection, designed on PGP

(b) Influence maximization across network sizes

Figure 3: Generalization across networks and scales. (a) Critical node detection transfer from PGP to Power Grid and Gnutella31, sign-corrected so higher is better. (b) Influence maximization performance across increasing network sizes under IC at the 10% budget.

Table 2: Prediction and rollout quality of the Network World Model across five diffusion models, on Network Science (IC, LT), Email-EU (CLT) and Oregon1 (SIR, SIS).

Dynamics	ΔF1 (↑)	Brier (↓)	Bias (%)
IC	0.8738	0.0003	+0.1
LT	0.9926	0.0003	−6.0
CLT	0.9880	0.0001	−2.2
SIR	0.8565	0.0006	+0.5
SIS	0.8461	0.0006	−0.1

Table 3: Decision value on 20 held-out graphs per family. Regret is the percent difference in reward between the algorithms chosen by the world model and by Monte Carlo simulation.

Graph family	Pref. accuracy [95% CI]	WM regret	Random regret
SBM (in dist.)	0.902 [0.890, 0.914]	0.11	10.27
BA (in dist.)	0.861 [0.818, 0.902]	0.13	12.03
WS (shift)	0.860 [0.832, 0.887]	1.18	10.07

model agrees with the trusted simulator on 90.2% and 86.1% of candidate pair comparisons, with selection regret of 0.11% and 0.13%. On Watts-Strogatz networks, used as a topology shift test, preference accuracy remains 86.0% and selection regret is 1.18%. The resulting low regret shows that the model can reliably support candidate selection even when pairwise agreement is not perfect.

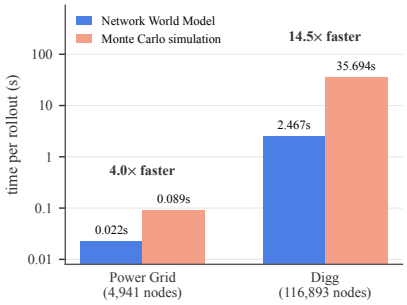


Figure 4: Wall-clock time of one rollout sample under the Network World Model and Monte Carlo simulation, log scale.

The practical utility of the Network World Model is the reduction in repeated evaluation cost. Figure 4 compares the time of one rollout against direct simulation: a world model rollout is 4.0 times faster on Power Grid and 14.5 times faster on Digg. The model incurs a one time offline cost of 38,800 simulator episodes to build the training set, whereas direct simulator based search requires 7,800 episodes for each new search, giving a break even point of about five searches. Reusing the learned dynamics therefore reduces both online evaluation time and cumulative trusted simulator use across repeated searches.

5 Conclusion

We introduce a Network World Model as an environment for algorithm design on complex networks, together with an LLM-based search framework that uses model rollouts and targeted feedback to refine executable algorithms. Across eight tasks and five diffusion models, the designed algorithms match or exceed the strongest reported baseline in 118 of 121 settings, transfer across networks with little performance loss, and can be evaluated faster than with Monte Carlo simulation. Overall, the results show that learned network dynamics can support reliable algorithm selection and extend beyond prediction to the iterative design and evaluation of algorithms for complex networks across a range of tasks.

AI USE STATEMENT

We used generative AI tools to edit the writing, including grammar checking and polishing the text for readability. In addition, large language models are part of the proposed method: the coding agent in the algorithm design loop is an LLM (GPT-5.6 Sol, Terra, or Luna) that writes and revises candidate algorithms. In the paper, we have reviewed all AI-assisted work. We read every suggested edit and accepted or rejected it by hand, and no edit changed the paper's technical content, claims, or results. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work does not raise ethical concerns beyond those in the established network diffusion literature it builds on. It involves no human subjects and no new data collection: all networks and cascade corpora are public, all datasets and baseline methods are used under their original terms, and we use their structure and adoption times, never personal attributes or content. All interventions are evaluated in simulation, and none is deployed on a real platform or population.

REPRODUCIBILITY STATEMENT

To support reproducibility, we release the full source code through an anonymized link: `https://anonymous.4open.science/r/NetworkWorldModel`. It includes the diffusion simulators, network loaders, the training and evaluation of the Network World Model, and the entire algorithm design loop. All datasets we evaluate are public, and we describe their sources in the appendix. The appendix specifies the tasks, baselines, evaluators, and experimental protocol; the world model setup; and the design loop, including its operators, probes, and implementation details. It also includes the complete prompts given to the coding agent. The coding agent itself uses closed-source LLMs accessed through an API, which we chose because they are the current state of the art. We provide examples of some of the best-performing designed algorithms with their source code in the appendix.

REFERENCES

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 946–957. SIAM, 2014.

Dirk Brockmann and Dirk Helbing. The hidden geometry of complex, network-driven contagion phenomena. *science*, 342(6164):1337–1342, 2013.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first international conference on machine learning*, 2024.

Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th international conference on World wide web*, pp. 665–674, 2011.

Qi Cao, Huawei Shen, Keting Cen, Wentao Ouyang, and Xueqi Cheng. Deephawkes: Bridging the gap between prediction and understanding of information cascades. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1149–1158, 2017.

Chen Chen, Hanghang Tong, B Aditya Prakash, Charalampos E Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. Node immunization on large graphs: Theory and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):113–126, 2015.

Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 199–208, 2009.

Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1029–1038, 2010a.

Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *2010 IEEE international conference on data mining*, pp. 88–97. IEEE, 2010b.

Zhen Chen, Kai Zhu, and Lei Ying. Detecting multiple information sources in networks under the sir model. *IEEE Transactions on Network Science and Engineering*, 3(1):17–31, 2016.

Justin Cheng, Lada A Adamic, P Alex Dow, Jon Kleinberg, and Jure Leskovec. Can cascades be predicted? *arXiv preprint arXiv:1403.4608*, 2014.

Pau Clusella, Peter Grassberger, Francisco J Pérez-Reche, and Antonio Politi. Immunization and targeted destruction of networks using explosive percolation. *Physical review letters*, 117(20): 208301, 2016.

Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, pp. 629–638, 2014.

Reuven Cohen, Shlomo Havlin, and Daniel Ben-Avraham. Efficient immunization strategies for computer networks and populations. *Physical review letters*, 91(24):247901, 2003.

Cesar Henrique Comin and Luciano da Fontoura Costa. Identifying the starting point of a spreading process in complex networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 84(5):056105, 2011.

Changjun Fan, Li Zeng, Yizhou Sun, and Yang-Yu Liu. Finding key players in complex networks through deep reinforcement learning. *Nature machine intelligence*, 2(6):317–324, 2020.

Tao Feng, Yexin Wu, Guanyu Lin, and Jiaxuan You. Graph world model. *arXiv preprint arXiv:2507.10539*, 2025.

Vincenzo Fioriti and Marta Chinnici. Predicting the sources of an outbreak with a spectral technique. *arXiv preprint arXiv:1211.2333*, 2012.

Qintian Guo, Sibo Wang, Zhewei Wei, and Ming Chen. Influence maximization revisited: Efficient reverse reachable set generation with bound tightened. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, pp. 2167–2181, 2020.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019b.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

Kai Han, Keke Huang, Xiaokui Xiao, Jing Tang, Aixin Sun, and Xueyan Tang. Efficient algorithms for adaptive influence maximization. *Proceedings of the VLDB Endowment*, 11(9):1029–1040, 2018.

Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of the 2012 siam international conference on data mining*, pp. 463–474. SIAM, 2012.

Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical review E*, 65(5):056109, 2002.

Kyomin Jung, Wooram Heo, and Wei Chen. Irie: Scalable and robust influence maximization in social networks. In *2012 IEEE 12th international conference on data mining*, pp. 918–923. IEEE, 2012.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 137–146, 2003.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *European conference on principles of data mining and knowledge discovery*, pp. 259–271. Springer, 2006.

Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Minimizing the spread of contamination by blocking links in a network. In *AAAI*, volume 8, pp. 1175–1180, 2008.

Jihoon Ko, Kyuhan Lee, Kijung Shin, and Noseong Park. Monstor: an inductive approach for estimating and maximizing influence over unseen networks. In *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 204–211. IEEE, 2020.

Chen Ling, Junji Jiang, Junxiang Wang, and Zhao Liang. Source localization of graph diffusion via variational autoencoders for graph inverse problems. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 1010–1020, 2022.

Chen Ling, Junji Jiang, Junxiang Wang, My T Thai, Renhao Xue, James Song, Meikang Qiu, and Liang Zhao. Deep graph representation learning and optimization for influence maximization. In *International conference on machine learning*, pp. 21350–21361. PMLR, 2023.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

Andrey Y Lokhov, Marc Mézard, Hiroki Ohta, and Lenka Zdeborová. Inferring the origin of an epidemic with a dynamic message-passing algorithm. *Physical Review E*, 90(1):012801, 2014.

Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*, 33:20000–20011, 2020.

Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

Zeeshan Memon, Chen Ling, Ruochen Kong, Vishwanath Seshagiri, Andreas Zufle, and Liang Zhao. Deep identification of propagation trees. *arXiv preprint arXiv:2503.00646*, 2025.

Zeeshan Memon, Yiqi Su, Christo Kurisummoottil Thomas, Walid Saad, Liang Zhao, and Naren Ramakrishnan. Toward world models for epidemiology. *arXiv preprint arXiv:2604.09519*, 2026.

Swapnil Mishra, Marian-Andrei Rizoiu, and Lexing Xie. Feature driven and point process approaches for popularity prediction. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pp. 1069–1078, 2016.

Flaviano Morone and Hernán A Makse. Influence maximization in complex networks through optimal percolation. *Nature*, 524(7563):65–68, 2015.

Salomon Mugisha and Hai-Jun Zhou. Identifying optimal targets of network attack by belief propagation. *Physical Review E*, 94(1):012305, 2016.

Hung T Nguyen, My T Thai, and Thang N Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 international conference on management of data*, pp. 695–710, 2016.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. GPT-5.6 Sol model documentation. `https://developers.openai.com/api/docs/models/gpt-5.6-sol`, 2026. Accessed 2026-09-25.

George Panagopoulos, Nikolaos Tziortziotis, Michalis Vazirgiannis, and Fragkiskos Malliaros. Maximizing influence with graph neural networks. In *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*, pp. 237–244, 2023.

Romualdo Pastor-Satorras and Alessandro Vespignani. Immunization of complex networks. *Physical review E*, 65(3):036104, 2002.

Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Reviews of modern physics*, 87(3):925–979, 2015.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

B Aditya Prakash, Jilles Vreeken, and Christos Faloutsos. Spotting culprits in epidemics: How many and which ones? In *2012 IEEE 12th international conference on data mining*, pp. 11–20. IEEE, 2012.

Victor M Preciado, Michael Zargham, Chinwendu Enyioha, Ali Jadbabaie, and George J Pappas. Optimal resource allocation for network protection against spreading processes. *IEEE Transactions on Control of Network Systems*, 1(1):99–108, 2014.

Xiao-Long Ren, Niels Gleinig, Dirk Helbing, and Nino Antulov-Fantulin. Generalized network dismantling. *Proceedings of the national academy of sciences*, 116(14):6554–6559, 2019.

Marian-Andrei Rizoiu, Lexing Xie, Scott Sanner, Manuel Cebrian, Honglin Yu, and Pascal Van Hentenryck. Expecting to be HIP: Hawkes intensity processes for social media popularity. In *Proceedings of the 26th international conference on world wide web*, pp. 735–744, 2017.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Giulio Rossetti, Letizia Milli, and Salvatore Rinzivillo. Ndlib: a python library to model and analyze diffusion processes over complex networks. In *Companion Proceedings of the The Web Conference 2018*, pp. 183–186, 2018.

Polina Rozenshtein, Aristides Gionis, B Aditya Prakash, and Jilles Vreeken. Reconstructing an epidemic over time. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1835–1844, 2016.

Sudip Saha, Abhijin Adiga, B Aditya Prakash, and Anil Kumar S Vullikanti. Approximation algorithms for reducing the spectral radius to control epidemic spread. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pp. 568–576. SIAM, 2015.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. Pmlr, 2020.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Emre Sefer and Carl Kingsford. Diffusion archeology for diffusion progression history reconstruction. *Knowledge and information systems*, 49(2):403–427, 2016.

Devavrat Shah and Tauhid Zaman. Rumors in a network: Who's the culprit? *IEEE Transactions on information theory*, 57(8):5163–5181, 2011.

Jiaxing Shang, Shangbo Zhou, Xin Li, Lianchen Liu, and Hongchun Wu. Cofim: A community-based framework for influence maximization on large-scale networks. *Knowledge-Based Systems*, 117:88–100, 2017.

Huawei Shen, Dashun Wang, Chaoming Song, and Albert-László Barabási. Modeling and predicting popularity dynamics via reinforced poisson processes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014.

Jilong Shi, Qiangpeng Fang, Xiaobin Rui, Jian Zhang, and Zhixiao Wang. Time-critical adversarial influence blocking maximization. *arXiv preprint arXiv:2511.16068*, 2025.

Shashidhar Sundareisan, Jilles Vreeken, and B Aditya Prakash. Hidden hazards: Finding missing nodes in large graph epidemics. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pp. 415–423. SIAM, 2015.

Gabor Szabo and Bernardo A Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, 2010.

Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. Online processing algorithms for influence maximization. In *Proceedings of the 2018 international conference on management of data*, pp. 991–1005, 2018.

Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 75–86, 2014.

Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 1539–1554, 2015.

Guangmo Amo Tong, Weili Wu, Ling Guo, Deying Li, Cong Liu, Bin Liu, and Ding-Zhu Du. An efficient randomized algorithm for rumor blocking in online social networks. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9. IEEE, 2017.

Hanghang Tong, B Aditya Prakash, Charalampos Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. On the vulnerability of large graphs. In *2010 IEEE international conference on data mining*, pp. 1091–1096. IEEE, 2010.

Hanghang Tong, B Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 245–254, 2012.

Piet Van Mieghem, Dragan Stevanović, Fernando Kuipers, Cong Li, Ruud Van De Bovenkamp, Daijie Liu, and Huijuan Wang. Decreasing the spectral radius of a graph by link removals. *Physical Review E*, 84(1):016101, 2011.

Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2024.

Sebastian Wandelt, Xiaoqian Sun, Daozhong Feng, Massimiliano Zanin, and Shlomo Havlin. A comparative analysis of approaches to network-dismantling. *Scientific reports*, 8(1):13513, 2018.

Jinghao Wang, Yanping Wu, Xiaoyang Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. Efficient influence minimization via node blocking. *arXiv preprint arXiv:2405.12871*, 2024.

Junxiang Wang and Liang Zhao. Graphsl: An open-source library for graph source localization approaches and benchmark datasets. *arXiv preprint arXiv:2405.03724*, 2024.

Zheng Wang, Chaokun Wang, Jisheng Pei, and Xiaojun Ye. Multiple source detection without knowing the underlying propagation model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Virality prediction and community structure in social networks. *Scientific reports*, 3(1):2522, 2013.

Peng Wu and Li Pan. Scalable influence blocking maximization in social networks under competitive independent cascade models. *Computer Networks*, 123:38–50, 2017.

Wenwen Xia, Yuchen Li, Jun Wu, and Shenghong Li. Deepis: Susceptibility estimation on social networks. In *Proceedings of the 14th ACM International Conference on web search and data mining*, pp. 761–769, 2021.

Han Xiao, Cigdem Aslay, and Aristides Gionis. Robust cascade reconstruction by Steiner tree sampling. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 637–646. IEEE, 2018a.

Han Xiao, Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. Reconstructing a cascade from temporal observations. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 666–674. SIAM, 2018b.

Jiadong Xie, Fan Zhang, Kai Wang, Jialu Liu, Xuemin Lin, and Wenjie Zhang. Influence minimization via blocking strategies. *INFORMS Journal on Computing*, 37(6):1587–1604, 2025.

Xovee Xu, Fan Zhou, Kunpeng Zhang, Siyuan Liu, and Goce Trajcevski. Casflow: Exploring hierarchical structures and propagation uncertainty for cascade prediction. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3484–3499, 2021.

Xin Yan, Hui Fang, and Qiang He. Diffusion model for graph inverse problems: Towards effective source localization on complex networks. *Advances in Neural Information Processing Systems*, 36:22326–22350, 2023.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Lenka Zdeborová, Pan Zhang, and Hai-Jun Zhou. Fast and simple decycling and dismantling of networks. *Scientific reports*, 6(1):37954, 2016.

Jian-Xiong Zhang, Duan-Bing Chen, Qiang Dong, and Zhi-Dan Zhao. Identifying a set of influential spreaders in complex networks. *Scientific reports*, 6(1):27823, 2016.

Yao Zhang and B Aditya Prakash. Data-aware vaccine allocation over large networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(2):1–32, 2015.

Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. SEISMIC: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21st ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1513–1522, 2015.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. *arXiv preprint arXiv:2501.08603*, 2025.

Kai Zhu and Lei Ying. Information source detection in the sir model: A sample-path-based approach. *IEEE/ACM Transactions on Networking*, 24(1):408–421, 2014.

Kai Zhu, Zhen Chen, and Lei Ying. Catch'em all: Locating multiple diffusion sources in networks with partial observations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Bo Zong, Yinghui Wu, Ambuj K Singh, and Xifeng Yan. Inferring the underlying structure of information cascades. In *2012 IEEE 12th International Conference on Data Mining*, pp. 1218–1223. IEEE, 2012.

A Theoretical Analysis

A.1 Proof of Theorem 1: Algorithm-Selection Regret

Let $P(\cdot \mid \boldsymbol{s}, a)$ denote the trusted transition kernel in Eq. (1), and let $\widehat{P}_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}, a)$ denote the transition kernel induced by the Network World Model in Eq. (4). For an executable algorithm π, let $J_P(\pi)$ and $J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi)$ denote its expected reward under the trusted and learned dynamics, respectively. Let Π be the executable algorithm class considered by the search, and let $\pi^\star \in \arg\max_{\pi\in\Pi} J_P(\pi)$ be the best algorithm in this class under the trusted dynamics. The algorithm-selection regret $\mathrm{Reg}_P(\widehat{\pi};\Pi)$ of the returned algorithm is defined in Eq. (11). The Network World Model is trained on one-step transitions through Eq. (2) to approximate the conditional transition law of the trusted dynamics. Its approximation quality is therefore characterized by the one-step transition error over the state-action support reachable during algorithm search.

Assumption 1 (Uniform transition accuracy on the search support)**.** *Let $\mathcal{Z}_{\Pi,H}$ be the set of state-action pairs reachable within horizon H by algorithms in Π. There exists $\epsilon \in [0,1]$ such that*

$$\sup_{(\boldsymbol{s},a)\in\mathcal{Z}_{\Pi,H}} D_{\mathrm{TV}}\left(P(\cdot \mid \boldsymbol{s}, a), \widehat{P}_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}, a)\right) \le \epsilon. \tag{13}$$

Assumption 2 (Bounded task reward)**.** *The terminal reward has bounded range over all feasible trajectories:* $\sup_\tau R(\tau) - \inf_\tau R(\tau) \le B_R < \infty$.

Assumption 3 (Approximate world-model search optimality)**.** *Let $\widehat{\pi}$ be the algorithm returned by the coding-agent search using the Network World Model evaluator, and let $\eta_{\mathrm{search}} \ge 0$ be its suboptimality under the learned evaluator as defined in Section 3.4, so that*

$$J_{\widehat{P}_{\boldsymbol{\theta}}}(\widehat{\pi}) \ge \sup_{\pi\in\Pi} J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi) - \eta_{\mathrm{search}}. \tag{14}$$

Theorem 1 is the statement of Eq. (12) under Assumptions 1 to 3.

Proof. Fix any $\pi \in \Pi$. By Assumption 1 and maximal coupling, the trusted and learned trajectories remain coupled at each transition with probability at least $1-\epsilon$. Therefore,

$$\begin{aligned} D_{\mathrm{TV}}\left(\mathbb{P}^\pi_P, \mathbb{P}^\pi_{\widehat{P}_{\boldsymbol{\theta}}}\right) &\le 1 - (1-\epsilon)^H \\ &\le H\epsilon. \end{aligned} \tag{15}$$

By Assumption 2 and Eq. (15),

$$\begin{aligned} \left|J_P(\pi) - J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi)\right| &\le B_R D_{\mathrm{TV}}\left(\mathbb{P}^\pi_P, \mathbb{P}^\pi_{\widehat{P}_{\boldsymbol{\theta}}}\right) \\ &\le B_R\left(1-(1-\epsilon)^H\right) \\ &\le B_R H\epsilon. \end{aligned} \tag{16}$$

Applying Eq. (16) to $\pi^\star$ and $\widehat{\pi}$ and using Assumption 3,

$$\begin{aligned} J_P(\pi^\star) &\le J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi^\star) + B_R\left(1-(1-\epsilon)^H\right) \\ &\le \sup_{\pi\in\Pi} J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi) + B_R\left(1-(1-\epsilon)^H\right) \\ &\le J_{\widehat{P}_{\boldsymbol{\theta}}}(\widehat{\pi}) + \eta_{\mathrm{search}} + B_R\left(1-(1-\epsilon)^H\right) \\ &\le J_P(\widehat{\pi}) + 2B_R\left(1-(1-\epsilon)^H\right) + \eta_{\mathrm{search}}. \end{aligned} \tag{17}$$

Combining Eq. (17) with the regret definition in Eq. (11),

$$\begin{aligned} \mathrm{Reg}_P(\widehat{\pi};\Pi) &\le 2B_R\left(1-(1-\epsilon)^H\right) + \eta_{\mathrm{search}} \\ &\le 2B_R H\epsilon + \eta_{\mathrm{search}}, \end{aligned} \tag{18}$$

which proves Eq. (12). □

A.2 Pairwise Ranking Preservation

Write $\Delta_H(\epsilon) = B_R\left(1-(1-\epsilon)^H\right)$ for the worst-case distortion the Network World Model induces in the value of an algorithm over horizon H, which is the bound established in Eq. (16).

Corollary 1 (Pairwise ranking preservation)**.** *Under Assumptions 1 and 2, let $\pi_i, \pi_j \in \Pi$. If*

$$J_P(\pi_i) - J_P(\pi_j) > 2\Delta_H(\epsilon), \tag{19}$$

then $J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi_i) > J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi_j)$.

Algorithms separated by more than the world-model distortion therefore preserve their true ordering under the learned evaluator. As training reduces ϵ and hence $\Delta_H(\epsilon)$, the guarantee extends to increasingly close algorithm pairs.

Proof. Applying Eq. (16) to π_i and π_j gives

$$\begin{aligned} J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi_i) - J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi_j) &\geq J_P(\pi_i) - \Delta_H(\epsilon) - J_P(\pi_j) - \Delta_H(\epsilon) \\ &= J_P(\pi_i) - J_P(\pi_j) - 2\Delta_H(\epsilon) \\ &> 0, \end{aligned} \tag{20}$$

where the last inequality follows from the margin condition in Eq. (19). Hence $J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi_i) > J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi_j)$, proving Corollary 1. □

A.3 Exact Recovery Under a Selection Margin

Corollary 2 (Exact recovery under a selection margin)**.** *Suppose Π is finite and $\pi^\star$ is its unique best algorithm under the trusted dynamics. Define its selection margin as*

$$\Gamma = J_P(\pi^\star) - \max_{\pi \in \Pi \setminus \{\pi^\star\}} J_P(\pi). \tag{21}$$

If $\Gamma > 2\Delta_H(\epsilon)$ and $\eta_{\text{search}} = 0$, then

$$\arg\max_{\pi \in \Pi} J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi) = \pi^\star. \tag{22}$$

This strengthens ranking preservation to exact algorithm recovery: when the best algorithm is separated from all alternatives by more than the world-model distortion and the learned objective is optimized exactly, the Network World Model selects the same optimum as the trusted dynamics.

Proof. For every $\pi \in \Pi \setminus \{\pi^\star\}$, the definition of the selection margin in Eq. (21) gives

$$\begin{aligned} J_P(\pi^\star) - J_P(\pi) &\geq \Gamma \\ &> 2\Delta_H(\epsilon). \end{aligned} \tag{23}$$

By Corollary 1, Eq. (23) implies

$$J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi^\star) > J_{\widehat{P}_{\boldsymbol{\theta}}}(\pi) \qquad \text{for every } \pi \in \Pi \setminus \{\pi^\star\}. \tag{24}$$

Therefore, $\pi^\star$ is the unique maximizer of the learned objective, and Eq. (22) follows. □

A.4 Exact Intervention Factorization

Theorem 1 shows that algorithm-selection reliability depends on transition accuracy. Transition error is therefore separated into endogenous dynamics error and intervention-semantics error. Let $\boldsymbol{x} = (\mathcal{G}, \boldsymbol{s})$ and let K denote the true endogenous dynamics after intervention. The exact intervention factorization is compared with a generic surrogate that approximates the intervention:

$$\widehat{P}^a_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x}) = \widehat{K}_{\boldsymbol{\theta}}\left(\cdot \mid T^a_{\text{exo}}\boldsymbol{x}, \boldsymbol{\eta}(a)\right), \qquad \widetilde{P}^a_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x}) = \widehat{K}_{\boldsymbol{\theta}}\left(\cdot \mid \widetilde{T}^a\boldsymbol{x}, \boldsymbol{\eta}(a)\right). \tag{25}$$

Here, T^a_{exo} is the exact intervention operator in Eq. (4), while $\widetilde{T}^a$ denotes an approximate intervention transformation.

Theorem 2 (Exact intervention removes action-semantics error)**.** *Suppose the learned endogenous dynamics have one-step approximation error at most ϵ_{dyn}. Then the exact intervention factorization in Eq. (25) has rollout error*

$$D_{\mathrm{TV}}\left(\mathbb{P}_P^{\pi}, \mathbb{P}_{\widehat{P}_{\boldsymbol{\theta}}}^{\pi}\right) \leq 1 - (1 - \epsilon_{\mathrm{dyn}})^H. \tag{26}$$

Further assume that the generic surrogate incurs intervention error ϵ_{exo} and that the endogenous dynamics are L_T-Lipschitz with respect to this error. Then

$$D_{\mathrm{TV}}\left(\mathbb{P}_P^{\pi}, \mathbb{P}_{\widetilde{P}_{\boldsymbol{\theta}}}^{\pi}\right) \leq 1 - \left(1 - \epsilon_{\mathrm{dyn}} - L_T\epsilon_{\mathrm{exo}}\right)^H, \tag{27}$$

whenever $\epsilon_{\mathrm{dyn}} + L_T\epsilon_{\mathrm{exo}} \leq 1$.

Because T_{exo} is applied exactly, the learned transition error contains only the endogenous dynamics term ϵ_{dyn}. A surrogate that must additionally infer how an action modifies the network can accumulate the extra term $L_T\epsilon_{\mathrm{exo}}$ before multi-step rollout begins.

Proof. Let K denote the true endogenous transition kernel after intervention. Further assume that the learned endogenous dynamics have one-step error at most ϵ_{dyn}, that K is L_T-Lipschitz over post-intervention network states under a metric $d_{\mathcal{X}}$, and that $d_{\mathcal{X}}(T_{\mathrm{exo}}^a\boldsymbol{x}, \widetilde{T}^a\boldsymbol{x}) \leq \epsilon_{\mathrm{exo}}$.

For the generic surrogate in Eq. (25), the one-step transition error satisfies

$$\begin{aligned} D_{\mathrm{TV}}\left(P^a(\cdot \mid \boldsymbol{x}), \widetilde{P}_{\boldsymbol{\theta}}^a(\cdot \mid \boldsymbol{x})\right) &\leq D_{\mathrm{TV}}\left(K(\cdot \mid T_{\mathrm{exo}}^a\boldsymbol{x}, \boldsymbol{\eta}(a)), K(\cdot \mid \widetilde{T}^a\boldsymbol{x}, \boldsymbol{\eta}(a))\right) \\ &\quad + D_{\mathrm{TV}}\left(K(\cdot \mid \widetilde{T}^a\boldsymbol{x}, \boldsymbol{\eta}(a)), \widehat{K}_{\boldsymbol{\theta}}(\cdot \mid \widetilde{T}^a\boldsymbol{x}, \boldsymbol{\eta}(a))\right) \\ &\leq L_T d_{\mathcal{X}}\left(T_{\mathrm{exo}}^a\boldsymbol{x}, \widetilde{T}^a\boldsymbol{x}\right) + \epsilon_{\mathrm{dyn}} \\ &\leq \epsilon_{\mathrm{dyn}} + L_T\epsilon_{\mathrm{exo}}. \end{aligned} \tag{28}$$

For the exact intervention factorization in Eq. (25),

$$\begin{aligned} D_{\mathrm{TV}}\left(P^a(\cdot \mid \boldsymbol{x}), \widehat{P}_{\boldsymbol{\theta}}^a(\cdot \mid \boldsymbol{x})\right) &= D_{\mathrm{TV}}\left(K(\cdot \mid T_{\mathrm{exo}}^a\boldsymbol{x}, \boldsymbol{\eta}(a)), \widehat{K}_{\boldsymbol{\theta}}(\cdot \mid T_{\mathrm{exo}}^a\boldsymbol{x}, \boldsymbol{\eta}(a))\right) \\ &\leq \epsilon_{\mathrm{dyn}}. \end{aligned} \tag{29}$$

Fix any executable algorithm $\pi \in \Pi$. Conditional on two coupled trajectories agreeing through time t, they have the same history and therefore π selects the same action. Applying maximal coupling at each transition and Eq. (29),

$$\begin{aligned} D_{\mathrm{TV}}\left(\mathbb{P}_P^{\pi}, \mathbb{P}_{\widehat{P}_{\boldsymbol{\theta}}}^{\pi}\right) &\leq 1 - \prod_{t=0}^{H-1}(1 - \epsilon_{\mathrm{dyn}}) \\ &= 1 - (1 - \epsilon_{\mathrm{dyn}})^H, \end{aligned} \tag{30}$$

which proves Eq. (26).

Similarly, applying Eq. (28) and $\epsilon_{\mathrm{dyn}} + L_T\epsilon_{\mathrm{exo}} \leq 1$,

$$\begin{aligned} D_{\mathrm{TV}}\left(\mathbb{P}_P^{\pi}, \mathbb{P}_{\widetilde{P}_{\boldsymbol{\theta}}}^{\pi}\right) &\leq 1 - \prod_{t=0}^{H-1}\left(1 - \epsilon_{\mathrm{dyn}} - L_T\epsilon_{\mathrm{exo}}\right) \\ &= 1 - \left(1 - \epsilon_{\mathrm{dyn}} - L_T\epsilon_{\mathrm{exo}}\right)^H, \end{aligned} \tag{31}$$

which proves Eq. (27). □

A.5 Tighter Selection Regret Under Exact Intervention

Combining Theorem 2 with Theorem 1 reduces the transition-error term entering the selection-regret bound from $\epsilon_{\mathrm{dyn}} + L_T\epsilon_{\mathrm{exo}}$ to ϵ_{dyn}.

Corollary 3 (Tighter selection regret under exact intervention)**.** *Under the conditions of Theorems 1 and 2, the exact intervention factorization satisfies*

$$\mathrm{Reg}_P(\widehat{\pi};\Pi) \le 2B_R\left(1-(1-\epsilon_{\mathrm{dyn}})^H\right)+\eta_{\mathrm{search}}. \tag{32}$$

For the generic surrogate in Eq. (25), the corresponding guarantee is

$$\mathrm{Reg}_P(\widetilde{\pi};\Pi) \le 2B_R\left[1-(1-\epsilon_{\mathrm{dyn}}-L_T\epsilon_{\mathrm{exo}})^H\right]+\eta_{\mathrm{search}}. \tag{33}$$

Exact intervention thus removes the intervention-semantics term before it can accumulate across the rollout horizon, yielding a tighter regret upper bound whenever $L_T\epsilon_{\mathrm{exo}} > 0$ under the nondegenerate conditions of Theorem 2.

Proof. By Theorem 1 and Eqs. (29)–(28),

$$\mathrm{Reg}_P(\widehat{\pi};\Pi) \le 2B_R\left(1-(1-\epsilon_{\mathrm{dyn}})^H\right)+\eta_{\mathrm{search}}. \tag{34}$$

For the generic surrogate,

$$\mathrm{Reg}_P(\widetilde{\pi};\Pi) \le 2B_R\left[1-(1-\epsilon_{\mathrm{dyn}}-L_T\epsilon_{\mathrm{exo}})^H\right]+\eta_{\mathrm{search}}. \tag{35}$$

The difference between the two regret upper bounds satisfies

$$\begin{aligned}
&2B_R\left[1-(1-\epsilon_{\mathrm{dyn}}-L_T\epsilon_{\mathrm{exo}})^H\right]-2B_R\left[1-(1-\epsilon_{\mathrm{dyn}})^H\right]\\
&=2B_R\left[(1-\epsilon_{\mathrm{dyn}})^H-(1-\epsilon_{\mathrm{dyn}}-L_T\epsilon_{\mathrm{exo}})^H\right]\\
&\ge 0.
\end{aligned} \tag{36}$$

The last inequality is strict when $L_T\epsilon_{\mathrm{exo}} > 0$ under the nondegenerate conditions of Theorem 2, which proves the result. □

B Network Algorithm Design Agent: Details

This appendix gives the parts of the design loop that Section 3.3 summarizes: one round in pseudocode, what the agent's prompt contains, the full list of probes, the instructions the agent writes under and how one is chosen, and the checks a candidate passes before it is scored. Throughout, π is the candidate written in the current round and $\hat{\pi}$ the current best algorithm, as in Section 3.3.

Algorithm 1 One round of the design loop. $\hat{\pi}$ is the current best algorithm, π the candidate written in this round, ξ the seed drawn for it, and $f_{\boldsymbol{\theta}}$ the world model. The anchors are the library's strongest classical members scored on the instance (Appendix C), and the best of them is the bar named in every prompt.

Require: task instance; the library of classical algorithms; the world model $f_{\boldsymbol{\theta}}$; the number of rounds G

score the anchors on the instance; the best is the bar to beat
$\hat{\pi} \leftarrow$ none
for each of the G rounds **do**
  choose an instruction (new mechanisms early, edits late; Appendix B.3)
  probe turn: the agent asks up to six "what if" questions about $\hat{\pi}$'s plan; $f_{\boldsymbol{\theta}}$ answers each with a rollout
  write turn: the agent writes π with its mechanism line, given the instruction, the prompt of Table 5, and the probe answers
  validate π (Appendix B.4); on failure, return the error as a repair turn and **continue**
  rollout: draw a fresh seed ξ and score π and $\hat{\pi}$ on the same n sampled trajectories of H steps through $f_{\boldsymbol{\theta}}$
  feedback $\leftarrow$ rollout diagnostics, per-action credit, the realizations π lost on (Section 3.3.2)
  if $\Delta > b$ in Eq. (10) **then**
    $\hat{\pi} \leftarrow \pi$
  end if
end for
return $\hat{\pi}$

B.1 ALL PROBES

Table 4: The probe operations. Every probe is metered as evaluator work, and a turn may ask at most six. `frontier`, `robust(hidden)` and `gradient` exist on the forward-model evaluators, and `gradient` only for seeding.

Probe	Question it answers about the plan
`drop`(v)	The reward with and without v's actions: v's marginal contribution
`swap`(u, v)	The reward if v replaces u
`add`(v)	The marginal gain of one more budgeted action on an unselected v, at $k+1$: where the headroom is
`best_swap`(v)	The best replacement for v among the 200 highest-degree unselected nodes, with its delta and the runner-up: local search handed over as information, for the model to turn into a rule
`overlap`(u, v)	The expected number of nodes the cascades of u and v both reach, $\sigma(u)+\sigma(v)-\sigma(u, v)$: whether two seeds are redundant, which `drop` cannot say
`horizon`(T)	The spread curve runs out to $T \leq 2H$: a plan that wins early and loses late against one that keeps growing
`frontier`(t)	The expected frontier size after step t and the nodes most likely spreading then: where the outbreak is when a cut lands
`region`$(\mathbb{U})$	The probability mass the plan captures inside a node set $\mathbb{U}$
`robust`(σ)	The reward with every p_{uv} multiplied by $\exp(\nu)$ with $\nu \sim \mathcal{N}(0, \sigma^2)$
`robust(hidden)`	The reward with the transmission probabilities hidden from the model: whether the plan exploited numbers it would not have at deployment
`gradient`(m)	The m highest-gradient unselected nodes and the m lowest-gradient selected ones
`resimulate`$(\mathbb{U}, i)$	Source localization: the consistency of an arbitrary candidate source set $\mathbb{U}$ on episode i, with the nodes it over- and under-explains
`transmission` (u, v, t, i)	Cascade reconstruction: the kernel's probability that v activates at $t+1$ given the state the decoded history asserts at t, and whether u is on that frontier

The `gradient` probe asks the forward model which nodes the plan should have chosen. A sampled rollout is not differentiable, so the probe runs a mean-field one: the plan's seeds enter as a relaxed vector $\boldsymbol{x} \in [\epsilon, 1-\epsilon]^N$ on the $t = 0$ action channel, and each step's predicted marginals $(\boldsymbol{m}_t, \boldsymbol{u}_t)$, the probabilities of being infected and of being on the frontier, are fed back as the next step's state instead of being sampled,

$$(\boldsymbol{m}_{t+1}, \boldsymbol{u}_{t+1}) = \sigma\big(f_{\boldsymbol{\theta}}(\mathcal{G}, \boldsymbol{m}_t, \boldsymbol{u}_t, \mathbf{1}_{t=0}\, \boldsymbol{x})\big), \qquad \nabla_{\boldsymbol{x}} \sum_{v \in \mathbb{V}} m_{H,v}. \tag{37}$$

The sum is the expected final spread under this propagation, and one backward pass through the H forward passes gives its derivative *with respect to* every node's seed weight at once. For an unselected node, the derivative is the marginal value of seeding it; for a selected node, the marginal cost of dropping it. The probe returns the m unselected nodes with the largest gradient and the m selected nodes with the smallest, which the leave-one-out probe and credit feedback miss, since they only ablate nodes already in the plan. The seed vector is kept with ϵ inside $[0, 1]$ because the head clamps its probabilities before the logit, which zeroes the gradient of a node sitting exactly at 0 or 1. The answer is a ranking, not an optimum: the mean-field spread is a surrogate for the sampled reward and the derivative is local, so the agent is told where the headroom is and decides what to do with it. It exists only on the forward-model evaluators, since a Monte Carlo simulator has no derivative, and only under the seeding lever, whose action is the one continuous relaxation the head can take.

B.2 AGENT CONTEXT

Table 5 lists what the agent reads. The static part is sent once, in the opening turn, and the dynamic part is rebuilt for the write turn of every round. Six of its items are not described in Section 3.3 and are defined here.

Search memory. A list of design rules, at most fifty words, shown at the top of every prompt. After every scored round whose candidate differs from the current best, one extra call is shown the worse

and the better program of the pair, ordered by the acceptance decision, with their diagnostics, and returns a hint of at most twenty words (what made the better one better, as a design rule) and the memory revised with that hint, keeping only the rules the evidence still supports (Ye et al., 2024).

Attempts table. One row per round: the instruction, the reward, the paired improvement Δ, whether the candidate was accepted, its mechanism line, the reflection hint, and the agent's own forecast beside what happened. The forecast is a structured comment the agent writes under its mechanism line, stating the signed change it expects against the program it edited and its probability that the change clears the margin b; the run records the forecast's sign accuracy and its calibration against the realized Δ, and the table shows the agent its own past forecasts next to the outcomes.

Population table. Every scored candidate, accepted or not, with its reward on its own realization, its paired strength (the least-squares rank of Appendix B.3), the seconds its plan took, its number of code lines, and its mechanism line.

Anchors and the reference comparison. The anchors are the strongest classical members of the library for the task, marked in Table 7. They are scored on the instance under the same world model, budget, horizon, and exogenous context as the candidates, and listed in the opening turn with their rewards and standard errors, best first, as the bar to beat. The best anchor's plan and rollout trace are kept, and the current best's diagnostics include a comparison against them: which actions the two plans share, where their traces diverge, and by how much.

Counterexample realizations. When a candidate loses the paired comparison, the three sampled trajectories on which it lost by the most are described and appended to its diagnostics, so that they reach the agent as feedback and travel with the record whenever the candidate is later shown as a parent or an inspiration. The description gives the candidate's and the current best's count on that trajectory and, under seeding, the nodes the current best reached that the candidate did not and the candidate's seeds that produced no new infection there; under containment, the nodes the outbreak reached on the candidate's watch that the current best kept clean. On the inverse and forecast tasks, the counterexample is an instance, with both scores and the residual diagnostics. Acceptance still reads the next round's fresh realization, so a counterexample informs the next edit and never decides it.

Last verdict and repair. The paired verdict on the agent's previous edit, so that a candidate that was not accepted still receives one, and the error text if the candidate failed validation, which is what the repair turn of Appendix B.4 answers.

Table 5: The prompt of the write turn. The static part is sent once in the opening turn and the dynamic part is rebuilt every round.

Component	Content
Static (the opening turn)	
task statement	family, objective and its direction, the diffusion model, k (absolute and as a share of N), H, the allowed operations, the exogenous context (outbreak or rumor with its size, observation block, rounds of an adaptive task)
network profile	N, arc count, directedness, density, degree quantiles, k-core size, reciprocity, clustering, assortativity, communities and their sizes; no adjacency
contract and rules	the entry point and return shape, the two structured comment lines (mechanism and forecast), the import whitelist, the budget and validity rules, two worked exemplars at the level the search should start from
library	the callable API of the library without its simulation-based members, with the source of the classical members, and the withheld members named with the reason
anchor leaderboard	every anchor with its reward and standard error on this instance, the best first, "beating the top row is the bar"
probe contract	the probe operations of Table 4 and their costs
Dynamic (every round)	
search memory	the design rules the evidence so far supports, at most fifty words, revised after every scored round
attempts table	one row per round: instruction, reward, Δ, accepted, mechanism, reflection hint, and the agent's own forecast beside what happened
population table	every member with its reward on its own realization, its paired strength, plan seconds, code lines, and mechanism
instruction and programs	the instruction for this round; the current best $\hat{\pi}$ with its full diagnostics; the partner, the two strongest alternatives, or the chosen idea as the instruction needs
diagnostics of $\hat{\pi}$	rollout diagnostics, paired verdict, the reference comparison against the best anchor, per-action credit, counterexample realizations (Section 3.3.2)
last verdict and repair	the paired verdict on the agent's previous edit, and the error text if it failed
probe answers	the probe turn's answers about $\hat{\pi}$, and any answers to questions asked beside the previous code block

B.3 Operators

Each round the agent writes under one of the six instructions in Table 6. Three of them explore, asking for a mechanism the population does not have, and three exploit, asking for an edit of the current best algorithm. The instruction text the agent receives for each is in Appendix D.

Table 6: The six instructions. The two that need a partner are withheld while the population has one member; the instruction text for each is in Appendix D.

Operator	Category	Purpose
`crossover`	Explore	Combine parts from a parent algorithm's mechanism with a partner algorithm's
`synthesize`	Explore	Write one new strategy taking the best-supported idea from each prior algorithm
`from_scratch`	Explore	Implement a completely novel idea, chosen by the idea search below, as a new strategy (must not re-implement a library algorithm or something already in the population)
`refine`	Exploit	One small targeted change: adjust one term or fix one weakness the diagnostics expose
`parameters`	Exploit	Only change numerical constants; keep the same functions and control flow
`simplify`	Exploit	Remove parts of the algorithm that the diagnostics do not justify; a shorter child algorithm is the goal

Exploration and exploitation schedule. Each round first decides whether to explore or exploit, then which instruction to use, so that exploration dominates early in the search and exploitation later. Let r be the index of the round, G the number of rounds, and stall_r the number of consecutive rounds before r whose candidate was rejected under an exploit instruction or failed validation; the count resets when a candidate is accepted and after any explore move. The exploration weight $\lambda_r \in [0, 1]$ is the probability mass given to the three explore instructions as a group,

$$\lambda_r = \min\Big(1,\ \lambda_0 \frac{G - r}{G} + \beta\,\mathrm{stall}_r\Big), \tag{38}$$

with the starting weight $\lambda_0 = 0.8$ and the mass a stall restores $\beta = 0.15$. Exploration fades as the rounds are spent, and stalling pushes back: two consecutive stalls saturate λ_r at one and force an explore move. Within each group, an instruction o is drawn in proportion to a fixed weight w_o, so the chance of drawing it is proportional to $w_o\lambda_r$ if it explores and to $w_o(1 - \lambda_r)$ if it exploits; `crossover` and `synthesize` are withheld while the population has fewer than two members. The decaying term is the budget-aware exploration constant of Zheng et al. (2025), and the stall term is added to it.

Idea search before writing. Under `from_scratch` and `synthesize`, one extra call precedes the write turn. Its prompt asks for three distinct mechanisms, shows the task's library menu (one line per algorithm, marked as available to every candidate so that re-implementing one is not new), the mechanism line and reward of every population member, the search memory, and the attempts table, and asks for the three ideas, a novelty score from zero to ten for each (ten meaning the idea shares no mechanism with the library or the population), the chosen one, and a one-sentence reason. The chosen idea is spliced into the write turn's prompt, and the write turn implements it.

Parent and partner. The program an exploit instruction edits is always the current best $\hat{\pi}$: never the latest attempt, and never the member with the highest raw reward. The distinction matters because each member's reward was measured on its own round's realization, so a rejected candidate that drew a generous realization can carry a raw reward above the current best's while having lost the paired comparison; ranking by raw reward would then hand `refine` the loser. The rest of the population is ranked by a paired strength: the weighted least-squares solution of the search's own comparisons, one equation $s(\pi) - s(\hat{\pi}) = \Delta$ per scored round with the first program's strength pinned at its reward and each equation weighted by $1/b$, so that a member's rank rests on the differences that were measured on shared trajectories rather than on rewards measured on different ones. Under `crossover`, the partner is drawn from the rest of the population with probability proportional to $1/(\mathrm{rank} + 2)$, the selection rule of Liu et al. (2024), and under `refine`, the two strongest other members are shown as alternatives.

B.4 Implementation Details

For end-to-end coding-agent experiments, we fix the LLM configuration within each comparison and report it with the corresponding run; world-model-only experiments do not require an LLM. All Monte Carlo simulations use $n = 200$ episodes per candidate, and all runs using the oracle and the trained world models use $n = 200$ sampled rollouts. We use a base seed of 42 for all experiments.

Budgets $k \in \{1, 5, 10, 20\}$ percent of N for all seeding, containment, and epidemic tasks; $k \in \{10, 20, 30, 40, 50\}$ for influence blocking, the convention in the literature; and a single $k = 10$ percent point for the recovery and forecast tasks, where k is a property of the instance or unused. The outbreak is 10 percent of N for critical node detection and 1 percent for influence blocking and epidemic control. Each execution of a generated program algorithm is limited to a variable wall-clock time, set to either 900 seconds or 1800 seconds on larger networks. We release our project's full code at `https://anonymous.4open.science/r/NetworkWorldModel`.

Validation. Every candidate is validated before it is scored. Its source must pass an abstract syntax tree check that rejects dynamic execution, file primitives, and imports outside the fixed set of numerical and network libraries named in Appendix C. It is then executed in the sandbox of that appendix, with every call under the wall-clock limit above, and any exception, wrong return type, or timeout becomes a structured error whose text is returned to the agent as a repair turn, one further write turn that must fix the error before the round continues. Once the program returns a plan, the

plan is checked against the feasible set $\mathbb{F}$ of Section 3.1: it may use only the allowed operations, must return node IDs that are in range and exist, must stay within the budget, and may not remove a node of the outbreak.

C Callable Library and Network Primitives

Every generated program runs in a sandbox whose namespace holds `ActionOp`, `State`, the read-only `GraphInfo` (node and arc counts, neighborhoods, degrees, and the transmission probability of every arc), the `Strategy` base class, the network primitives as `primitives`, and its own family's library as a module (`algorithms` for influence maximization, `dismantling_algorithms` for critical node detection, and so on), with the source code of the classical members. The AST check rejects any import beyond `numpy`, `networkx`, `scipy`, and the standard math and container modules. We withhold members that select by running their own Monte Carlo simulation from generated programs, since a simulation inside the program would bypass the metered evaluator; the two simulation primitives stay callable, run the NDlib simulator (Rossetti et al., 2018) inside the program, and count against the per-call wall-clock cap.

Every intervention member takes `(network, budget, diffusion_model)` plus the family's exogenous context as a keyword (`negative_seeds=` for blocking, `outbreak=` for containment and immunization) and returns `budget` node IDs, or `(u, v)` arcs under an edge lever; a per-round adaptive policy takes `(state, network, batch, diffusion_model)` and returns at most `batch` seeds for one round. A localizer takes `(network, observation, budget)` and returns `budget` node IDs; a decoder takes `(network, observation, horizon)` and returns {`node: (activation timestep, parent)`} with `parent = None` for a source; a predictor takes `(network, observation, horizon, fit_examples=None)` and returns the popularity at the horizon, or `None` to decline. Optional keyword arguments are the published method's own knobs at the paper's defaults. Table 7 lists every callable member by the mechanism it belongs to.

Table 7: The callable library by mechanism. Counts are callable members.

Group	Members
Network primitives (`primitives`, 16)	
degree and centrality (6)	`compute_degree`, `compute_out_degree`, `compute_weighted_degree`, `get_top_degree_nodes`, `compute_pagerank`, `compute_centrality`
communities and budget (2)	`detect_communities` (label propagation), `allocate_budget` (largest-remainder quota per community)
path influence (1)	`path_influence_scores` (truncated path products)
reverse influence sampling (3)	`batch_reverse_sample`, `ris_select`, `estimate_sample_size`
live-edge realizations (2)	`sample_live_edge_graph`, `reachable_count`
in-program simulation (2)	`mc_simulate_spread`, `compute_marginal_gain` (NDlib inside the program, under the wall-clock cap)
Influence maximization (`algorithms`, 25)	
centrality rankings (7)	`high_degree`, `weighted_degree`, `pagerank_seeds`, `eigenvector_seeds`, `closeness_seeds`, `betweenness_seeds`, `kshell_seeds`
discounted and iterative rankings (4)	`degree_discount` (Chen et al., 2009), `voterank` (Zhang et al., 2016), `collective_influence` (Morone & Makse, 2015), `irie` (Jung et al., 2012)
path-based influence (3)	`sp1m` (Kimura & Saito, 2006), `mia_pmia` (Chen et al., 2010a) and `ldag` (Chen et al., 2010b)
reverse influence sampling (5)	`ris_basic` (Borgs et al., 2014), `tim` and `imm` (Tang et al., 2014; 2015), `ssa` (Nguyen et al., 2016), `filtered_ris`
sketches and communities (5)	`skim` (Cohen et al., 2014), `community_im`, `cofim` (Shang et al., 2017), `community_ris`, `degree_ris_refine`
floor (1)	`random_seeds`
Adaptive influence maximization (`adaptive_algorithms`, 6; one round per call)	

Group	Members
sampling per round (1)	`adapt_epic` (EPIC, Han et al., 2018)
centralities per round (3)	`adapt_degree`, `adapt_degree_discount`, `adapt_pagerank`, each restricted to the susceptible nodes of the realized state
controls (2)	`static_split` (one static seed set dealt out per round), `adapt_random`
Critical node detection (`dismantling_algorithms`, 24)	
one-pass centralities (4)	`degree_removal`, `pagerank_removal`, `betweenness_removal`, `kshell_removal`
adaptive centralities (4)	`adaptive_degree` (HDA, Holme et al., 2002), `iterative_betweenness` and `approx_iterative_betweenness` (Wandelt et al., 2018), `collective_influence_removal` (Morone & Makse, 2015)
decycling and message passing (3)	`corehd` (Zdeborová et al., 2016), `decycling`, `bpd` (Mugisha & Zhou, 2016)
with the published reinsertion pass (5)	`collective_influence_r`, `corehd_r`, `decycling_r`, `bpd_r`, `gndr`; separate rows because each pair is cited under one name
spectral and partition cuts (4)	`gnd` and `egnd` (Ren et al., 2019), `netshield` (Tong et al., 2010), `articulation_removal`
percolation (1)	`explosive_immunization` (Clusella et al., 2016)
controls (3)	`frontier_removal` (the susceptible ring of the outbreak, highest degree first), `acquaintance_immunization` (Cohen et al., 2003), `random_removal`
Influence blocking (`blocking_algorithms`, 19; node levers return nodes, edge levers return arcs)	
proximity to the rumor (3)	`proximity` (the baseline of He et al., 2012), `multi_hop_proximity`, `imin_lhga` (SandIMIN, Wang et al., 2024)
centralities in the reachable region (3)	`degree_blocking` (reported to fail at this task by He et al., 2012), `pagerank_blocking`, `betweenness_blocking`
sampling-based (4)	`forward_blocking` and `reverse_blocking` (TC-AIBM, Shi et al., 2025), `rps` (Tong et al., 2017), `imin_lsbm` (SandIMIN, Wang et al., 2024)
model-based greedy (4)	`cmia_o` (Wu & Pan, 2017), `cldag` (He et al., 2012), `advanced_greedy` and `greedy_replace` (Xie et al., 2025)
edge levers (4)	`kimura_link_blocking` (Kimura et al., 2008), `out_edge_blocking`, `edge_betweenness_blocking`, `random_edge_blocking`
floor (1)	`random_blocking`
Epidemic control (`immunization_algorithms`, 22; node levers return doses, edge levers return arcs)	
degree and centrality doses (7)	`degree_immunization` (Pastor-Satorras & Vespignani, 2002), `adaptive_degree_immunization`, `acquaintance_immunization` (Cohen et al., 2003), `pagerank_immunization`, `eigenvector_immunization`, `betweenness_immunization`, `kshell_immunization`
spectral (4)	`netshield` and `netshield_plus` (Tong et al., 2010; Chen et al., 2015), `greedy_walk` (Saha et al., 2015), `preciado_allocation` (Preciado et al., 2014)
data-aware (3)	`dava` and `dava_fast` (Zhang & Prakash, 2015), `frontier_immunization`
edge levers (7)	`netmelt` (Tong et al., 2012), `product_degree` and `eigen_score` (Van Mieghem et al., 2011), `greedy_walk_edge`, `edge_betweenness_cut`, `frontier_edge_cut`, `random_edge_cut`
floor (1)	`random_immunization`
Source localization (`localization_algorithms`, 13; a per-node scorer of the same name for each in `localization_scorers`)	
label propagation and spectral (3)	`lpsi` (Wang et al., 2017), `netsleuth` (Prakash et al., 2012), `dynamic_age` (Fioriti & Chinnici, 2012)
distance and centre estimators (4)	`rumor_centrality` (Shah & Zaman, 2011), `jordan_center` (Zhu & Ying, 2014), `ojc` (Zhu et al., 2017), `effective_distance` (Brockmann & Helbing, 2013)
message passing (1)	`dmp_localize` (Lokhov et al., 2014)

Group	Members
infected-subgraph centralities (4)	`infected_degree`, `infected_betweenness`, `infected_closeness`, `infected_eigenvector` (Comin & da Fontoura Costa, 2011)
floor (1)	`random_sources`
Cascade reconstruction (`reconstruction_algorithms`, 16; every decoder shares one parent rule)	
ordered Steiner trees (4)	`delayed_bfs`, `ordered_steiner_closure`, `greedy_ordered` (Xiao et al., 2018b), `steiner_tree`
sampling and random walks (2)	`tree_sampling` (Xiao et al., 2018a), `personalized_pagerank`
consistent trees (3)	`consistent_tree_wpct` and `consistent_tree_wbct` (Zong et al., 2012), `cult` (Rozenshtein et al., 2016)
likelihood-based (3)	`dhrec` (Sefer & Kingsford, 2016), `cri` (Chen et al., 2016), `netfill` (Sundareisan et al., 2015)
controls (4)	`jordan_backward`, `observed_only` (the reports alone), `one_hop`, `random_reconstruction`
Cascade prediction (`prediction_algorithms`, 16; `None` declines and is counted)	
feature-based (4)	`szabo_huberman` (Szabo & Huberman, 2010), `feature_linear`, `feature_gbt` (the features of Cheng et al., 2014), `weng_communities` (Weng et al., 2013)
generative point processes (6)	`seismic` (Zhao et al., 2015), `hawkes` and `hawkes_hybrid` (Mishra et al., 2016), `rpp` (Shen et al., 2014), `hip` (Rizoiu et al., 2017), `branching_factor`
graph-only (3)	`neighborhood_size`, `degree_scaled`, `reachability` (the structural ceiling)
floors (3)	`persistence`, `mean_size` (the geometric mean of training sizes), `random_prediction`

D PROMPTS OF THE DESIGN AGENT

Every prompt is assembled by `coding_agent/prompts.py`; the texts below are the load-bearing parts, verbatim where short and condensed where a block runs to pages (the worked exemplars and the per-family briefs). The gateway drops the system role, so the system text is folded into the first user turn.

System prompt (intervention families). The task brief, then:

```
OUTPUT FORMAT: the first line inside your fenced block MUST
be `# MECHANISM: <one sentence>` naming the idea of the
algorithm; from the second attempt on, the second line is
`# EXPECTED: <signed change you expect against the program
you edited> p=<0-1>`.  Reply with exactly ONE fenced python
block and nothing else, no prose before or after.  The block
contains import lines (if you need any) and then exactly ONE
class subclassing `Strategy`.  Nothing else at module level.
IMPORTS: you MAY import any of numpy, networkx, scipy, math,
random, statistics, heapq, bisect, collections, itertools,
functools.  Use numpy for anything you would otherwise write
as a Python loop over all nodes.  Importing anything else is
rejected.
AVAILABLE NAMES (already in your script's namespace:  do NOT
import these):  `ActionOp(op, target, destination=None,
weight=None)` [...]; `State` [...]; `GraphInfo`:
.num_nodes, .out_neighbors(node), .in_neighbors(node),
.degree(node), .edge_index, .ic_probs; the `algorithms`,
`adaptive_algorithms` and `primitives` modules (API below).
ACTION RULES: A seed is ActionOp("add_node", node).  Emit at
most `budget` add_node actions in total.  Seeding the same
node twice is REJECTED. Node ids must be in [0, num_nodes).
```

```
[...]
METHOD: ONE-SHOT SUPER-ALGORITHM. Implement
`plan_horizon(self, graph, budget, horizon) ->
list[list[ActionOp]]`.  Return a list of length (horizon+1):
element t is the action bag applied at timestep t.  This is
your whole multi-timestep plan, decided up front.
```

followed by two worked exemplars of the family, the timing note for the family (when to spend the budget), and the remove-semantics note when the task may emit removals. Containment replaces the action rules with the removal rules (at most `budget remove_node` actions, the harness expands each into a deletion, removing a source is rejected, `add_node` is rejected); the recovery and forecast families replace the whole block with their own contract text.

Opening user turn. The task line (task, objective and direction), `diffusion_model`, `budget` (absolute and as a percentage of nodes, with its unit), `horizon`, `allowed_ops`; the outbreak, rounds, observation, mask or cascade block the family needs; the graph profile of Table 5; the library API reference with the source of the classical members and the blocked names; the anchor leaderboard (`REFERENCE SCORES: ... run on THIS graph, under THIS evaluator, at the same budget and horizon. Beating the top row is the bar`); the probe contract; and `Write the Strategy now`.

Generation turn.

```
You are evolving a population of strategies.
SEARCH MEMORY (rules the evidence so far supports):  [M_g]
ATTEMPTS SO FAR (every generation of this search):
[attempts table]
POPULATION (what survives):  [population table]
YOUR LAST EDIT: [paired verdict]
PARENT: the best in the population, which is NOT necessarily
your last attempt (reward=..., mechanism:  ...):  [script]
Parent rollout diagnostics:  [feedback bundle]
[PARTNER / ALTERNATIVE / STRATEGY blocks, or CHOSEN IDEA,
per the operator]
OPERATION: [instruction below]
The first line inside your code block must be `# MECHANISM:
<one sentence>` naming the idea of the strategy.  The second
line must be `# EXPECTED: <signed change in reward you
expect against the program you edited, in the task's units>
p=<your probability, 0 to 1, that the change clears the
noise band>`; the realized delta is compared with it and
your calibration is reported back to you.  Reply with one
python block.
[PROBE RESULTS blocks]
```

Operator instructions.

```
REFINE: make a SMALL, targeted improvement to the PARENT:
adjust one term, fix one weakness the diagnostics expose,
keep its overall approach.
PARAMETERS: change ONLY numeric constants of the PARENT
(weights, thresholds, sample counts, radii).  No structural
change:  same functions, same control flow, different
numbers, chosen from what the diagnostics say.
SIMPLIFY: remove components of the PARENT that the
diagnostics do not justify:  a term, a stage, a special
case, a parameter.  The child must be SHORTER and score
within the noise band of the parent; that counts as a
success here, because an algorithm a reader can follow is
the goal.
CROSSOVER: combine the PARENT's mechanism with the PARTNER's
into one strategy that keeps the parent's contract.  The
child must contain an identifiable part of each; it is not a
```

```
re-tune of either.
SYNTHESIZE: read every strategy shown and write ONE new
strategy that takes the best-supported idea from each,
according to their diagnostics.  It must differ from every
strategy shown and from every library algorithm.
FROM SCRATCH: do NOT edit any program shown.  Implement the
CHOSEN IDEA as a new strategy under the same contract.  It
must not be a re-implementation of a library algorithm or of
a mechanism already in the population.
```

Idea search.

```
Before writing code, propose 3 DISTINCT mechanisms for a new
strategy on this task, then choose the one most different
from everything below that you still expect to score well.
LIBRARY ALGORITHMS (already available to everyone;
re-implementing one is not new):  [menu]
MECHANISMS ALREADY IN THE POPULATION: [one line per member
with reward]
SEARCH MEMORY: [M_g]   ATTEMPTS SO FAR: [attempts table]
Reply with JSON only:  {"ideas":  ["<one sentence each>",
...], "novelty":  [<0-10 per idea:  10 = shares no mechanism
with the library or the population>], "chosen":  <index into
ideas>, "why":  "<one sentence>"}
```

Reflection.

```
Two strategies for the same task were scored on the same
realization ([direction], unit:  [unit]).  The second scored
better.
[Worse:  reward, mechanism, script, diagnostics]   [Better:
reward, mechanism, script, diagnostics]
Prior memory of this search (may be empty):  [M_g]
Reply with JSON only:  {"hint":  "<at most 20 words:  what
made the better one better, as a design rule>", "memory":
"<at most 50 words:  the prior memory revised with this
hint; keep only rules the evidence still supports>"}
```

Probe turn.

```
PROBE TURN before your next [operator] edit.  The program
you will edit is the incumbent (iteration [g], reward [...],
mechanism:  [...]).  Its diagnostics:  [feedback bundle]
You may ask up to 6 what-if questions about THAT plan; the
answers arrive in the prompt for this generation.  Every
probe costs evaluator work.  Ops:  [the operations of
Table 4 with their JSON shapes]
Reply with exactly ONE fenced probes block {"probes":
[...]} or the single word NONE if you have nothing to ask.
No code in this reply.
```

The probe contract in the opening turn lists the same operations with their JSON shapes and states that a probes block may also be placed beside a code block, answered with the next generation's feedback, and that every probe is metered like a rollout. Under the native condition, the contract is absent, and a block is answered with: `probes are unavailable under the native condition:  no forward model may be consulted during this search.`

E Experimental Protocol

E.1 Rollout Evaluators

Every evaluator exposes one operation, $\tau = \mathrm{rollout}(\pi, H, k, \xi)$, which executes π's plan $\boldsymbol{a}$ under seed ξ and returns a trajectory τ carrying the per-trajectory scores $R_i(\pi; \xi)$.

Monte Carlo. n independent NDlib episodes (Rossetti et al., 2018) per call (each call's seed is different per episode). Each stops early once the frontier node set and the action bag are both empty. The total cost of the Monte Carlo evaluator grows with n times the number of candidates.

Oracle. The environment whose per-edge transmission is pinned to the truth: $q_{uv} = p_{uv}$ under IC, the closed-form threshold hazard under LT, the simulator's own rates under the compartmental dynamics, and both campaigns' probabilities composed through the tie-break under competition. Nothing is learned in the oracle: it reproduces the NDlib process exactly, in a vectorized implementation. The oracle is the ceiling condition (an agent with perfect dynamics) and the shared referee for all final number evaluations and comparisons.

Network World Model. One rollout advances n sampled states. Each timestep, the algorithm's policy is queried once per active sample, so one forward pass yields activation probabilities. One uniform draw per (sample, node) against the *frontier* channel decides the new activations. A sample deactivates when its frontier node set and action bag are empty, and the call ends when all are inactive; dead samples hold their final count so per-sample curves stay aligned. Its online cost is a batched learned forward pass per timestep.

E.2 TASKS

Table 8: The eight tasks on complex systems, summarized.

Task	Dynamics	Task output	Reported metric
Influence maximization	IC, LT	Seed set	Final spread (↑)
Adaptive influence maximization	IC, LT	Per-round seeding policy	Final spread (↑)
Critical node detection	IC, LT	Node removals against an outbreak	Final infected (↓)
Influence blocking	IC, CLT	Counter-seeds, node or arc blocks, or weight cuts against a rumor	Final rumor size (↓)
Epidemic control	SIR, SIS	Vaccinations, quarantines, arc cuts, or contact reductions	Attack rate (↓)
Source localization	IC, LT	Source set	Consistency with the observation (↑)
Cascade reconstruction	IC, LT	Trajectory with parents	Referee reward of the history (↑)
Cascade prediction	Logged cascades	Popularity at the prediction horizon	MSLE (↓)

Every task shares the transition of Eq. (4) and the evaluators of Appendix E.1; what differs is the contract a program implements, what the harness injects as Ω, and the reward. In every case, the reward is computable without a label, and the label metrics of the recovery and forecast families are computed only after the search, on the winner.

Influence maximization (maximize). π implements `plan_horizon`$(\mathcal{G}, k, H)$ and returns the plan of actions $\boldsymbol{a}$; the budget buys k `add_node` seeds; the reward is the final spread $|\mathbb{I}_H|$.

Adaptive influence maximization (maximize). Same dynamics, same data; the algorithm is a per-round policy `act`$(\boldsymbol{s}_t, \mathcal{G}, t)$ called at each of ρ round boundaries with the state the previous batch's diffusion produced, committing batches that sum to k, with g timesteps of diffusion between rounds and a feedback model (full adoption or myopic) deciding what the policy sees. The registry pairs every adaptive arm with the matching static arm so the adaptivity gap, $\mathrm{spread(adaptive)}/\mathrm{spread(static)}$ at matched k on the referee, is always computable.

Critical node detection (minimize). The harness seeds an outbreak $\Omega = \mathbb{S}$ of 10 percent of N at $t = 0$; the budget buys k `remove_node` deletions, each expanded by the harness into the node and its incident arcs; the reward is the final infected count. Removing a source is rejected (deleting patient zero ends an outbreak rather than containing it), and because the planner is told the outbreak,

the budget must sit below the one-hop ring $|N_1(\mathbb{S})\backslash\mathbb{S}|$, at which every outbreak-aware plan scores exactly $|\mathbb{S}|$.

Influence blocking (minimize). A rumor $\mathbb{S}_N$ is committed at $t = 0$ and already spreading on a two-cascade simulator; the lever decides what the budget buys: counter-seeds (`add_node` seeds the positive cascade), node blocks, arc blocks, or weight cuts. The reward is the rumor's final size, and the report also carries prevented influence $\mathrm{spread}(\mathbb{S}_N, \emptyset) - \mathrm{spread}(\mathbb{S}_N \mid \text{blockers})$. The tie-break between the two cascades, the positive campaign's probabilities, and the detection delay are protocol parameters recorded with every run.

Epidemic control (minimize). Index cases are seeded at $t = 0$ under SIR and SIS on a stepper with a per-arc transmission probability; the lever maps the budget onto vaccination (immune, uncounted), quarantine (isolated, still counted), arc cuts, or graded contact reduction through `set_edge_weight`. The reward is the attack rate, the number of nodes ever infected; the report carries the prevalence curve and the spectral eigendrop as context.

Source localization (recover). π implements `localize`$(\mathcal{G}, \boldsymbol{y}, k)$ and names the k nodes it believes started the cascade whose observed state $\boldsymbol{y} \in [0,1]^N$ it is shown; k is a property of the instance. The reward is label-free consistency; we compute precision, recall, F_1, and AUC against the stored sources only after the search, on both the selection and a disjoint evaluation pool.

Cascade reconstruction (recover). π implements `reconstruct`$(\mathcal{G}, o, H)$ and returns $\{v \mapsto (t_v, \mathrm{parent}_v)\}$ for every node it believes was infected, from a partial observation o of a diffusion that already happened (partial times, partial nodes, final snapshot, or hidden nodes). The reward is the arm's kernel log-likelihood of the asserted history per node minus the observation it contradicts; after the search, we compute path precision, event F_1, and timing error against the stored history.

Cascade prediction (forecast). The only task with no simulator: transitions are replayed from a real corpus, π implements `predict`$(\mathcal{G}, o, t_p)$ and returns the popularity a real cascade will have reached by t_p from its first t_o steps, and may return `None` to decline. The reward minimizes mean squared log error; declines are counted but never scored. The split is chronological by default, because the field's random split leaks.

E.3 DATASETS

Dataset	Nodes	Edges	Tasks
Email-EU	1,005	24,929	Influence Blocking
UCI Students	1,266	6,451	Cascade Reconstruction
Network Science	1,589	2,742	Influence Maximization, Adaptive Influence Maximization
Cora-ML	2,810	7,981	Source Localization
Power Grid	4,941	6,594	Critical Node Detection, Source Localization
CA-GrQc	5,242	14,484	Cascade Reconstruction
Oregon1	10,670	22,002	Epidemic Control
PGP	10,680	24,316	Critical Node Detection
Infectious SocioPatterns	10,972	44,517	Epidemic Control
NetHEPT	15,229	62,752	Influence Maximization, Adaptive Influence Maximization
Gnutella24	26,518	65,369	Influence Blocking
Taoke	29,711	–	Cascade Prediction
Deezer	47,538	222,887	Source Localization
Gnutella31	62,561	147,878	Critical Node Detection
Digg	116,893	2,011,447	Influence Maximization, Adaptive Influence Maximization
Digg (cascades)	279,630	1,731,653	Cascade Prediction
APS	616,316	3,304,400	Cascade Prediction

Table 9: The datasets used across the eight tasks, ordered by node count. For the three cascade-prediction corpora, the counts describe the full underlying network, while the runs replay a subsample restricted to the most active participants.

E.4 BASELINES

Table 10 lists every baseline that we compare against: five per task, and four for adaptive influence maximization. Entries marked library are our implementations of the published algorithms, and entries marked published code are the authors' released implementations, run in their own environment. Every baseline runs on the same network, budget, and rumor or outbreak, and its returned solution is scored on the shared referee. Learned approaches to these tasks that are not run here include graph neural influence maximization (Panagopoulos et al., 2023) and deep cascade popularity models (Cao et al., 2017; Xu et al., 2021).

Table 10: The baseline methods we compare against, what each does, and where it comes from.

Task	Method	What it does	Reference
Influence maximization	IMM	Reverse influence sampling with a martingale stopping rule; the standard RIS reference (library)	(Tang et al., 2015)
	OPIM	Online-processing RIS that tightens its bound as samples arrive (published code)	(Tang et al., 2018)
	SubSIM	Sublinear-time reverse reachable set generation with tightened bounds (published code)	(Guo et al., 2020)
	DeepIM	Learned seed-set generator with a graph encoder and a spread predictor, trained per network (published code)	(Ling et al., 2023)
	DegreeDiscount	Degree ranking with a discount for neighbors already seeded; the anchor (library)	(Chen et al., 2009)
Adaptive influence maximization	EPIC	AdaptGreedy instantiated with reverse influence sampling per round; the anchor (library)	(Han et al., 2018)
	Adaptive DegreeDiscount	DegreeDiscount recomputed each round on the nodes still susceptible (library)	(Chen et al., 2009)
	IMM	The non-adaptive seed set, committed in full at $t = 0$ (library)	(Tang et al., 2015)
	Static-Split	One static seed set dealt out over the rounds; the control for the round machinery	this work
Critical node detection	HDA	Remove the highest-degree node, recompute degrees, repeat (library)	(Holme et al., 2002)
	BPD+R	Belief-propagation-guided decycling with the published reinsertion pass (library)	(Mugisha & Zhou, 2016)
	CI+R	Collective influence, optimal percolation, with the reinsertion pass (library)	(Morone & Makse, 2015)
	EI	Explosive immunization, percolation-based targeted removal (library)	(Clusella et al., 2016)
	Frontier	Delete the susceptible ring of the observed outbreak, highest degree first; the outbreak-aware control	this work
Epidemic control	DAVA	Data-aware vaccine allocation by a dominator tree on the observed infection state (library)	(Zhang & Prakash, 2015)
	NetShield+	Batched spectral-radius reduction by NetShield's shield value (library)	(Chen et al., 2015)
	GreedyWalk	Greedy spectral-radius reduction through closed-walk counts (library)	(Saha et al., 2015)
	EI	Explosive immunization (published code)	(Clusella et al., 2016)
	CI	Collective influence (published code)	(Morone & Makse, 2015)
Influence blocking	RPS	Greedy maximum coverage over reverse prevention sets (library)	(Tong et al., 2017)
	Reverse	Reverse reachable set coverage restricted to the rumor's reachable region, TC-AIBM's baseline (library)	(Shi et al., 2025)

Task	Method	What it does	Reference
	Proximity	Seed the rumor's out-neighbors, highest degree first, CLDAG's baseline; the anchor (library)	(He et al., 2012)
	GreedyReplace	Node blocking by greedy replacement, node lever (published code)	(Xie et al., 2025)
	SandIMIN	Sandwich-bounded node blocking, node lever (published code)	(Wang et al., 2024)
Source localization	LPSI	Label propagation; the sources are the local maxima of the converged field; the anchor (library)	(Wang et al., 2017)
	Rumor Centrality	The founding single-source estimator, applied per infected component (library)	(Shah & Zaman, 2011)
	Dynamic Age	Spectral multi-source estimator from the dynamical age of nodes (library)	(Fioriti & Chinnici, 2012)
	Infected-Degree	Degree within the infected subgraph (library)	(Comin & da Fontoura Costa, 2011)
	SL-VAE	Variational autoencoder over source sets with a learned forward model, run through the GraphSL library (published code)	(Ling et al., 2022; Wang & Zhao, 2024)
Cascade reconstruction	Reports	The observed nodes alone with no inferred hidden ones; precision one by construction (library)	(Rozenshtein et al., 2016)
	Steiner Tree	Minimum Steiner tree over the reported nodes, the control of the ordered variants (library)	(Xiao et al., 2018b)
	Jordan-Backward	The Jordan center of each observed component as its source, then a greedy backward decode (library)	(Zhu & Ying, 2014)
	DHREC	Diffusion archaeology, maximum-likelihood history reconstruction (library)	(Sefer & Kingsford, 2016)
	CRI	Cluster the infected subgraph, then reverse-infect to the sources (library)	(Chen et al., 2016)
Cascade prediction	Persistence	Predicts no growth after the observation window; the floor	this work
	Szabo-Huberman	Log-linear regression from early to final popularity; the anchor (library)	(Szabo & Huberman, 2010)
	Hawkes	A marked Hawkes process fitted per cascade (library)	(Mishra et al., 2016)
	RPP	Reinforced Poisson process with fitness, aging and rich-get-richer terms (library)	(Shen et al., 2014)
	Weng-Communities	Early diffusion across communities as the virality signal (library)	(Weng et al., 2013)

F Choice of Coding Model

We test whether the results depend on the LLM used as the coding agent by repeating the search on Network Science and Power Grid with GPT-5.6 Sol, Terra, and Luna. Table 11 reports the reward of the algorithm returned by the design loop under each model at every budget. The three models converge to similar performance, and each outperforms the strongest classical baseline, indicating that the gains are not specific to a particular coding model.

Table 11: Algorithm design repeated on GPT-5.6 Sol, Terra, and Luna.

	Network Science (1,589 nodes) (↑)				Power Grid (4,941 nodes) (↓)			
Model	1%	5%	10%	20%	1%	5%	10%	20%
GPT-5.6 Sol	8.92	25.27	39.06	60.06	26.18	22.04	17.26	10.24
GPT-5.6 Terra	8.89	25.23	38.98	60.01	26.19	22.05	17.28	10.25
GPT-5.6 Luna	8.81	25.21	38.95	60.00	26.32	22.34	17.82	10.49

G Comparison with LLM Algorithm Discovery Systems

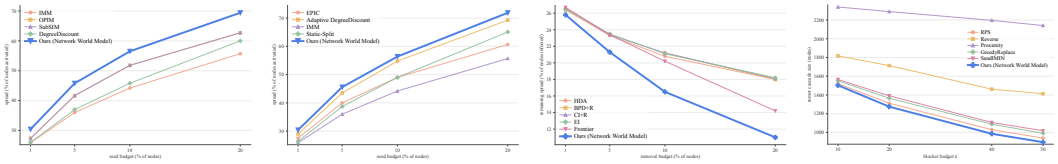
The design loop is one of several systems that use an LLM to search over programs. We compare it with four published algorithm discovery systems: EoH (Liu et al., 2024), OpenEvolve, the open-source implementation of AlphaEvolve (Novikov et al., 2025), LLaMEA (Van Stein & Bäck, 2024), and ReEvo (Ye et al., 2024). Each system runs its own search loop, prompts, and published defaults, and is given the same problem as the design loop: the task statement and a fitness function that scores a candidate on the exact simulator. No system sees the Network World Model. All four and the design loop use the same coding model, GPT-5.6 Sol.

Table 12: Comparison with LLM algorithm discovery systems on influence maximization on Network Science (spread, % of nodes activated, ↑) and critical node detection on Power Grid (remaining spread, % of nodes infected, ↓), under IC. All rows use GPT-5.6 Sol as the coding model.

	Influence maximization (↑) Network Science (1,589 nodes)				Critical node detection (↓) Power Grid (4,941 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%
EoH	5.5	19.5	32.5	54.6	26.6	23.4	19.4	14.3
OpenEvolve	5.1	21.9	36.8	50.8	26.6	23.8	21.4	17.7
LLaMEA	6.1	21.1	34.9	56.2	26.6	23.4	19.5	14.3
ReEvo	6.6	22.3	35.5	57.0	26.6	23.5	19.6	15.0
Network World Model	**8.9**	**25.3**	**39.0**	**60.0**	**26.2**	**22.0**	**17.3**	**10.2**

Table 12 reports the results. For both influence maximization and critical node detection, the design loop performs best at every budget. None of the four discovery systems reaches the strongest classical baseline on Network Science at any budget (IMM or OPIM, 8.8, 24.8, 38.0, and 59.2 in Table 1), although EoH and LLaMEA both converged to reverse-reachable-set sampling, the mechanism behind IMM. Therefore, the contributions of the algorithm design loop and the Network World Model are necessary to achieve considerable gains over the baselines.

H Full Results



(a) IM, Digg (IC) (b) AIM, Digg (IC) (c) CND, PGP (IC) (d) IB, Gnutella24 (IC)

Figure 5: Performance across intervention budgets under IC for influence maximization (IM), adaptive influence maximization (AIM), critical node detection (CND), and influence blocking (IB).

Table 13 reports every end-to-end result behind Section 4.2: all eight tasks and every dataset that was run. The two-dataset view of Table 1 is a subset of it.

Table 13: Designed algorithms results for all eight tasks and every dataset. Every cell is the IC / LT pair (IC / CLT for influence blocking, SIR / SIS for epidemic control) obtained by replaying the returned solution on the shared oracle referee, so all methods are scored on the same referee. A dash in a cell indicates no result because of either an Out of Memory (OOM) error or a timeout that exceeds the 1,800 s wall-clock limit.

Influence maximization (IC / LT): spread, % of nodes activated (↑)

	Network Science (1,589 nodes)				Digg (116,893 nodes)				NetHEPT (15,229 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%	1%	5%	10%	20%
IMM	8.7 / 10.9	24.8 / 30.2	38.0 / 45.1	59.2 / 68.5	25.9 / 47.0	36.0 / 60.9	44.2 / 69.7	55.7 / 79.6	11.6 / 14.8	28.9 / 36.2	42.4 / 51.6	61.7 / 72.5
OPIM	8.8 / 10.8	24.2 / 29.5	37.7 / 45.3	58.3 / 67.4	27.4 / 50.5	41.6 / 68.4	51.8 / 77.6	62.7 / 85.5	12.1 / 15.5	31.4 / 39.2	44.7 / 54.3	64.5 / 75.7
SubSIM	8.7 / 10.7	24.3 / 29.1	37.3 / 45.0	57.8 / 68.5	27.3 / 50.1	41.7 / 68.4	51.8 / 77.7	62.7 / 85.4	12.6 / 16.2	31.3 / 39.0	44.8 / 54.5	64.3 / 75.4
DeepIM	4.8 / 5.3	15.3 / 20.0	27.7 / 32.9	45.8 / 53.0	– / –	– / –	– / –	– / –	– / –	– / –	– / –	– / –
DegreeDiscount	8.3 / 10.7	23.7 / 29.7	36.3 / 44.5	55.0 / 65.5	26.0 / 49.6	37.0 / 62.7	45.8 / 70.0	60.0 / 81.6	11.9 / 15.9	30.3 / 38.8	43.0 / 53.5	61.1 / 73.2
Network World Model	**8.9 / 11.3**	**25.3 / 30.9**	**39.1 / 46.8**	**60.1 / 69.3**	**30.4 / 56.8**	**45.8 / 73.2**	**56.5 / 83.1**	**69.4 / 93.5**	**12.9 / 16.9**	**33.0 / 41.4**	**47.8 / 58.1**	**68.5 / 80.0**

Adaptive influence maximization (IC / LT): spread, % of nodes activated (↑)

	Network Science (1,589 nodes)				Digg (116,893 nodes)				NetHEPT (15,229 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%	1%	5%	10%	20%
EPIC	8.4 / 10.8	23.1 / 28.7	34.6 / 41.5	52.9 / 62.3	27.5 / 48.4	40.0 / 65.8	49.0 / 74.0	60.7 / 82.4	11.4 / 14.6	28.9 / 36.5	41.2 / 51.3	60.4 / 70.6
Adaptive DegreeDiscount	8.2 / 10.8	23.6 / 30.2	36.5 / 45.8	56.9 / 69.3	29.0 / 54.2	43.5 / 68.9	54.8 / 79.4	69.3 / 93.6	11.9 / 16.4	31.6 / 41.3	46.7 / 58.2	67.9 / 80.1
IMM	8.7 / 10.9	24.8 / 30.2	38.0 / 45.1	59.2 / 68.5	25.9 / 47.0	36.0 / 60.9	44.2 / 69.7	55.7 / 79.6	11.6 / 14.8	28.9 / 36.2	42.4 / 51.6	61.7 / 72.5
Static-Split	8.3 / 10.7	24.1 / 30.3	37.5 / 45.9	58.1 / 69.3	26.4 / 49.6	38.7 / 64.5	49.0 / 73.8	65.1 / 87.5	12.1 / 16.0	31.0 / 39.8	44.7 / 55.8	63.8 / 76.8
Network World Model	**8.9 / 11.3**	**25.2 / 30.9**	**39.1 / 46.8**	**60.0 / 69.4**	**30.4 / 56.8**	**45.6 / 73.2**	**56.4 / 83.0**	**71.9 / 94.0**	**12.9 / 16.9**	**33.0 / 41.5**	**47.8 / 58.7**	**68.6 / 82.0**

Critical node detection (IC / LT): remaining spread, % of nodes infected (↓)

	Power Grid (4,941 nodes)				PGP (10,680 nodes)				Gnutella31 (62,561 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%	1%	5%	10%	20%
HDA	26.5 / 28.9	24.0 / 27.1	21.8 / 25.4	18.6 / 22.3	26.4 / 31.9	23.3 / 28.5	20.8 / 25.6	18.0 / 21.6	31.3 / 39.0	26.0 / 35.0	21.9 / 30.8	17.4 / 22.9
BPD+R	26.7 / 29.1	24.1 / 27.5	21.9 / 25.8	18.5 / 22.4	26.4 / 32.0	– / –	21.2 / 25.8	18.0 / 21.6	31.4 / 39.1	26.4 / 35.5	22.5 / 31.4	– / –
CI+R	26.7 / 29.0	24.1 / 27.3	21.9 / 25.7	18.5 / 22.3	26.6 / 32.0	23.4 / 28.5	21.2 / 25.9	18.1 / 21.6	31.3 / 39.1	– / –	– / –	– / –
EI	26.8 / 29.2	24.5 / 27.9	22.1 / 26.0	19.1 / 23.2	26.5 / 32.0	23.5 / 28.8	21.1 / 25.9	18.2 / 22.2	32.0 / 39.4	26.4 / 35.3	22.1 / 31.1	17.9 / 23.7
Frontier	26.3 / 28.6	22.7 / 25.0	18.7 / 20.4	10.7 / **10.7**	26.7 / 32.0	23.4 / 27.9	20.2 / 23.6	14.2 / 14.5	31.3 / 39.0	26.0 / 34.2	21.6 / 28.6	16.0 / 17.6
Network World Model	**26.2 / 28.2**	**22.0 / 24.1**	**17.3 / 19.6**	**10.2 / 10.7**	**25.8 / 31.1**	**21.3 / 26.7**	**16.5 / 21.5**	**11.0 / 13.0**	**31.1 / 38.2**	**24.7 / 32.0**	**19.5 / 26.1**	**12.3 / 16.6**

Influence blocking (IC / CLT): rumor cascade size at blocker budget k (↓)

	Email-EU (1,005 nodes)					Gnutella24 (26,518 nodes)				
Method	10	20	30	40	50	10	20	30	40	50
RPS	60.2 / **70.9**	47.4 / 56.7	39.8 / 48.2	35.0 / 42.8	31.9 / 39.1	1520.6 / 1625.3	1311.6 / 1380.6	1151.5 / 1206.3	1029.6 / 1076.5	939.8 / 980.4
Reverse	61.0 / 73.1	49.2 / 57.9	42.1 / 49.9	37.2 / 44.8	34.5 / 42.1	1817.6 / 1961.8	1713.1 / 1856.4	1540.1 / 1676.8	1462.0 / 1587.7	1413.1 / 1535.7
Proximity	63.1 / 74.9	52.4 / 61.2	44.6 / 53.2	39.6 / 47.7	36.3 / 44.2	2338.9 / 2531.6	2289.9 / 2481.1	2239.7 / 2424.9	2196.5 / 2379.0	2141.1 / 2319.7
GreedyReplace	92.2 / 162.3	77.2 / 163.5	65.9 / 158.7	60.1 / 157.3	54.8 / 154.9	1551.8 / 1640.5	1364.9 / 1449.2	1216.8 / 1342.3	1088.2 / 1211.8	990.4 / 1072.4
SandIMIN	92.5 / 160.0	77.4 / 158.7	66.5 / 157.4	63.7 / 157.8	54.4 / 150.9	1564.9 / 1633.3	1391.2 / 1456.3	1228.9 / 1300.2	1106.4 / 1173.9	1017.8 / 1096.9
Network World Model	**59.4** / 71.7	**46.3 / 55.4**	**39.4 / 47.1**	**34.5 / 42.2**	**31.0 / 38.7**	**1503.3 / 1582.3**	**1275.0 / 1333.8**	**1113.6 / 1154.8**	**985.6 / 1023.0**	**896.0 / 924.7**

Epidemic control (SIR / SIS): attack rate, % of nodes ever infected (↓)

	Infectious SocioPatterns (10,972 nodes)				Oregon1 (10,670 nodes)			
Method	1%	5%	10%	20%	1%	5%	10%	20%
DAVA	**19.9** / 23.0	**7.4** / 8.3	**1.0 / 1.0**	**1.0 / 1.0**	7.9 / 9.6	**1.0 / 1.0**	**1.0 / 1.0**	**1.0 / 1.0**
NetShield+	21.2 / 24.9	17.3 / 20.6	12.8 / 15.6	7.4 / 8.9	4.3 / 5.4	2.1 / 2.3	1.8 / 1.9	1.6 / 1.7
GreedyWalk	20.9 / 24.6	16.5 / 19.8	12.3 / 14.9	7.0 / 8.5	4.8 / 5.9	2.5 / 2.8	1.9 / 2.1	1.6 / 1.7
EI	20.7 / 24.4	16.3 / 19.3	11.5 / 13.6	8.1 / 9.5	4.2 / 5.2	2.0 / 2.1	1.7 / 1.8	1.6 / 1.6
CI	20.7 / 24.3	16.1 / 18.9	12.4 / 14.6	6.9 / 8.1	4.2 / 5.2	2.0 / 2.1	1.7 / 1.8	1.6 / 1.6
Network World Model	20.4 / **19.5**	12.2 / **4.1**	**1.0 / 1.0**	**1.0 / 1.0**	**2.4 / 4.3**	**1.0 / 1.0**	**1.0 / 1.0**	**1.0 / 1.0**

Source localization (IC / LT): consistency of the recovered sources with the observation, 0 is perfect (↑)

Method	Cora-ML (2,810 nodes)	Power Grid (4,941 nodes)	Deezer (47,538 nodes)
LPSI	−0.173 / −0.130	−0.106 / −0.100	−0.176 / −0.174
Rumor Centrality	−0.237 / −0.257	−0.242 / −0.260	−0.190 / −0.200
Dynamic Age	−0.200 / −0.183	−0.234 / −0.274	−0.178 / −0.183
Infected-Degree	−0.173 / −0.154	−0.116 / −0.119	−0.178 / −0.186
SL-VAE	−0.370 / −0.437	−0.306 / −0.323	−0.314 / −0.240
Network World Model	**−0.139 / −0.106**	**−0.080 / −0.073**	**−0.147 / −0.145**

Cascade reconstruction (IC / LT): referee reward of the decoded history, 0 is perfect (↑)

Method	UCI Students (1,266 nodes)	CA-GrQc (5,242 nodes)
Reports	−0.852 / −0.946	−0.791 / −0.692
Steiner Tree	−0.901 / −0.978	−0.798 / −0.753
Jordan-Backward	−0.852 / −0.946	−0.791 / −0.692
DHREC	−1.464 / −1.525	−1.175 / −1.176
CRI	−0.885 / −0.995	−1.106 / −1.151
Network World Model	**−0.675 / −0.717**	**−0.566 / −0.566**

Cascade prediction: MSLE at the prediction horizon (↓)

Method	APS (30,000 nodes)	Taoke (29,711 nodes)	Digg (5,000 nodes)
Persistence	0.288	1.101	4.222
Szabo-Huberman	0.098	0.716	0.834
Hawkes	0.141	0.957	2.744
RPP	0.116	1.009	3.283
Weng-Communities	0.196	0.850	1.142
Network World Model	**0.078**	**0.544**	**0.774**

H.1 Cross-Network Transfer

Table 14 gives the per-budget results behind Figure 3a, together with the influence maximization and adaptive influence maximization transfers. For each task, we replay the algorithm designed on one network unmodified on the task's other networks and compare it with the algorithm designed directly on each target; the two agree within 0.1 percentage points in every completed cell.

Table 14: Generalizability of the designed algorithms. The algorithm designed on one graph is replayed unmodified on the task's other graphs. Designed on the target is that graph's own returned program, as in Tables 1 and 13. The NetHEPT adaptive program exceeded the 1,800 s wall-clock limit for one call on Digg at every budget.

Influence maximization (IC), program designed on Digg (↑)

	Network Science (1,589)				NetHEPT (15,229)			
Algorithm	1%	5%	10%	20%	1%	5%	10%	20%
Designed on the target	8.9	25.3	39.1	60.1	12.9	33.0	47.8	68.5
Transferred from Digg	8.9	25.3	39.1	60.0	12.9	33.0	47.8	68.6

Adaptive influence maximization (IC), program designed on NetHEPT (↑)

	Network Science (1,589)				Digg (116,893)			
Algorithm	1%	5%	10%	20%	1%	5%	10%	20%
Designed on the target	8.9	25.2	39.1	60.0	30.4	45.6	56.4	71.9
Transferred from NetHEPT	8.9	25.3	39.1	60.1	–	–	–	–

Critical node detection (IC), program designed on PGP (↓)

	Power Grid (4,941)				Gnutella31 (62,561)			
Algorithm	1%	5%	10%	20%	1%	5%	10%	20%
Designed on the target	26.2	22.1	17.4	10.2	31.1	24.7	19.5	12.3
Transferred from PGP	26.2	22.1	17.3	10.2	31.1	24.7	19.5	12.3

The one incomplete transfer is the adaptive influence maximization algorithm designed on NetHEPT, which exceeds the $1{,}800$ s per-call limit on the $116{,}893$-node Digg network; its cost, not its decision rule, fails to transfer.

I Additional Network World Model Experiments

I.1 World Model Setup

Data and model. The IC training suites are network-disjoint: BA-24 holds 24 Barabási-Albert networks with $N = 100$, and SBM-24 holds 24 stochastic block model networks with $N = 120$; LT is trained on BA-24. The default setting uses $w_{uv} = 1/\mathrm{indeg}(v)$. A random-weight setting draws $\mathrm{w}_{uv} \sim U(0.02, 0.4)$, so that the transmission probability cannot be reconstructed from the network structure, and is evaluated both with the weights visible to the model and with them hidden. Held-out BA and SBM networks are used for the decision-value tests, WS, ER, and SBM networks for topology shift, and BA networks up to $N = 1000$ for size transfer. Training and evaluation networks are disjoint whenever we make a cross-network claim. The model is the GraphSAGE encoder with the structured transition head of Section 3.2, at $34{,}113$ parameters.

Targets and metrics. For IC, each recorded transition stores a soft next-state target estimated from 30 simulator draws, together with one realized next state; for LT, the soft target is the analytic one-step conditional hazard. Probabilistic metrics are computed on at-risk nodes only: the KL divergence to the soft target and the Brier score $(y - \hat{y})^2$, and, on realized states, NLL and UROC. Multi-step quality uses the sampled rollout of Eq. (7) and reports the rollout Brier score and the relative bias of the final spread, $(\hat{R} - R)/R$, at $H = 10$. To measure whether the model preserves the effect of

an intervention, and not only the marginal state distribution, we compare counterfactual effects with the exact transition mechanism,

$$d_v^{\text{pred}} = \hat{p}_v(\boldsymbol{s}, a) - \hat{p}_v(\boldsymbol{s}, a'), \qquad d_v^{\text{exact}} = p_v^{\text{exact}}(\boldsymbol{s}, a) - p_v^{\text{exact}}(\boldsymbol{s}, a'), \tag{39}$$

and report the normalized MAE and the Pearson correlation on nodes with a non-zero exact effect. Using the exact mechanism avoids treating Monte Carlo noise as ground truth.

I.1.1 Scale and Topology Transfer

Table 15 freezes the BA-100 IC checkpoint and evaluates it without retraining, changing only the evaluation network. The model transfers to networks up to $10\times$ larger and to WS, ER, and SBM topologies: one-step KL stays between 0.0036 and 0.0063, and degradation appears in long-horizon spread calibration, not in one-step discrimination. These runs keep the edge weights visible to the model, so the structured head can read the transmission probability from its input.

Table 15: Frozen-checkpoint generalization across network size and topology. "Exact Brier" compares the predicted one-step probability with the analytic IC probability. The rollout column reports the model's Brier score and the noise floor of the exact process at $H = 10$, and spread bias reports the same pair. The counterfactual effect is measured against the analytic intervention effect. Arrows give the better direction; spread bias is signed, so closer to zero is better.

Shift	Target	KL (↓)	Exact Brier (↓)	AUROC (↑)	H=10 Brier (↓)	Spread bias (→ 0)	Effect MAE_{norm} (↓) / Pearson (↑)
None	BA-100	0.0059	0.00011	0.923	0.0083 / 0.0078	$+0.2\%$ / -0.4%	0.0095 / 1.0000
Size $2\times$	BA-200	0.0048	0.00002	0.934	0.0076 / 0.0072	$+1.6\%$ / -0.5%	0.0103 / 0.9999
Size $5\times$	BA-500	0.0043	0.00002	0.943	0.0079 / 0.0064	$+3.0\%$ / 0.0%	0.0098 / 0.9999
Size $10\times$	BA-1000	0.0041	0.00003	0.948	0.0075 / 0.0066	$+2.7\%$ / -0.3%	0.0129 / 0.9999
Topology	WS-100	0.0036	0.00000	0.932	0.0074 / 0.0073	-0.9% / -1.2%	0.0088 / 1.0000
Topology	ER-100	0.0051	0.00023	0.930	0.0093 / 0.0081	-2.5% / -0.8%	0.0131 / 0.9995
Topology	SBM-100	0.0063	0.00084	0.946	0.0113 / 0.0087	-5.0% / -0.9%	0.0299 / 0.9951

I.1.2 Transition Head and Action Conditioning

Comparing the structured transition head with a linear one in the world model reveals that encoding the known form of the process in the readout matters most over long rollouts: the linear head leaves one-step KL almost unchanged (0.0061 against 0.0059), but raises the spread bias at $H = 10$ from $+1.7\%$ to $+8.7\%$ and the normalized error of the predicted action effect from 0.0105 to 0.0398, so encoding the known form of the process matters over a horizon and not at one step. By freezing a checkpoint and changing only the evaluation network, the model transfers to networks ten times larger and to WS, ER, and SBM topologies with one-step KL between 0.0036 and 0.0063, and what degrades is long-horizon spread calibration.

We also evaluate the world model on two benchmark networks: Network Science under IC and Power Grid under IC. The metrics are those of Table 2, with two additions from the action-conditioning evaluation: the drop in one-step ΔF1 when the action channels are zeroed at evaluation, which is zero for a model that ignores its actions, and the Pearson correlation between the predicted effect of an action and the effect measured on the Monte Carlo simulator's counterfactual forks.

I.2 World Model Encoder Ablation

Table 16 compares three encoders against a GraphSAGE checkpoint trained on the data and with the settings of the end-to-end runs. Every row keeps the data, the structured head, and the training budget, and changes the backbone; because only GraphSAGE supports per-edge message conditioning, every row in this table is trained without it, so the GraphSAGE row is a retrained reference, not the checkpoint from the end-to-end runs. On both networks, the four encoders are within 0.002 ΔF1 of one another, and the final count bias of every rollout stays within 3%, so the structured head, not the encoder, carries the result.

Table 16: World-model encoders on Network Science. Each row changes only the encoder against the first row. Count bias is the relative bias of the final count of a free-running sampled rollout at the horizon; "ΔF1 drop" is the loss in one-step ΔF1 when the action channels are zeroed at evaluation; "Effect corr." is the Pearson correlation between the predicted and the measured effect of an action.

Variant	ΔF1 (↑)	Brier (↓)	Count bias	ΔF1 drop	Effect corr. (↑)	Train (min)
GraphSAGE	0.8765	0.0003	−0.5%	0.792	0.849	6.5
GCN	0.8750	0.0003	−1.4%	0.810	0.849	8.8
GATv2	0.8756	0.0004	−2.3%	0.812	0.849	11.4
GCNII	0.8745	0.0004	−1.6%	0.796	0.848	20.8

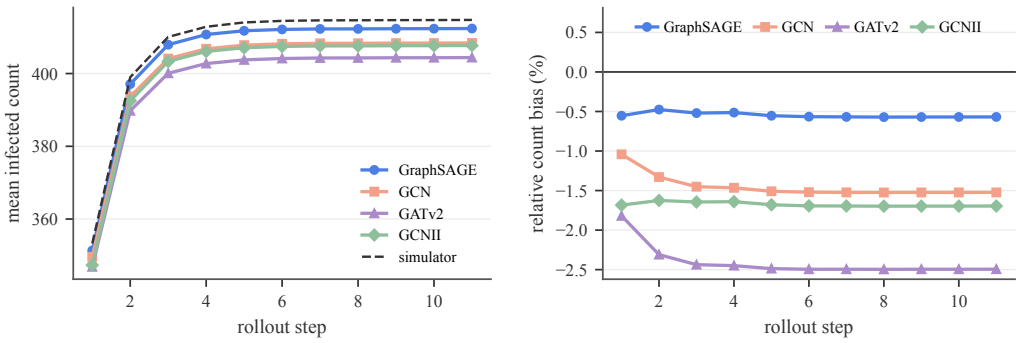


Figure 6: The four encoders on Network Science. Left: mean infected count of the free-running sampled rollout at each step, against the simulator (dashed). Right: the same rollouts as relative count bias by step; the bias settles within the first five steps on every backbone and does not grow with the horizon.

I.3 Error Against Horizon

Table 17 and Figure 7 report the relative count bias of the free-running rollout of the end-to-end checkpoints by step, on the 50 held-out episodes of the main runs. On both networks, the bias settles within the first two to five steps and then stays put, so one-step error does not compound over a rollout. Power Grid's rollout overcounts by about 3%, a constant offset the search tolerates because it shifts every candidate's score equally.

Table 17: Relative count bias of the free-running sampled rollout against the Monte Carlo simulator, by rollout step, for the checkpoints used in the end-to-end runs, on the 50 held-out episodes of each run. Zero means no bias; a magnitude that grows with the step would mean compounding error.

Network	Step 1	Step 2	Step 5	Step 10
Network Science	+0.0%	+0.1%	+0.1%	+0.1%
Power Grid	+1.6%	+2.4%	+3.2%	+3.3%

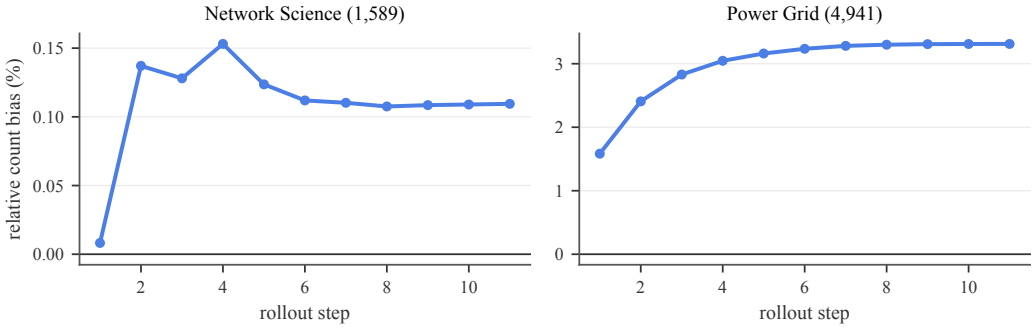


Figure 7: Relative count bias of the free-running sampled rollout by step on the two benchmark networks, on the 50 held-out episodes of the main runs. The bias settles within the first few steps on both networks and does not grow.

J Examples of Designed Algorithms

We highlight four algorithms returned by the algorithm design loop for four different tasks: critical node detection, influence maximization, source localization, and influence blocking. Each is the program returned for its task and network under IC, and it is reported in Tables 1 and 13. For each program, we provide a short description, pseudocode, results, and a Python source-code excerpt. Shaded lines in the pseudocode mark the steps that set the program apart from the classical methods in the library (Appendix C), the description sets the same steps in bold, and the excerpt shades the code lines that implement them.

J.1 Critical Node Detection: Ensemble Cavity-Gradient Removal

Description. The program removes nodes greedily, but it ranks them by a quantity the dismantling methods in the library do not compute: a gradient conditioned on the outbreak, which estimates to first order how many infections at the horizon the removal of each node prevents. The forward part **runs non-backtracking cavity messages from the outbreak sources for H steps**, where the message on $u \to v$ is the probability that u is infected without help from v. The backward part **propagates adjoints back through the same H steps**, so a node's score counts both its own infection and the infections its outgoing messages cause downstream. HDA, CI and EI rank nodes without regard to where the outbreak starts, and Frontier looks only at the ring around the sources; the gradient accounts for both the source location and the horizon. The program **computes the gradient under two models**, the given transmission probabilities and a copy capped at 0.9 that tempers edges transmitting almost surely, and ranks nodes by an **ensemble score that weights the capped model more and adds a bonus for disagreement between the two**, so a node one model undervalues is not averaged away. Recomputing the gradient after every removal would take k passes, so the program **removes six nodes per gradient pass and shrinks the scores of each removed node's neighbors** in between, which keeps one batch from spending two removals on the same route.

Algorithm 2 Ensemble cavity-gradient removal, designed for critical node detection on PGP under IC.

Require: network $\mathcal{G}$ with transmission probabilities p_{uv}, outbreak sources $\mathbb{S}$, budget k, horizon H

$\bar{p}_{uv} \leftarrow \min(p_{uv}, 0.9)$; $\mathbb{R} \leftarrow \emptyset$ {capped probabilities; removed nodes}
while $|\mathbb{R}| < k$ **do**
  $\boldsymbol{g}^{\text{cap}} \leftarrow \textsc{CavityGradient}(\bar{p}, \mathbb{R})$, $\boldsymbol{g}^{\text{orig}} \leftarrow \textsc{CavityGradient}(p, \mathbb{R})$
  for up to six removals **do**
    $\text{score}_v \leftarrow 0.65\, g_v^{\text{cap}} + 0.35\, g_v^{\text{orig}} + 0.10\, |g_v^{\text{cap}} - g_v^{\text{orig}}|$ for every $v \notin \mathbb{S} \cup \mathbb{R}$
    $v^\star \leftarrow \arg\max_v \text{score}_v$; $\mathbb{R} \leftarrow \mathbb{R} \cup \{v^\star\}$
    in both models, shrink the neighbors' scores: $g_w \leftarrow g_w(1 - 0.55\, p_{v^\star w})$ for out-neighbors and $g_w \leftarrow g_w(1 - 0.35\, p_{w v^\star})$ for in-neighbors
  end for
end while
return remove every $v \in \mathbb{R}$ at $t = 0$
function $\textsc{CavityGradient}(p, \mathbb{R})$, on $\mathcal{G}$ with the nodes in $\mathbb{R}$ deleted:
  forward: run non-backtracking cavity messages from $\mathbb{S}$ for H steps and read each node's probability of being infected by step H
  backward: propagate adjoints through the H steps; g_v is v's infection probability plus the downstream infections its outgoing messages cause
return $\boldsymbol{g}$

Results. At the 10% budget on PGP, the program leaves 16.5% of the nodes infected, against 20.2% for Frontier, the strongest baseline, and the same program gives the 5% and 20% cells (21.3% and 11.0%). Replayed unchanged, it leaves 17.3% infected on Power Grid and 19.5% on Gnutella31 at the 10% budget, matching the programs designed on those networks (Table 14).

Excerpt. The selection loop, with the ensemble score and the neighbor discount:

```
while len(chosen) < count:
    capped_scores = cavity_gradient(capped_p)
    actual_scores = cavity_gradient(actual_p)
    batch = min(6, count - len(chosen))
    for _ in range(batch):
        # Modestly favor cuts whose value differs between models,
        # rather than allowing averaging to conceal a useful gateway.
        scores = (
            0.65 * capped_scores
            + 0.35 * actual_scores
            + 0.10 * np.abs(capped_scores - actual_scores)
        )
        scores[~eligible] = -np.inf
        node = int(np.argmax(scores))
        if not np.isfinite(scores[node]):
            break
        chosen.append(node)
        alive[node] = 0.0
        eligible[node] = False
        out_ids = outgoing_order[
            outgoing_ptr[node]:outgoing_ptr[node + 1]
        ]
        in_ids = incoming_order[
            incoming_ptr[node]:incoming_ptr[node + 1]
        ]
        for model_scores, model_p in (
            (capped_scores, capped_p),
            (actual_scores, actual_p),
        ):
            np.multiply.at(
                model_scores,
```

```
                v[out_ids],
                1.0 - 0.55 * model_p[out_ids],
            )
            np.multiply.at(
                model_scores,
                u[in_ids],
                1.0 - 0.35 * model_p[in_ids],
            )
```

J.2 Influence Maximization: Stratified Horizon-Limited Coverage with Exchange Sweeps

Description. The program builds on reverse-reachable sampling, the mechanism behind IMM, OPIM and SubSIM, and changes it in three ways. Its reverse-reachable sets are **horizon-limited**: each keeps only the nodes that reach its root within H reverse steps of sampled live edges, matching the reward, which counts activations within H steps, whereas the classical estimators count activations at any time. Its roots are **stratified**: every node is the root of exactly r sets instead of a random number of them, which removes the variance that uniformly drawn roots add to the coverage estimate. After greedy maximum coverage, an **exchange sweep** revisits the seeds in increasing order of **unique coverage**, the number of sets only that seed covers, and **replaces a seed only when a non-seed covers more by a fixed margin**. A second sweep, bounded to $4{,}096$ seeds, recomputes unique coverage after the first sweep's swaps and revisits the seeds that are now weakest.

Algorithm 3 Stratified horizon-limited coverage with exchange sweeps, designed for influence maximization on Digg under IC.

Require: network $\mathcal{G}$ with N nodes and transmission probabilities p_{uv}, budget k, horizon H

1: sample $2{,}048$ pilot sets for the mean set size $\bar{m}$; $r \leftarrow 20 \cdot \min(40, \max(1, \lfloor 3 \times 10^7/(N\bar{m}) \rfloor))$
2: stratified roots: make every node the root of exactly r reverse-reachable sets, rN sets in total
3: horizon-limited sets: for each root, keep the nodes that reach it within H reverse steps of sampled live edges
4: $\mathbb{A} \leftarrow \emptyset$; repeat k times: add the node covering the most uncovered sets (ties by weighted out-degree)
5: **for** sweep limit $L \in (k,\ \min(4096, k))$ **do**
6: order the seeds by unique coverage, the number of sets only that seed covers, and take the L lowest
7: **for** each such seed s **do**
8: remove s; $s' \leftarrow$ the non-seed with the largest marginal coverage
9: keep s' only if its marginal coverage exceeds that of s by more than $\max(1, \lfloor 0.005\, r \rfloor)$ sets, otherwise restore s
10: **end for**
11: **if** no seed changed **then** stop
12: **end for**
13: **return** seed every $v \in \mathbb{A}$ at $t = 0$

Results. At the 10% budget on Digg, the program activates 56.5% of the nodes, against 51.8% for OPIM and SubSIM, and the same program gives all four budget cells (30.4%, 45.8%, 56.5% and 69.4%). Replayed unchanged on Network Science and NetHEPT, it matches the programs designed on those networks to within 0.1 points (Table 14).

Excerpt. The two exchange sweeps:

```
margin = max(1, int(0.005 * repetitions))
# Preserve the parent's full sweep and its exact acceptance threshold.
# Recompute removal costs once to revisit opportunities created later.
for sweep_limit in (k, min(4096, k)):
    losses = np.empty(k, dtype=np.int64)
    for i, node in enumerate(seeds):
```

```
        rr = cover_ids[cover_ptr[node]:cover_ptr[node + 1]]
        losses[i] = np.count_nonzero(multiplicity[rr] == 1)
    reconsider = np.argsort(losses, kind="stable")[:sweep_limit]
    changes = 0
    for position in reconsider:
        old = seeds[int(position)]
        rr = cover_ids[cover_ptr[old]:cover_ptr[old + 1]]
        multiplicity[rr] -= 1
        exposed = rr[multiplicity[rr] == 0]
        self._update_scores(exposed, 1, scores, rr_ptr, members, n)
        selected[old] = False
        candidates = np.where(selected[priority], -1, scores[priority])
        new = int(priority[np.argmax(candidates)])
        if scores[new] <= scores[old] + margin:
            new = old
        rr = cover_ids[cover_ptr[new]:cover_ptr[new + 1]]
        fresh = rr[multiplicity[rr] == 0]
        self._update_scores(fresh, -1, scores, rr_ptr, members, n)
        multiplicity[rr] += 1
        selected[new] = True
        seeds[int(position)] = new
        changes += int(new != old)
    if changes == 0:
        break
```

J.3 Source Localization: Cavity Consistency Optimization with Component Repair

Description. Most localization methods in the library score each infected node by a centrality or a propagated label and return the top k. The program instead fits a source set to the observation through **a differentiable forward model**: $F(\boldsymbol{x})$ runs non-backtracking cavity messages for ten steps from source probabilities $\boldsymbol{x}$ and returns each node's infection probability, and the loss is the squared error between $F(\boldsymbol{x})$ and the observation. Starting from the LPSI answer, it **relaxes the source indicator to fractions that sum to k, descending the loss with an integrality penalty that grows over three stages**. The binary set is then improved by an **exchange search in two branches, ranked by gradient and curvature, whose disagreements are recombined**, and a **component repair** step targets observed infections the current sources leave unexplained.

Algorithm 4 Cavity consistency optimization with component repair, designed for source localization on Power Grid under IC.

Require: network $\mathcal{G}$ with transmission probabilities p_{uv}, observed infection indicator $\boldsymbol{y}$, number of sources k

$\mathbb{E} \leftarrow \{v : y_v = 1\}$; $\boldsymbol{x} \leftarrow 0.8\,\frac{k}{|\mathbb{E}|}\mathbf{1}_{\mathbb{E}} + 0.2\,\mathbf{1}_{\mathrm{LPSI}}$ {start from LPSI's k sources}

$F(\boldsymbol{x})$: infection probabilities after ten steps of non-backtracking cavity messages from source probabilities $\boldsymbol{x}$; $L(\boldsymbol{x}) = \|F(\boldsymbol{x}) - \boldsymbol{y}\|^2$

for $\lambda \in (0,\ 0.04,\ 0.15)$ **do**

  projected gradient descent on $L(\boldsymbol{x}) + \lambda \sum_v x_v(1 - x_v)$ over $\{\boldsymbol{x} \in [0,1]^{\mathbb{E}} : \sum_v x_v = k\}$, with the gradient from an adjoint pass through F and a backtracking line search

  every seven steps, round $\boldsymbol{x}$ to its k largest entries and keep the best binary set $\boldsymbol{b}$

end for

exchange search in two branches, adjacency-aware and global: swap 1 to 32 sources ranked by gradient plus a stochastic curvature estimate, accepting only if L decreases; recombine the branches' disagreements

component repair: group the observed-infected nodes that $F(\boldsymbol{b})$ predicts uninfected into connected components, and move low-value sources to one representative per component, accepting only if L decreases

a short adjacency-aware exchange search

return the k sources in $\boldsymbol{b}$

Results. On Power Grid the recovered sources reach a consistency of -0.080, against -0.106 for LPSI, the strongest baseline (Table 13). Replayed unchanged, the program reaches -0.138 on Cora-ML and -0.148 on Deezer, against -0.173 and -0.176 for LPSI, and within 0.001 of the programs designed on those networks.

Excerpt. The relaxation with the growing integrality penalty:

```
seeds = 0.8 * uniform + 0.2 * best
step = 0.35
for penalty in (0.0, 0.04, 0.15):
    for iteration in range(28):
        predicted, context = self._forward(seeds, trace=True)
        residual = predicted - y
        objective = float(np.dot(residual, residual)) + penalty * float(
            np.sum(seeds * (1.0 - seeds))
        )
        gradient = self._adjoint(seeds, context, 2.0 * residual)
        gradient += penalty * (1.0 - 2.0 * seeds)
        accepted, trial_step = False, step
        for _ in range(10):
            proposal = np.zeros(self.n)
            proposal[eligible] = self._project(
                seeds[eligible] - trial_step * gradient[eligible], count
            )
            value = self._loss(proposal, y) + penalty * float(
                np.sum(proposal * (1.0 - proposal))
            )
            directional = float(np.dot(gradient, proposal - seeds))
            if value <= objective + 1e-4 * directional + 1e-9:
                seeds = proposal
                step = min(trial_step * 1.2, 3.0)
                accepted = True
                break
            trial_step *= 0.5
        if not accepted:
            step = max(trial_step, 1e-5)
        if iteration % 7 == 6 or not accepted:
            rounded = self._binary(seeds, eligible, count, tie)
            loss = self._loss(rounded, y)
            if loss < best_loss:
                best, best_loss = rounded, loss
        if not accepted:
            break
```

J.4 Influence Blocking: Differentiable Competitive-Cavity Blocking

Description. RPS and the other sampling-based blockers in the library (Table 7) sample prevention sets and cover them greedily, and the first candidate of this search did the same. The returned program replaces sampling with a **competitive mean-field recursion** that tracks, for every arc, the probability that it has not yet transmitted, and lets the counter cascade take a node first when both cascades reach it in the same step, the positive-dominance tie-break of our IC runs (Appendix E.2). Because the recursion is differentiable, the program **computes the gradient of the rumor size with respect to every counter-seed probability in one backward pass** and chooses all k seeds jointly by **projected gradient descent on a relaxation, with an integrality penalty that grows over three stages**. Taking the k largest entries of the relaxed solution can lose value, so it also rounds by **moving mass between pairs of fractional entries in whichever direction lowers the rumor size**, from two starting points. Last, a **swap repair recomputes the gradient after vacating a seed's seat**. A replacement can look redundant while the outgoing seed still protects its route, so the gradient taken with the seat empty ranks candidates that the original gradient misses.

Algorithm 5 Differentiable competitive-cavity blocking, designed for influence blocking on Gnutella24 under IC.

Require: network $\mathcal{G}$ with transmission probabilities p_{uv}, rumor sources $\mathbb{S}_N$, k counter-seeds, horizon H

$J(\boldsymbol{x})$: expected rumor size after H steps of a competitive mean-field recursion in which the counter cascade, seeded with probabilities $\boldsymbol{x}$, takes a node first when both cascades reach it in the same step

$\nabla J(\boldsymbol{x})$ from one adjoint pass back through the H steps

$\mathbb{E} \leftarrow \mathbb{V} \setminus \mathbb{S}_N$; $x_v \leftarrow k/|\mathbb{E}|$ for $v \in \mathbb{E}$

for $(\lambda, T) \in \big((0, 44),\ (0.4, 12),\ (1.5, 12)\big)$ **do**

  T steps of projected gradient descent on $J(\boldsymbol{x}) + \lambda\,(k - \|\boldsymbol{x}\|^2)$ over $\{\boldsymbol{x} \in [0,1]^{\mathbb{E}} : \sum_v x_v = k\}$ with a backtracking line search

  periodically round $\boldsymbol{x}$ to its k largest entries and keep the best binary set $\boldsymbol{b}$

end for

for the relaxed solutions after the first and the last stage **do**

  restrict to the $3k$ largest entries; move mass between the largest and the smallest fractional entry in whichever direction lowers J, until the solution is binary; keep it if it beats $\boldsymbol{b}$

end for

for two passes **do**

  for each of the three seeds with the least benefit under ∇J: vacate its seat, recompute ∇J with the seat empty, and try the six best additions under each of the two gradients

  apply the best swap that lowers J; stop if none does

end for

return seed every v with $b_v = 1$ in the counter cascade at $t = 0$

Results. With $k = 30$ counter-seeds on Gnutella24 the rumor reaches 1,113.6 nodes, against 1,151.5 for RPS, the strongest baseline. The same program gives all five budget cells and is ahead of RPS at every budget (Table 13). Replayed unchanged on Email-EU, it is also ahead of RPS at every budget, with rumor sizes from 59.4 nodes at $k = 10$ to 30.8 at $k = 50$, against 60.2 to 31.9.

Excerpt. The competitive recursion, in which the counter cascade claims a node before the rumor; the lines that record values for the backward pass are omitted:

```
for _ in range(self.steps):
    middle = np.maximum(
        theta - self.weight * positive_front[self.tail], eps
    )
    following = np.maximum(
        middle - self.weight * negative_front[self.tail], eps
    )
    middle_product = self._products(middle)
    following_product = self._products(following)
    new_positive = susceptible_initial * (
        old_product - middle_product
    )
    new_negative = susceptible_initial * (
        middle_product - following_product
    )
    objective += float(new_negative.sum())
    theta = following
    old_product = following_product
    positive_front = new_positive
    negative_front = new_negative
```